

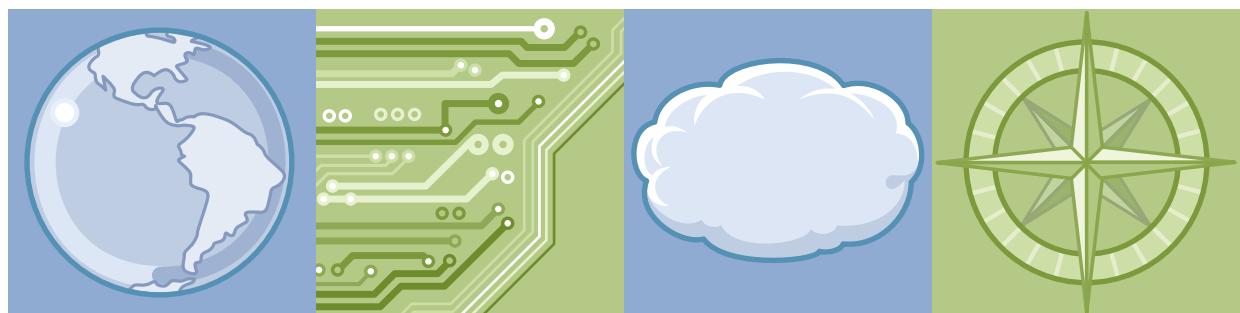


# IBM Training

Student Notebook

## **IBM Integration Bus V10 Application Development I**

Course code WM666 / ZM666 ERC 1.0



IBM Systems Middleware

## 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®	Redbooks®
Tivoli®	Watson™	WebSphere®
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.

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

## August 2015 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

Trademarks .....	xv
Course description .....	xix
Agenda .....	xxi
<b>Unit 1. Introduction to IBM Integration Bus .....</b>	<b>1-1</b>
Unit objectives .....	1-2
IBM Integration Bus .....	1-3
IBM Integration Bus themes .....	1-5
IBM Integration Bus features .....	1-6
IBM Integration Bus main components .....	1-7
IBM Integration Bus runtime components .....	1-9
IBM Integration Bus integration services .....	1-11
IBM Integration Toolkit .....	1-12
IBM Integration web user interface .....	1-13
IBM Integration Bus command utilities .....	1-14
IBM Integration API .....	1-15
IBM Integration Bus operation modes .....	1-16
Supported hardware and environments .....	1-17
Supported software .....	1-18
Technology components and prerequisites .....	1-19
Flexible IBM MQ topologies .....	1-20
Connectivity to external systems .....	1-21
IBM Integration Bus web services .....	1-22
IBM Integration Bus and publish/subscribe support .....	1-23
Decision management in IBM Integration Bus .....	1-24
IBM Integration Bus runtime security .....	1-25
Operational management and performance .....	1-26
Connectivity to WebSphere eXtreme Scale grids .....	1-27
IBM Integration Bus in the Cloud .....	1-28
Using IBM Integration Bus to provide a REST API .....	1-29
IBM Integration Bus Industry Packs .....	1-30
Conversion from WebSphere Enterprise Service Bus .....	1-31
Supported migration paths .....	1-32
Unit summary .....	1-33
Checkpoint questions .....	1-34
Checkpoint answers .....	1-35
<b>Unit 2. Application development fundamentals .....</b>	<b>2-1</b>
Unit objectives .....	2-2
Main components .....	2-3
Main components interaction .....	2-4
IBM Integration Bus integration node .....	2-5
Integration servers .....	2-6
IBM Integration Bus runtime environment .....	2-7
Message flows .....	2-9
Message model and tree .....	2-11

Message parsing overview . . . . .	2-12
IBM Integration Bus administration tools . . . . .	2-14
Command utilities . . . . .	2-15
Basic IBM Integration Bus commands . . . . .	2-16
IBM Integration Toolkit . . . . .	2-17
Team development . . . . .	2-19
IBM Integration Toolkit Welcome page . . . . .	2-20
Default configuration for development . . . . .	2-21
Integration Development perspective at start . . . . .	2-22
Integration Development views . . . . .	2-24
Integration Nodes view . . . . .	2-25
Projects, applications, and libraries (1 of 2) . . . . .	2-26
Projects, applications, and libraries (2 of 2) . . . . .	2-28
Static libraries . . . . .	2-29
Shared libraries . . . . .	2-30
Application, library, and project maintenance . . . . .	2-31
Creating a Working set . . . . .	2-32
Exporting to create a Project Interchange file . . . . .	2-34
Importing a Project Interchange file . . . . .	2-35
Broker schemas (namespaces) . . . . .	2-36
Message flow nodes . . . . .	2-38
Message flow node properties . . . . .	2-40
Message processing nodes overview . . . . .	2-42
Message transport nodes . . . . .	2-43
Message transport-related nodes . . . . .	2-44
Other message processing nodes (1 of 2) . . . . .	2-45
Other message processing nodes (2 of 2) . . . . .	2-46
Using patterns for simplified development . . . . .	2-47
Pattern concepts . . . . .	2-48
Patterns Explorer . . . . .	2-50
Getting patterns from the GitHub repository . . . . .	2-51
Generating a pattern instance (1 of 2) . . . . .	2-52
Generating a pattern instance (2 of 2) . . . . .	2-53
Configuring pattern parameters . . . . .	2-54
Generated artifacts . . . . .	2-55
Integrated testing with IBM Integration Toolkit . . . . .	2-56
IBM Integration Toolkit Flow exerciser . . . . .	2-57
Flow exerciser simple test . . . . .	2-58
Flow exerciser toolbar . . . . .	2-59
Running the Flow exerciser . . . . .	2-60
Viewing message paths and connections . . . . .	2-61
Flow exerciser example . . . . .	2-62
Unit summary . . . . .	2-63
Checkpoint questions . . . . .	2-64
Checkpoint answers . . . . .	2-65
Exercise 1 . . . . .	2-66
Exercise objectives . . . . .	2-67
 <b>Unit 3. Creating message flow applications.</b> . . . . .	 3-1
Unit objectives . . . . .	3-2
Quick starts . . . . .	3-3

IBM Integration Bus Tutorials Gallery .....	3-4
Creating an application .....	3-5
Creating a message flow .....	3-6
Adding nodes to the message flow .....	3-7
Wiring nodes .....	3-8
Saving the message flow .....	3-9
Deploying solutions .....	3-10
Creating a BAR file .....	3-11
BAR file contents .....	3-13
Creating the BAR file in the Integration Toolkit .....	3-14
Adding components to the BAR file .....	3-15
Modifying a BAR file .....	3-16
BAR file message flow properties .....	3-17
BAR file node properties .....	3-18
Message flow packaging and validation .....	3-19
Deploying in the Integration Toolkit .....	3-20
Deployment details .....	3-21
Deployment Log .....	3-22
Managing the integration node in the Integration Toolkit .....	3-23
Using the IBM Integration web user interface .....	3-24
Integration server status example .....	3-25
Deploy by using the IBM Integration web user interface .....	3-26
IBM Integration web user interface BAR file properties .....	3-27
Unit summary .....	3-28
Checkpoint questions .....	3-29
Checkpoint answers .....	3-30
Exercise 2 .....	3-31
Exercise objectives .....	3-32
 <b>Unit 4. Connecting to IBM MQ.....</b>	 <b>4-1</b>
Unit objectives .....	4-2
IBM MQ functional overview .....	4-3
Messages .....	4-4
Queues .....	4-5
Queue manager .....	4-6
IBM MQ Explorer .....	4-7
IBM MQ Explorer default views .....	4-8
IBM MQ connectivity options with IBM Integration Bus .....	4-9
Advantages of messaging with IBM MQ .....	4-10
MQInput node .....	4-11
MQOutput node .....	4-12
MQReply node .....	4-13
MQGet node .....	4-14
MQGet node processing outline .....	4-15
MQGet example .....	4-16
Flexible IBM MQ topologies .....	4-17
IBM Integration Bus features that require IBM MQ .....	4-18
Creating a local queue manager with IBM MQ Explorer (1 of 5) .....	4-19
Creating a local queue manager with IBM MQ Explorer (2 of 5) .....	4-20
Creating a local queue manager with IBM MQ Explorer (3 of 5) .....	4-21
Creating a local queue manager with IBM MQ Explorer (4 of 5) .....	4-22

Creating a local queue manager with IBM MQ Explorer (5 of 5) .....	4-23
Queue manager content view .....	4-24
Creating a local queue with IBM MQ Explorer (1 of 3) .....	4-25
Creating a local queue with IBM MQ Explorer (2 of 3) .....	4-26
Creating a local queue with IBM MQ Explorer (3 of 3) .....	4-27
Creating a default queue manager for an integration node .....	4-28
Specifying a default queue manager for an integration node .....	4-29
IBM Integration Bus system queues .....	4-30
Creating the IBM Integration Bus system queues .....	4-31
IBM MQ client .....	4-32
Client connections to IBM MQ .....	4-33
MQEndpoint policy .....	4-34
MQEndpoint policy document .....	4-35
Creating an MQEndpoint policy .....	4-36
Generating an MQ Policy in the IBM Integration Toolkit (1 of 2) .....	4-37
Generating an MQ Policy in the IBM Integration Toolkit (2 of 2) .....	4-38
Creating an MQEndpoint policy in the IBM Integration web user interface .....	4-39
Integration Registry .....	4-40
Testing message flows by using IBM MQ Explorer .....	4-41
Testing message flows with RFHUtil (SupportPac IH03) .....	4-42
Message flow result dependencies .....	4-43
Message flow error behavior with IBM MQ .....	4-44
Message flow error behavior: BOTHRESH exceeded .....	4-45
Unit summary .....	4-46
Checkpoint questions .....	4-47
Checkpoint answers .....	4-48
Exercise 3 .....	4-49
Exercise objectives .....	4-50

<b>Unit 5. Controlling the flow of messages .....</b>	<b>5-1</b>
Unit objectives .....	5-2
Message parsing overview .....	5-3
Logical message model .....	5-4
IBM Integration Bus message assembly .....	5-5
Example data representations .....	5-6
Logical message model example .....	5-7
ResetContentDescriptor node .....	5-8
Message routing nodes .....	5-9
Filter node .....	5-10
ESQL .....	5-12
Accessing message trees with ESQL correlation names .....	5-13
ESQL module .....	5-14
Sample ESQL for Filter node .....	5-15
ESQL editor .....	5-16
ESQL editor options .....	5-17
ESQL content assist and validation .....	5-18
Route node .....	5-19
Accessing message trees with XPath .....	5-21
XPath Expression Builder .....	5-22
Adding terminals to the Route node .....	5-23
RouteToLabel and Label nodes .....	5-24

FlowOrder node .....	5-25
FlowOrder node example .....	5-26
Subflows .....	5-27
Message flow nodes that are used with subflows .....	5-28
Subflow that is defined in a .subflow file .....	5-30
Subflow that is defined in a .msgflow file .....	5-32
Subflows in shared libraries .....	5-33
Subflow properties in the BAR file .....	5-34
Example: Generic error handler subflow .....	5-35
Unit summary .....	5-36
Checkpoint questions .....	5-37
Checkpoint answers .....	5-38
Exercise 4 .....	5-39
Exercise objectives .....	5-40
<b>Unit 6. Modeling the data.....</b>	<b>6-1</b>
Unit objectives .....	6-2
Message modeling .....	6-3
Message model object properties .....	6-4
IBM Integration Bus message modeling .....	6-5
Parser types .....	6-6
Model definition files .....	6-7
Message definition importers .....	6-8
Specification, model, logical message tree .....	6-9
Custom wire format (binary or CWF) .....	6-10
Logical message and CWF format .....	6-11
Tagged and delimited string format (text or TDS) .....	6-12
Logical message and TDS format .....	6-13
Data Format Description Language (DFDL) .....	6-14
DFDL data support (1 of 2) .....	6-16
DFDL data support (2 of 2) .....	6-17
Example: Delimited text data .....	6-18
Example: DFDL schema .....	6-19
Example: DFDL schema (short form) .....	6-20
DFDL support in IBM Integration Bus .....	6-21
When should you use DFDL? .....	6-22
DFDL object model .....	6-23
DFDL schema editor .....	6-24
Creating a DFDL schema file with the New Message Model wizard .....	6-25
Creating a DFDL model .....	6-26
Wizard options for creating a DFDL model .....	6-27
Creating a DFDL model by using guided authoring .....	6-28
Creating a DFDL model by using the editor (1 of 2) .....	6-29
Creating a DFDL model by using the editor (2 of 2) .....	6-30
Created DFDL schemas .....	6-31
Helper DFDL schemas .....	6-32
New Message Model wizard example .....	6-33
DFDL schema editor (1 of 3) .....	6-34
DFDL schema editor (2 of 3) .....	6-35
DFDL schema editor (3 of 3) .....	6-36
DFDL schema editor toolbar actions .....	6-37

Testing the DFDL schema . . . . .	6-38
Testing a DFDL model within the editor (1 of 4) . . . . .	6-39
Testing a DFDL model within the editor (2 of 4) . . . . .	6-40
Testing a DFDL model within the editor (3 of 4) . . . . .	6-41
Testing a DFDL model within the editor (4 of 4) . . . . .	6-42
Test parse logical instance . . . . .	6-43
Test parse failure . . . . .	6-44
Debugging a DFDL model test failure (1 of 2) . . . . .	6-45
Debugging a DFDL model test failure (2 of 2) . . . . .	6-46
Trace console . . . . .	6-47
Test serialize model . . . . .	6-48
DFDL points of uncertainty . . . . .	6-49
Getting started with DFDL . . . . .	6-50
Assigning the DFDL parser to a message . . . . .	6-51
DFDL message tree . . . . .	6-52
Creating an XML message model . . . . .	6-53
Parsing XML data . . . . .	6-54
Configuring nodes for an XMLNSC parser (1 of 2) . . . . .	6-55
Configuring nodes for an XMLNSC parser (2 of 2) . . . . .	6-56
Message validation . . . . .	6-57
Parse timing option on input nodes . . . . .	6-58
Parsing and validation . . . . .	6-59
Default and fixed values for elements . . . . .	6-61
Organizing message models . . . . .	6-62
Unit summary . . . . .	6-63
Checkpoint questions . . . . .	6-64
Checkpoint answers . . . . .	6-65
Exercise 5 . . . . .	6-66
Exercise objectives . . . . .	6-67
 <b>Unit 7. Processing file data . . . . .</b>	 <b>7-1</b>
Unit objectives . . . . .	7-2
File processing nodes . . . . .	7-3
FileInput node . . . . .	7-4
FileInput node: Basic algorithm . . . . .	7-5
FileInput node: Successful processing . . . . .	7-6
Configuring the FileInput node . . . . .	7-7
Record detection . . . . .	7-8
Record detection options . . . . .	7-9
Record detection examples . . . . .	7-10
FileInput node error handling . . . . .	7-11
FileInput node that uses FTP . . . . .	7-12
FileOutput node . . . . .	7-13
FileOutput node processing . . . . .	7-14
FileOutput node Request tab . . . . .	7-16
Appending records with the FileOutput node . . . . .	7-17
FileRead node . . . . .	7-18
Configuring the FileRead node . . . . .	7-19
Unit summary . . . . .	7-20
Checkpoint questions . . . . .	7-21
Checkpoint answers . . . . .	7-22

Exercise 6 . . . . .	7-23
Exercise objectives . . . . .	7-24
<b>Unit 8. Using problem determination tools and help resources . . . . .</b>	<b>8-1</b>
Unit objectives . . . . .	8-2
Message flow result dependencies . . . . .	8-3
Basic error handling . . . . .	8-4
ExceptionList . . . . .	8-5
Options for catching exceptions . . . . .	8-6
TryCatch node . . . . .	8-7
Message flow error behavior with TryCatch . . . . .	8-8
Controlled throwing and logging of exceptions . . . . .	8-10
TryCatch node example . . . . .	8-11
Message flow error tips . . . . .	8-12
Problem determination tools . . . . .	8-13
Local error log (syslog) . . . . .	8-14
IBM Integration Bus activity logs . . . . .	8-15
Writing Activity logs to files . . . . .	8-16
Using a command to create an ActivityLog configurable service . . . . .	8-17
Other logs . . . . .	8-18
IBM Integration Toolkit Flow exerciser . . . . .	8-19
User trace (1 of 2) . . . . .	8-20
User trace (2 of 2) . . . . .	8-21
Trace node . . . . .	8-22
Configuring the Trace node . . . . .	8-23
Reporting current trace options for an integration node . . . . .	8-24
Message flow debugger . . . . .	8-25
Breakpoints . . . . .	8-27
Debug perspective example . . . . .	8-28
Debug view options . . . . .	8-30
Message flow debugger configuration . . . . .	8-31
Setting the Java debug port (1 of 2) . . . . .	8-32
Setting the Java debug port (2 of 2) . . . . .	8-33
Identifying projects that contain source artifacts . . . . .	8-34
Enabling and disabling the message flow debugger . . . . .	8-35
Testing message flows with the Unit Test Client . . . . .	8-36
Test Client events . . . . .	8-38
Test Client configuration . . . . .	8-40
Specifying the Test Client deployment location (1 of 2) . . . . .	8-41
Specifying the Test Client deployment location (2 of 2) . . . . .	8-42
Test and debug by using the Test Client (1 of 2) . . . . .	8-44
Test and debug by using the Test Client (2 of 2) . . . . .	8-45
Test Client component trace . . . . .	8-46
Test Client trace events . . . . .	8-47
Enabling a Test Client component trace . . . . .	8-48
Sample component trace: Successful flow . . . . .	8-49
Sample component trace: Failed flow . . . . .	8-50
RFHUtil (SupportPac IH03) . . . . .	8-51
IBM Knowledge Center . . . . .	8-52
Integrated Help . . . . .	8-53
Help on wizards . . . . .	8-54

Unit summary .....	8-55
Checkpoint questions .....	8-56
Checkpoint answers .....	8-57
Exercise 7 .....	8-58
Exercise objectives .....	8-59
Exercise 8 .....	8-60
Exercise objectives .....	8-61
<b>Unit 9. Mapping messages with the Graphical Data Mapping editor.....</b>	<b>9-1</b>
Unit objectives .....	9-2
Message maps .....	9-3
Message map implementation .....	9-5
Mapping node .....	9-6
Graphical Data Mapping editor (1 of 2) .....	9-7
Graphical Data Mapping editor (2 of 2) .....	9-8
Map navigation .....	9-9
Map editor .....	9-10
Graphical Data Mapping editor toolbar .....	9-11
Creating a map in the IBM Integration Toolkit (1 of 3) .....	9-12
Creating a map in the IBM Integration Toolkit (2 of 3) .....	9-13
Creating a map in the IBM Integration Toolkit (3 of 3) .....	9-14
Creating transforms in a map .....	9-15
Core mapping transforms .....	9-16
Database transforms .....	9-17
Structural mapping transforms .....	9-18
Transform properties .....	9-19
Automatically mapping input to output elements .....	9-20
Primary and supplementary inputs and connections .....	9-22
Using XPath 2.0 in the Graphical Data Mapping editor .....	9-23
User-defined elements .....	9-24
Considerations when adding user-defined elements .....	9-25
Mapping JSON object messages .....	9-26
Tips for choosing a Transform type for JSON messages .....	9-28
Properties mapping .....	9-29
Mapping the Environment tree .....	9-30
Deploying a message map .....	9-31
Message maps and shared libraries .....	9-32
Choosing a graphical data map type .....	9-33
Troubleshooting maps .....	9-34
Unit summary .....	9-35
Checkpoint questions .....	9-36
Checkpoint answers .....	9-37
<b>Unit 10. Referencing a database in a message flow application .....</b>	<b>10-1</b>
Unit objectives .....	10-2
Using databases in message flows .....	10-3
Configure database access for the integration node .....	10-4
Verifying ODBC connections .....	10-5
DatabaseInput node .....	10-6
Database node .....	10-7
Node properties for database access .....	10-8

Transaction support . . . . .	10-9
Transaction coordinator options . . . . .	10-10
ESQL SELECT compared with SQL SELECT . . . . .	10-11
PASSTHRU example . . . . .	10-12
THE (SELECT ...) . . . . .	10-13
SELECT (ITEM ...) . . . . .	10-14
DatabaseRoute node . . . . .	10-15
Configuring the DatabaseRoute node . . . . .	10-16
DatabaseRoute node query elements . . . . .	10-17
DatabaseRoute node filter expressions . . . . .	10-18
DatabaseRetrieve node . . . . .	10-19
Using databases in a Mapping node . . . . .	10-20
Using New Database Definition File wizard (1 of 2) . . . . .	10-21
Using New Database Definition File wizard (2 of 2) . . . . .	10-22
Database service discovery . . . . .	10-23
Unit summary . . . . .	10-24
Checkpoint questions . . . . .	10-25
Checkpoint answers . . . . .	10-26
Exercise 9 . . . . .	10-27
Exercise objectives . . . . .	10-28
<b>Unit 11. Using Compute nodes to transform messages . . . . .</b>	<b>11-1</b>
Unit objectives . . . . .	11-2
Transformation options in IBM Integration Bus . . . . .	11-3
Compute node . . . . .	11-4
ESQL editor . . . . .	11-6
ESQL content assist and validation . . . . .	11-7
Compute node ESQL . . . . .	11-8
ESQL module and integration node schema . . . . .	11-9
Compute node ESQL module skeleton . . . . .	11-10
OrderMsg sample message . . . . .	11-11
Sample ESQL syntax to transform OrderMsg . . . . .	11-12
ESQL debugger . . . . .	11-13
Some ESQL syntax . . . . .	11-14
Inserting, updating, and deleting fields . . . . .	11-15
Special ESQL data types for tree structures . . . . .	11-16
Moving REFERENCE variables . . . . .	11-17
Field references in tree structures . . . . .	11-18
Deleting and reordering fields . . . . .	11-19
CREATE statement . . . . .	11-20
NULL value . . . . .	11-21
Procedures . . . . .	11-22
Variables . . . . .	11-24
Variable scope, lifetime, and sharing (1 of 2) . . . . .	11-25
Variable scope, lifetime, and sharing (2 of 2) . . . . .	11-26
Using a Compute node to access a database . . . . .	11-28
Reusable ESQL . . . . .	11-29
JavaCompute node . . . . .	11-30
Getting started with the JavaCompute node . . . . .	11-31
Using the Java wizard (1 of 2) . . . . .	11-33
Using the Java wizard (2 of 2) . . . . .	11-34

JavaCompute templates . . . . .	11-35
JavaCompute template example . . . . .	11-36
JavaCompute node basics (1 of 2) . . . . .	11-37
JavaCompute node basics (2 of 2) . . . . .	11-38
Message assembly and standard trees . . . . .	11-39
Using traversal methods to browse message trees . . . . .	11-40
Using XPath in JavaCompute node . . . . .	11-41
Updating the tree . . . . .	11-42
JavaCompute node at run time . . . . .	11-43
JavaCompute node database access example . . . . .	11-44
Combine ESQL and Java . . . . .	11-45
Example: Calling Java method from ESQL . . . . .	11-46
Unit summary . . . . .	11-47
Checkpoint questions (1 of 2) . . . . .	11-48
Checkpoint answers (1 of 2) . . . . .	11-49
Checkpoint questions (2 of 2) . . . . .	11-50
Checkpoint answers (2 of 2) . . . . .	11-51
Exercise 10 . . . . .	11-52
Exercise objectives . . . . .	11-53
<b>Unit 12. Processing JMS, HTTP, and web service messages . . . . .</b>	<b>12-1</b>
Unit objectives . . . . .	12-2
Java Message Service (JMS) . . . . .	12-3
JMS architecture . . . . .	12-4
IBM Integration Bus support for JMS . . . . .	12-6
JMS nodes . . . . .	12-7
Making the JMS provider client available to the JMS nodes . . . . .	12-8
JMSSInput node . . . . .	12-9
Parsing the JMS message payload . . . . .	12-10
JMS message tree . . . . .	12-12
JMSOutput node . . . . .	12-14
Serializing the JMS output message . . . . .	12-15
JMSReceive node . . . . .	12-16
JMSReply node . . . . .	12-18
JMS transformation nodes . . . . .	12-19
JMS transactional processing . . . . .	12-20
Securing JMS message flows . . . . .	12-21
Web services overview . . . . .	12-22
SOAP . . . . .	12-23
Web Services Description Language (WSDL) . . . . .	12-24
IBM Integration Bus and web services . . . . .	12-26
Message flow as HTTP/HTTPS client . . . . .	12-27
Message flow as service provider . . . . .	12-28
IBM Integration Bus and web services configuration . . . . .	12-29
HTTP nodes . . . . .	12-30
SOAP nodes . . . . .	12-31
SOAP over JMS . . . . .	12-32
Unit summary . . . . .	12-33
Checkpoint questions . . . . .	12-34
Checkpoint answers . . . . .	12-35

<b>Unit 13. Preparing for production .....</b>	<b>13-1</b>
Unit objectives .....	13-2
Applications and libraries at run time .....	13-3
Applications and libraries on the integration node .....	13-4
Resource isolation with applications .....	13-5
Shared libraries at run time .....	13-6
Deploying updates to shared libraries .....	13-7
Creating BAR files .....	13-8
mqsipackagebar command .....	13-9
mqsicreatebar command .....	13-10
Integration node properties .....	13-11
Configuring for production .....	13-13
Environment-dependent message flows .....	13-15
Example: UDP determines route in flow .....	13-17
Promoted properties .....	13-18
Runtime version information in IBM Integration Bus .....	13-20
Embedding version information in message flows .....	13-21
Runtime version information: Embedding keywords .....	13-22
Linking with version control systems .....	13-23
Accounting and statistics .....	13-24
Statistics data details .....	13-25
Message flow statistics in IBM Integration Bus .....	13-26
Governance and policy-driven flows .....	13-27
DecisionService node .....	13-28
EndpointLookup node .....	13-29
RegistryLookup node .....	13-30
Monitoring support in IBM Integration Bus .....	13-31
Generate monitoring events in the Integration Toolkit .....	13-32
Configure event monitoring by using commands .....	13-33
Record and replay .....	13-34
Record and replay features .....	13-36
Performance tuning .....	13-37
Common causes of performance problems .....	13-38
IBM Integration Bus tuning options .....	13-39
Multiple copies of a message flow .....	13-41
Scaling message throughput .....	13-43
Tuning the integration node .....	13-45
Tuning the IBM MQ queue manager .....	13-46
Tuning databases .....	13-48
Detecting and removing obstructions .....	13-50
Unit summary .....	13-52
Checkpoint questions .....	13-53
Checkpoint answers .....	13-54
Exercise 11 .....	13-55
Exercise objectives .....	13-56
<b>Unit 14. Course summary .....</b>	<b>14-1</b>
Unit objectives .....	14-2
Course learning objectives .....	14-3
Course learning objectives .....	14-4
To learn more on the subject .....	14-5

Unit summary .....	14-6
<b>Appendix A. ESQL examples.</b> .....	<b>A-1</b>
<b>Appendix B. List of abbreviations.</b> .....	<b>B-1</b>
<b>Appendix C. Resource guide.</b> .....	<b>C-1</b>
<b>Bibliography.</b> .....	<b>D1</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®	Redbooks®
Tivoli®	Watson™	WebSphere®
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.

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







# Course description

## IBM Integration Bus V10 Application Development I

**Duration:** 5 days

### Purpose

IBM Integration Bus provides connectivity and universal data transformation in heterogeneous IT environments. It enables businesses of any size to eliminate point-to-point connections and batch processing, regardless of operating system, protocol, and data format. This course teaches you how to use IBM Integration Bus to develop, deploy, and support message flow applications. These applications use various messaging topologies to transport messages between service requesters and service providers, and also allow the messages to be routed, transformed, and enriched during processing. In this course, you learn how to construct applications to transport and transform data. The course also explores how to control the flow of data by using various processing nodes, and how to use databases and maps to transform and enrich data during processing. You also learn how to construct data models by using the Data Format Description Language (DFDL).

### Audience

This course is designed for experienced integration specialists and senior-level developers with experience in application development, messaging middleware applications, and transport protocols such as HTTP and FTP.

### Prerequisites

Before taking this course, you should have:

- A basic understanding of current IT technologies such as Structured Query Language (SQL), Extensible Markup Language (XML), Java, and XML Path language (XPath)
- An understanding of the business needs of your organization
- A basic understanding of transport protocols such as HTTP and FTP, and message-oriented middleware such as Java Message Service (JMS) and IBM MQ

### Objectives

After completing this course, you should be able to:

- Describe the features and uses of the IBM Integration Bus
- Develop, deploy, and test message flow applications
- Generate message flow applications from predefined patterns
- Use IBM Integration Bus problem determination aids to diagnose and solve development and runtime errors
- Describe the function and appropriate use of IBM Integration Bus processing nodes
- Write basic Extended Structured Query Language and Java programs to transform data
- Use the IBM Graphical Data Mapping editor to transform data
- Define, use, and test simple XML and Data Format Description Language (DFDL) data models
- Describe supported transport protocols and how to call them in message flows

## **Contents**

- IBM Integration Bus application design
- Troubleshooting

## **Curriculum relationship**

Prerequisite for WM676/ZM676, IBM Integration Bus V10 Application Development II

# Agenda

## Day 1

- Course introduction
- Unit 1. Introduction to IBM Integration Bus
- Unit 2. Application development fundamentals
- Exercise 1. Importing and testing a message flow
- Unit 3. Creating message flow applications
- Exercise 2. Creating a message flow application

## Day 2

- Unit 4. Connecting to IBM MQ
- Exercise 3. Connecting to IBM MQ
- Unit 5. Controlling the flow of messages
- Exercise 4. Adding flow control to a message flow application
- Unit 6. Modeling the data

## Day 3

- Exercise 5. Creating a DFDL model
- Unit 7. Processing file data
- Exercise 6. Processing file data
- Unit 8. Using problem determination tools and help resources
- Exercise 7. Using problem determination tools

## Day 4

- Exercise 8. Implementing explicit error handling
- Unit 9. Mapping messages with the Graphical Data Mapping editor
- Unit 10. Referencing a database in a message flow application
- Exercise 9. Referencing a database in a map
- Unit 11. Using Compute nodes to transform messages

## Day 5

- Exercise 10. Transforming data by using the Compute and JavaCompute nodes
- Unit 12. Processing JMS, HTTP, and web service messages
- Unit 13. Preparing for production
- Exercise 11. Creating a runtime-aware message flow
- Unit 14. Course summary



# Unit 1. Introduction to IBM Integration Bus

## What this unit is about

This unit introduces IBM Integration Bus, including its components, functions, and business value. The unit also summarizes the options for extending IBM Integration Bus and the migration path for WebSphere Enterprise Service Bus applications.

## What you should be able to do

After completing this unit, you should be able to:

- Describe the features and functions of IBM Integration Bus
- Describe the business value of IBM Integration Bus
- Describe the IBM Integration Bus architecture and components
- Identify the IBM Integration Bus editions

## How you will check your progress

- Checkpoint questions

## References

IBM Knowledge Center



## Unit objectives

After completing this unit, you should be able to:

- Describe the features and functions of IBM Integration Bus
- Describe the business value of IBM Integration Bus
- Describe the IBM Integration Bus architecture and components
- Identify the IBM Integration Bus editions

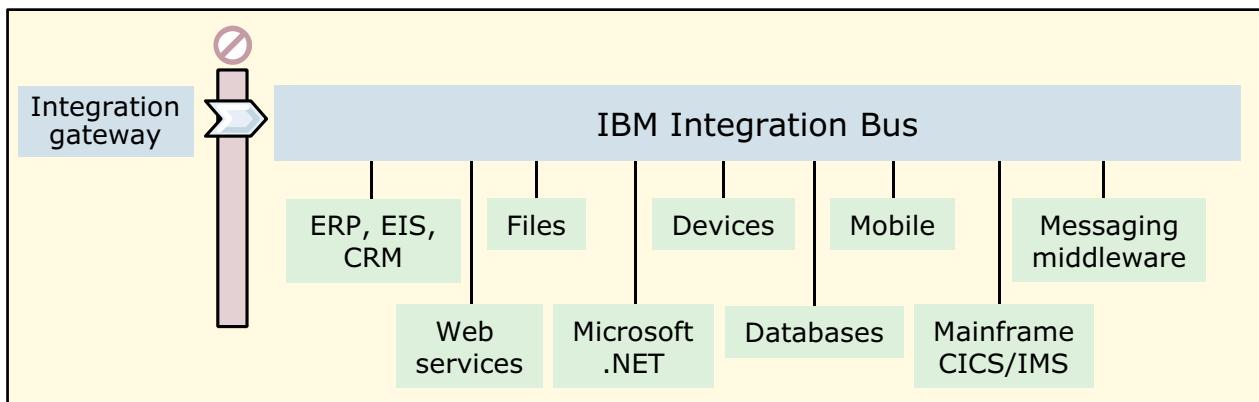
© Copyright IBM Corporation 2015

Figure 1-1. Unit objectives

WM666 / ZM6661.0

### Notes:

## IBM Integration Bus



- IBM Integration Bus provides connectivity across enterprise systems, applications, and data
  - Avoids rewrites in response to new integration requirements
  - Simplifies maintenance by reducing expensive coupling
  - Provides flexibility, which adds anonymity between data producers and consumers
  - Adds insight into applications and the business value that they bring

© Copyright IBM Corporation 2015

Figure 1-2. IBM Integration Bus

WM666 / ZM6661.0

### Notes:

Enterprise systems consist of many logical endpoints such as off-the-shelf applications, services, packaged applications, web applications, devices, appliances, and custom built software.

Endpoints expose a set of inputs and outputs, which include messaging middleware such as IBM MQ and JMS, protocols such as TCP/IP, HTTP, FTP, and SMTP, and databases such as DB2 and SQL Server. Endpoints also include formats such as flat files, COBOL), XML, industry standards (SWIFT, EDI, HL7), and user-defined data.

IBM Integration Bus connects these endpoints in the following ways:

- IBM Integration Bus avoids rewrites in response to new integration requirements
- IBM Integration Bus simplifies maintenance by reducing expensive coupling
- IBM Integration Bus flexibility adds anonymity between producers and consumers of data
- IBM Integration Bus adds insight into applications and business value they bring
- IBM Integration Bus delivers a comprehensive integration solution

IBM Integration Bus connects a wide range of applications, services, and systems across heterogeneous IT environments. It provides the visibility and control capabilities that are needed to

support critical business activities such as monitoring, auditing, process management, and analytics.

IBM Integration Bus can help to:

- Rapidly enable business insight to be applied to in-flight data
- Accelerate creation of integration services for business process management (BPM)
- Increase operational awareness and control over workload
- Gain visibility and insight of integration in application environments

IBM Integration Bus represents IBM's strategic ESB offering. IBM Integration Bus provides tools that helps convert WebSphere Enterprise Service Bus assets and so they can run on IBM Integration Bus.

## IBM Integration Bus themes

- Simple and productive
  - Quickly develop, deploy, manage, and migrate integration solutions
- Universal and independent
  - Connects to a range of different systems
  - Universal connectivity includes standards, *de facto* standards, industry, and custom systems
- Industry specific and relevant
  - Provides industry relevant connectivity packs to solve domain-specific problems
  - Industry-specific processing nodes, solution-oriented patterns, and user-oriented tooling
- Dynamic and intelligent
  - Allows the creation of dynamic solutions that provide business insight
  - Flexible configuration tools, analysis of data and intelligence
- High performing and scalable
  - Provides hardware, software, and technology neutral connectivity options
  - Works on the widest possible range of hardware, software, and virtualized environments

© Copyright IBM Corporation 2015

Figure 1-3. IBM Integration Bus themes

WM666 / ZM6661.0

### Notes:

The process to deploy and configure IBM Integration Bus so that an integration developer can use the IBM Integration Toolkit to start creating applications is simple and quick to complete.

IBM Integration Bus provides a universal integration capability that addresses a wide range of integration scenarios. These scenarios include web services such as SOAP and REST, messaging, database, file, ERP systems, mobile, physical devices, email, custom systems and more.

IBM Integration Bus supports various data standards such as EDI and SWIFT and connects to wide range of operating systems.

IBM Integration Bus allows the creation of dynamic solutions that provide business insight and flexible configuration.

IBM Integration Bus provides hardware, software, and technology neutral connectivity options.

## IBM Integration Bus features

- Transform and route data from anywhere, to anywhere
  - Supports a wide range of protocols and data formats
  - Includes comprehensive operations to route, filter, transform, enrich, monitor, distribute, decompose, sequence, correlate, and detect
  - Converts transport protocols between a requester and a service
  - Handles business events from disparate sources
  - Implements transformation by using graphical mapping, Java, ESQL, and XSL
  - Publish/subscribe with IBM MQ or MQTT
- Patterns provide reusable solutions that encapsulate a tested approach to solving a common architecture, design, or deployment task
- Operational management and performance
  - Provides administration and systems management options for developed solutions
  - Offers performance of traditional transaction processing environments
  - Web tools for real-time performance statistics
  - Integration with software products from IBM and other vendors that provide related management and connectivity services

© Copyright IBM Corporation 2015

Figure 1-4. IBM Integration Bus features

WM666 / ZM6661.0

### Notes:

IBM Integration Bus simplifies application connectivity, routes, and transforms messages, simplifies programming, and provides operational management and performance tools.

IBM Integration Bus includes features for universal connectivity to provide a flexible and dynamic infrastructure. These features include tools to help with the conversion of WebSphere Enterprise Service Bus assets so they can run on IBM Integration Bus.

IBM Integration Bus includes comprehensive operations and supports a wide range of protocols so that messages can be transformed and routed from anywhere, to anywhere.

IBM Integration Bus has extensive administration and systems management facilities included web tools for real-time performance statistics.

## IBM Integration Bus main components

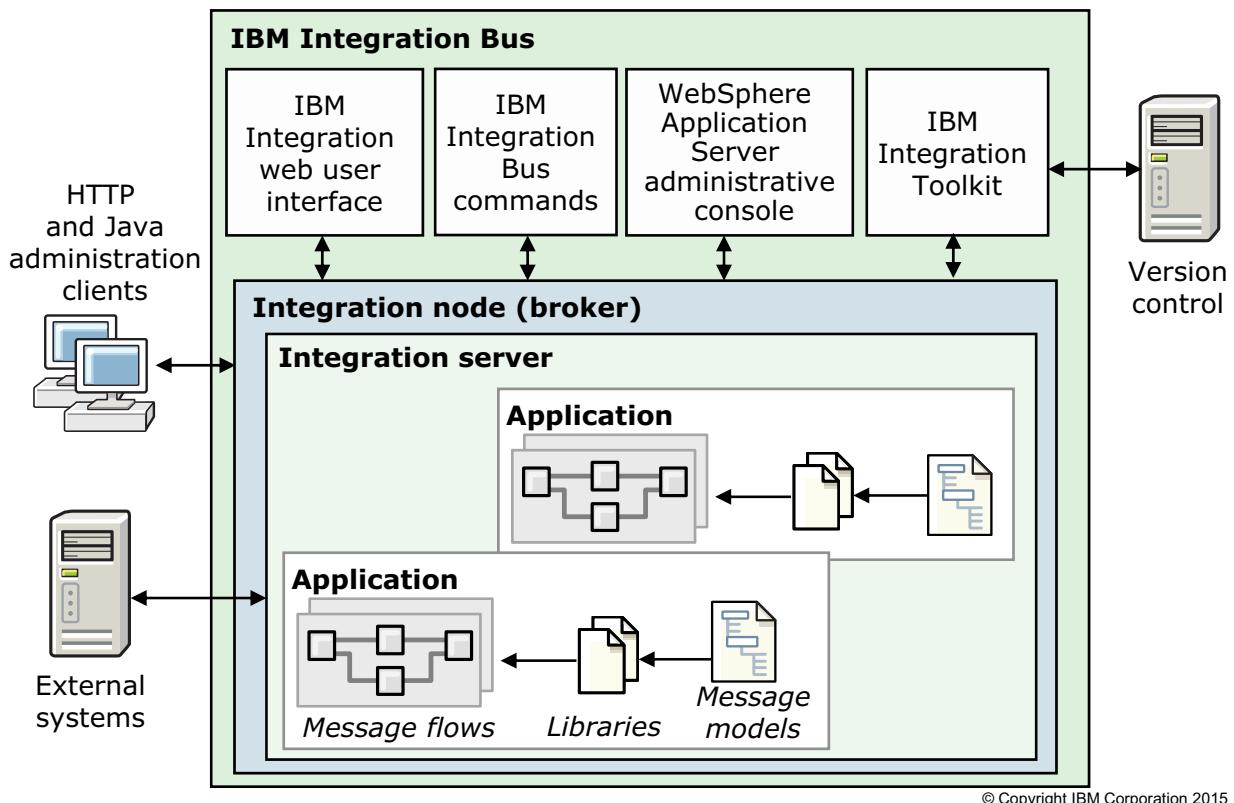


Figure 1-5. IBM Integration Bus main components

WM666 / ZM6661.0

### Notes:

IBM Integration Bus includes components for development and runtime administration.

The runtime engine of IBM Integration Bus is the *integration node*, also known as the integration broker. The integration node processes in-flight messages that are based on *message flows* and *messages*. The message flow controls the type and sequence of operations on the incoming messages. Logical *message trees* describe the structure of the data as it is processed.

Message flows are run on *integration servers* to provide isolation and scalability. Message flows can interact with external systems such as web services and databases.

An *application* is a container for all the resources that are required to create a solution. A *library* is typically, a reusable, logical grouping of related code, data, or both that an application references.

As shown in the figure, various administrative tools can be used to manage the integration node and monitor message flows and message content. The IBM Integration web user interface and commands are used to manage integration nodes, integration servers, and message flows at run time.

You create message flows in the IBM Integration Toolkit, which is an integrated development and administration console. It can connect with external source control applications for team development.

The IBM Integration Toolkit, IBM Integration web user interface, and commands communicate directly to one or more integration nodes. Each integration node can contain one or more *integration servers*, which can run one or more message flows.



## IBM Integration Bus runtime components

- Integration node
  - Routes, transforms, and enriches in-flight messages as determined by message flows and message models
  - Can be many integration nodes, each running on separate systems to provide protection against failure or separate the work
- Integration server
  - Named grouping of message flows that are assigned to an integration node
  - Each integration server is a separate operating system process, which provides isolated runtime environments for a set of deployed message flow applications
- Message flow applications
  - Describe the application connectivity logic, which defines the exact path that the data takes in the integration node, and the processing that is applied to it by the message processing nodes in that flow
  - Reference message models that describe the data

© Copyright IBM Corporation 2015

Figure 1-6. IBM Integration Bus runtime components

WM666 / ZM6661.0

### Notes:

The runtime engine of IBM Integration Bus is the *integration node*, also known as the integration broker. The integration node routes, transforms, and enriches in-flight messages based on the message flows and message models.

You can create integration nodes on every operating system that IBM Integration Bus supports. An integration node is a system service on Windows, a daemon process on UNIX, or a started task on z/OS. It controls processes that run message flows.

There can be many integration nodes, and each can be running on a different system. This architecture provides protection against failure, and can separate work across different divisions in a business.

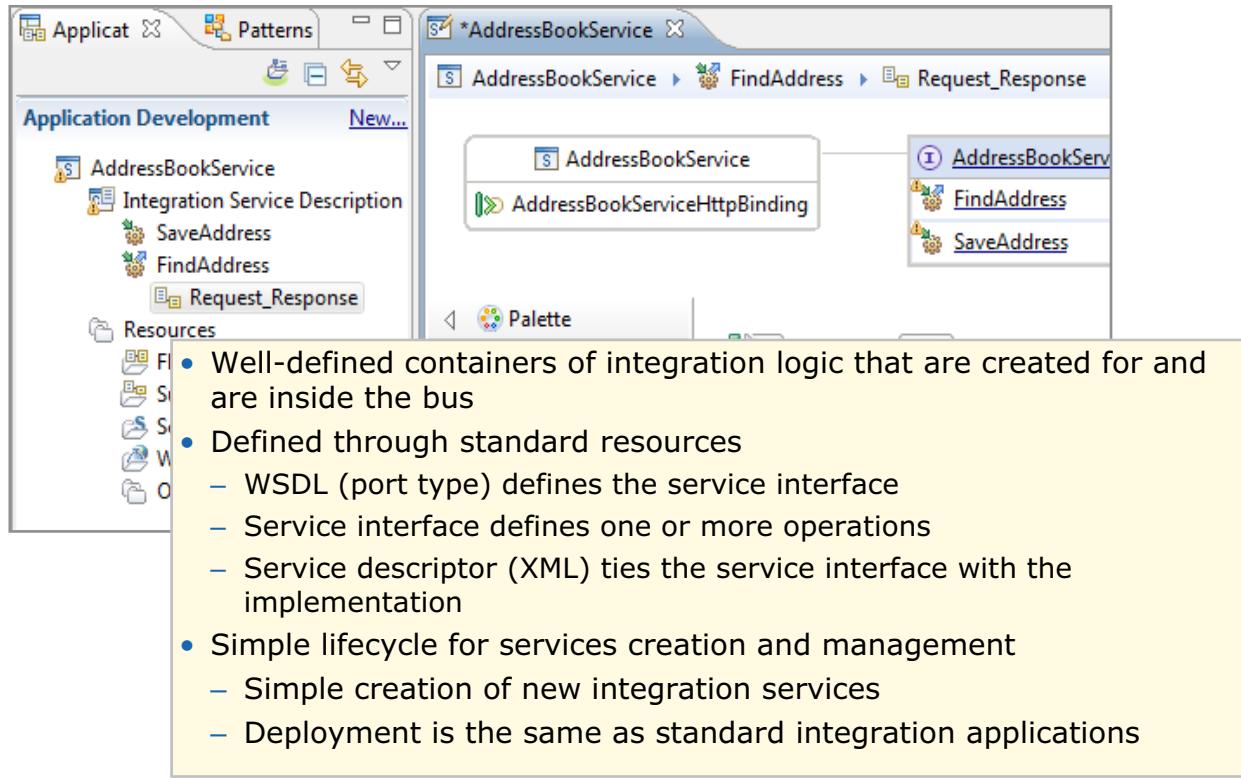
An *integration server* is a named grouping of message flows that are assigned to an integration node. Each integration server is started as a separate operating system process, providing an isolated runtime environment for a set of deployed message flows.

In IBM Integration Bus, the *message flow* defines the sequence of operations on a message. The actions are defined in terms of the message format, its content, and the results of individual actions along the message flow as determined by the message flow processing nodes and connections.

A message flow can route messages from a sender to a recipient based on the content of the message. The direction of the flow is from the source application to the destination application. In other words, it is assumed that the destination wants, expects, and handles all messages sent to it.



## IBM Integration Bus integration services



© Copyright IBM Corporation 2015

Figure 1-7. IBM Integration Bus integration services

WM666 / ZM6661.0

### Notes:

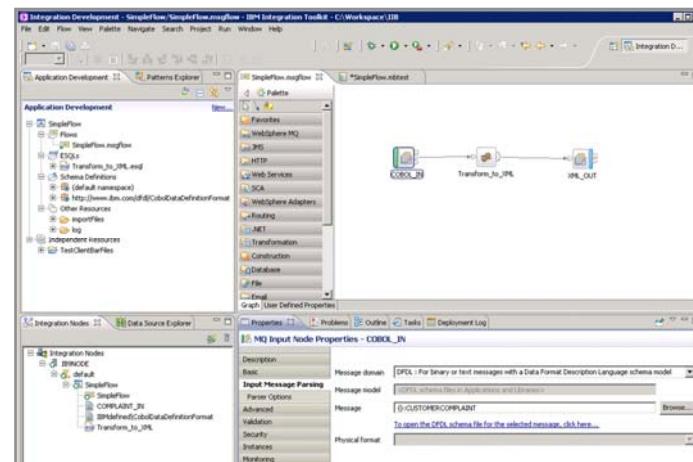
In IBM Integration Bus, an integration service is a specialized application with a defined interface that acts as a container for a web services solution. It contains message flows to implement the specified web service operations. The interface is defined through a WSDL file.

When you implement an integration service, you can deploy it to an IBM Integration Bus integration server. You can start and stop the deployed service as you would an application. A web service consumer can interrogate the deployed service to return its interface.



## IBM Integration Toolkit

- An integrated development environment and graphical user interface that is based on Eclipse
- A single perspective for compiling, testing, deploying, and fixing message flows
- Connects to one or more integration nodes to which the message flow applications are deployed
- Connects to patterns galleries for getting started quickly
- Runs on Microsoft Windows and Linux on x86



© Copyright IBM Corporation 2015

Figure 1-8. IBM Integration Toolkit

WM666 / ZM6661.0

### Notes:

The IBM Integration Toolkit is an integrated development environment and graphical user interface that is based on Eclipse. Application developers work in separate instances of the IBM Integration Toolkit to develop applications, integration services, libraries, message models, and message flows.

The IBM Integration Toolkit runs on Windows and Linux and uses the local file system for its information storage. In the Integration Toolkit, you can complete the following tasks:

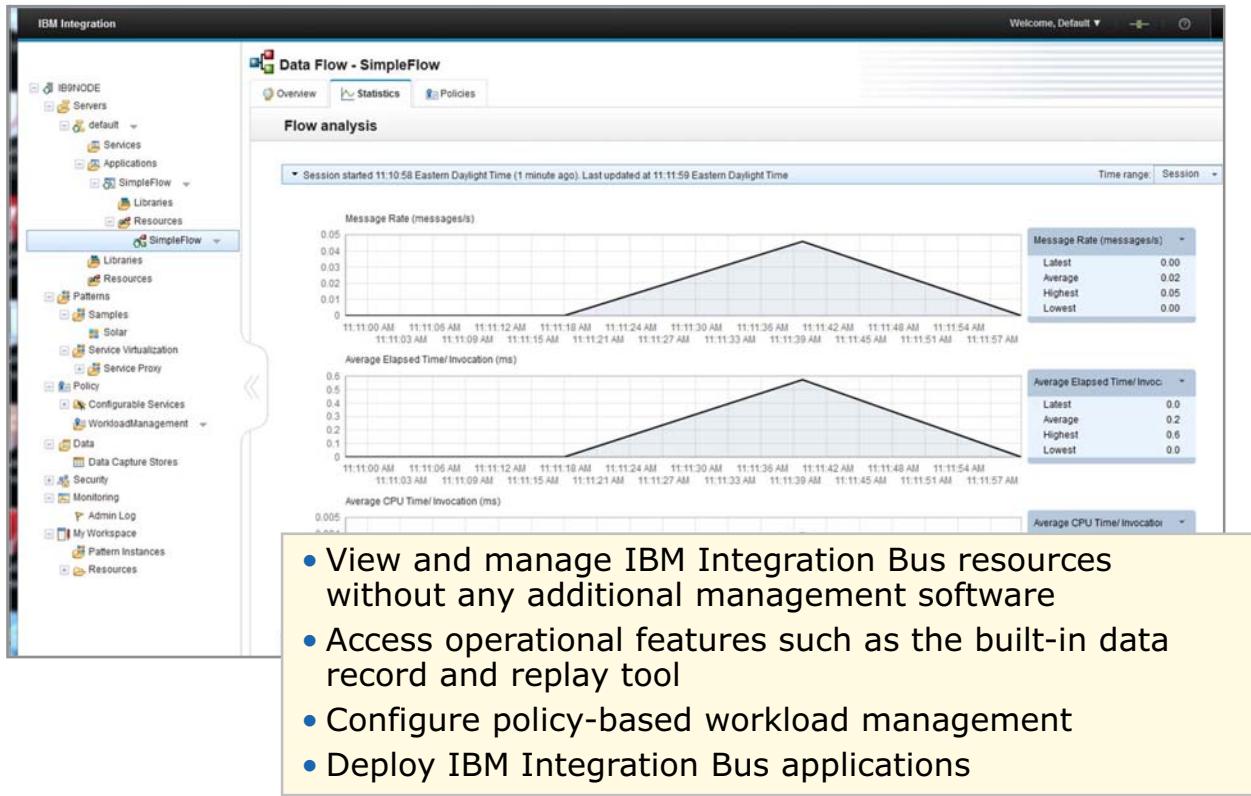
- Define applications, message flows, and message flow components
- Define and import message definitions
- Deploy message flows and message models to integration nodes
- Control log entries that are written during deployment
- Start, stop, and trace message flows that are running in integration nodes

The IBM Integration Toolkit includes access to tutorials and a patterns repository to help you get started with your application development.

The IBM Integration Toolkit communicates with one or more integration nodes.



## IBM Integration web user interface



© Copyright IBM Corporation 2015

Figure 1-9. IBM Integration web user interface

WM666 / ZM6661.0

### Notes:

With the IBM Integration Bus web user interface, you can view and manage Integration Bus resources without any extra management software. It provides a view of all deployed integration solutions for an integration node, and gives you access to important operational features such as the built-in data record and replay tool.

IBM Integration Bus includes a set of performance monitoring tools that visually portray current server throughput rates. The performance monitoring tools show various metrics such as elapsed and CPU time in ways that immediately draw attention to performance bottlenecks and spikes in demand. You can get more detail, such as rates for individual connectors. You can use tools to correlate performance information with configuration changes so that you can quickly determine the performance impact of specific configuration changes.



## IBM Integration Bus command utilities

- IBM Integration Toolkit and run time commands
- Available on operating systems that IBM Integration Bus supports
  - z/OS requires a product such as SDSF to allow mixed case on the command line
  - On Windows, components are services and can be started automatically
- Some commands require extra security configuration

© Copyright IBM Corporation 2015

Figure 1-10. IBM Integration Bus command utilities

WM666 / ZM6661.0

### Notes:

In addition to using the IBM Integration web user interface to manage the integration node environment, you can administer IBM Integration Bus by using a command interface.

You can also use the command interface to automate specific tasks. For example, you can write scripts to deploy applications to production integration servers on a schedule.

Some commands access the local components directly, others, need access to IBM Integration Toolkit resources and can run only on Windows or Linux.

Some commands require extra authorization. For example, on some operating systems, administrators must be a member of the `mqbrkrs` group to run administrative commands. For more information about security requirements, see the IBM Knowledge Center for IBM Integration Bus V10.



## IBM Integration API

- Java administration API for IBM Integration Bus
- Applications can use the API to control integration nodes and their resources through a remote interface
  - Deploy files
  - Change the integration node configuration properties
  - Create, modify, and delete integration servers
  - Inquire and set the status of the integration node and its associated resources
  - Get information about the status of integration servers, deployed message flows, and deployed files that are used by the message flows
  - View the integration node Administration log
  - View the integration node Activity log
  - Create and modify message flow applications

© Copyright IBM Corporation 2015

Figure 1-11. IBM Integration API

WM666 / ZM6661.0

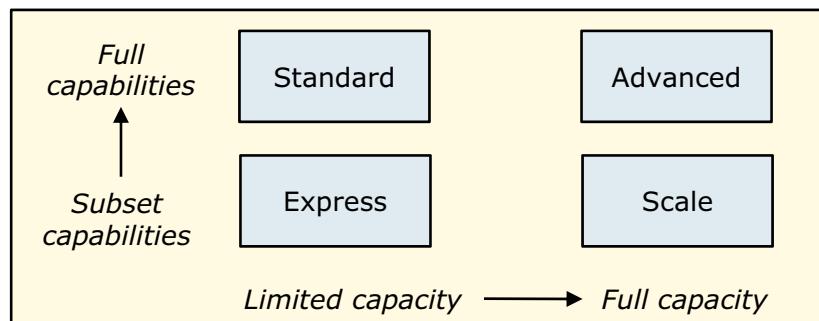
### Notes:

The IBM Integration API is a remote programming interface that your custom integration applications can use to control integration nodes and their resources.

The IBM Integration API is a set of lightweight Java classes that sit logically between the user application and the integration node. It is an alternative interface for administering integration nodes. All IBM Integration Bus applications, such as the IBM Integration Toolkit, use the API to communicate with the integration node environment.

IBM Integration Bus includes samples to learn the basic features that the IBM Integration API provides.

## IBM Integration Bus operation modes



Mode	Features	Integration servers	Deployed message flows
<b>Express</b>	Limited set of message flow nodes	One	Unlimited
<b>Scale</b>	Limited set of message flow nodes	Unlimited	Unlimited
<b>Standard</b>	All features enabled	One	Unlimited
<b>Advanced</b>	All features enabled	Unlimited	Unlimited

© Copyright IBM Corporation 2015

Figure 1-12. IBM Integration Bus operation modes

WM666 / ZM6661.0

### Notes:

The capability and capacity that is provided by IBM Integration Bus varies according to the operation modes in which your integration nodes are running. Your entitlement to run in a particular mode depends on the edition of the product that you purchased.

Operation modes typically restrict the number of message flow nodes that are available, and integration server capacity, but many features are available across all modes of operation. This figure summarizes four of the IBM Integration Bus operation modes: Express, Scale, Standard, and Advanced. IBM Integration Bus is also available in Developer mode and Adapter mode.

- With Developer mode, all features are enabled, but you can use the product for evaluation, development, and test purposes only. Developer mode is available on Windows x86-64 and Linux x86-64 operating systems and is limited to one message per second at the message flow level.
- With Adapter mode, only adapter-related features are enabled, and the types of message flow node that you can use, and the number of integration servers that you can create, are limited.

## Supported hardware and environments

- Operating systems and hardware
  - AIX, Windows, z/OS, HP-UX, Linux on System x, pSeries, zSeries, Solaris (x86-64 and SPARC), Ubuntu
  - Optimized 64-bit support on all platforms
- Virtual images for efficient utilization and simple provisioning
  - Extensive support for virtualized environments such as VMWare and AIX Hypervisor
  - Pre-built images (Hypervisor editions) available on Linux on System x and AIX
  - Support for public and private clouds such as SoftLayer and Pure
  - Chef scripts for automated building of flexible IBM Integration Bus images on GitHub

© Copyright IBM Corporation 2015

Figure 1-13. Supported hardware and environments

WM666 / ZM6661.0

### Notes:

This figure lists the supported hardware and software environments for IBM Integration Bus.

For a complete list of supported operating systems, hardware, virtual images, databases, and ERP systems, see the IBM Integration Bus prerequisites on the IBM website.



## Supported software

- Supports access to industry standard databases
  - DB2, Oracle, Sybase, SQL Server, Informix, solid DB
  - Open Driver Manager support enables new ODBC databases to be accessed
  - JDBC Type 4 for popular databases
- Supports access to message-oriented middleware
  - IBM MQ 7.0.1, 7.1, 7.5, and 8.0
  - JMS 1.2 and 2.0
- Includes access to ERP systems such as SAP, Siebel, PeopleSoft, and JD Edwards

© Copyright IBM Corporation 2015

Figure 1-14. Supported software

WM666 / ZM6661.0

### Notes:

IBM Integration Bus also supports many industry standard databases, message-oriented middleware, and ERP systems.

## Technology components and prerequisites

- IBM MQ
  - Optional on distributed systems for most applications
  - Required on z/OS and for the use of some IBM Integration Bus features
- Java 7.1 on all platforms
- Other prerequisites are determined by operating system and hardware
  - Detailed system requirements are on [www.ibm.com/integration-bus](http://www.ibm.com/integration-bus)

© Copyright IBM Corporation 2015

Figure 1-15. Technology components and prerequisites

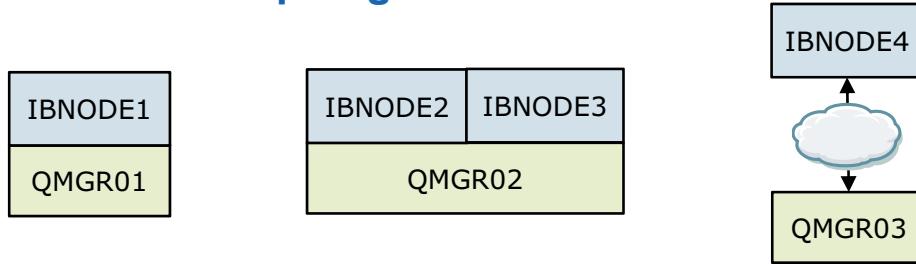
WM666 / ZM6661.0

### Notes:

IBM Integration Bus relies on IBM MQ for some operations on distributed operating systems such as Windows and Linux. IBM MQ is required on z/OS for IBM Integration Bus.

Hardware and software determine more requirements. For more information about prerequisites, see the IBM Integration Bus product page at [www.ibm.com/integration-bus](http://www.ibm.com/integration-bus).

## Flexible IBM MQ topologies



- On distributed systems, IBM MQ is not a prerequisite in most implementations
- On z/OS, IBM MQ is required for installation
- Flexible topology options for IBM MQ access for simplicity, scalability, availability, and migration
- If IBM MQ is installed, IBM Integration Bus detects IBM MQ and configures it
- IBM MQ policies identify run time resources

**\*Note:** On z/OS, only local connections to queue managers are supported

© Copyright IBM Corporation 2015

Figure 1-16. Flexible IBM MQ topologies

WM666 / ZM6661.0

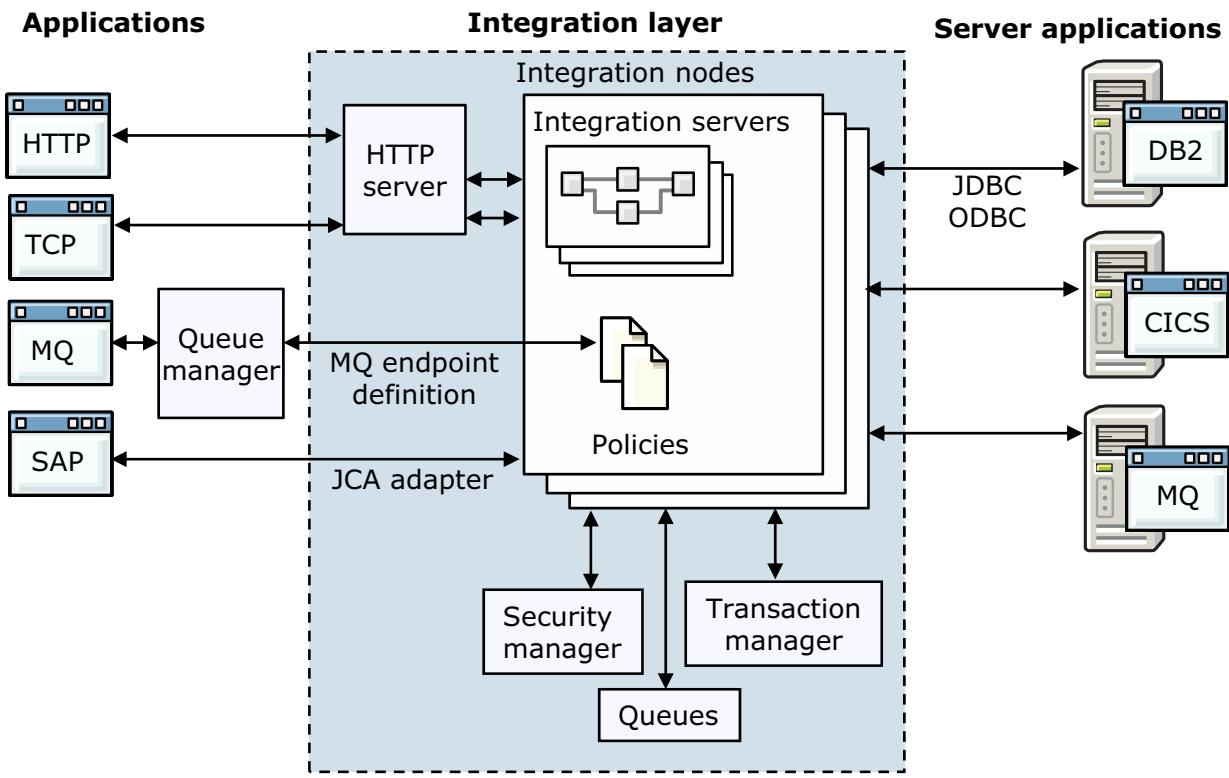
### Notes:

With IBM Integration Bus, you can configure local or client connections to IBM MQ, enabling your integration nodes to get messages from, or put messages to, queues on a local or remote queue manager.

You can configure either a local or client connection between your integration node and your queue manager, depending on the configuration of your existing architecture. If your IBM MQ queue manager is running on the same computer as your integration node, you can specify a local connection to the queue manager. Alternatively, if the IBM MQ queue manager that you want to connect to is hosted on a separate computer from IBM Integration Bus, you can configure a client connection from your integration node. The integration node with the client connection can access the messages on the remote queue manager.

You can define IBM MQ endpoint policies to create reusable run time resource and connection definitions.

## Connectivity to external systems



© Copyright IBM Corporation 2015

Figure 1-17. Connectivity to external systems

WM666 / ZM6661.0

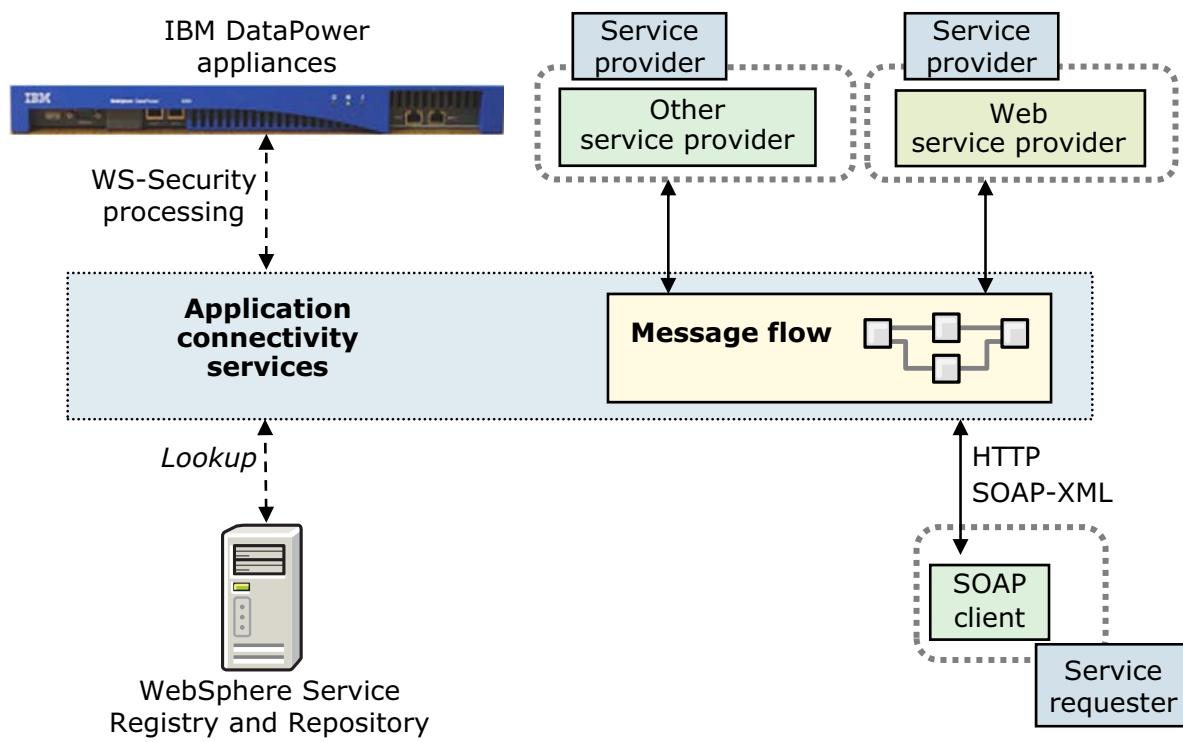
### Notes:

IBM Integration Bus integrates many different products. The functions and features that you add by using supplementary products does not have an impact on your applications, but grants more options in the integration layer, which represents the operations of IBM Integration Bus.

There are many different ways to connect into an IBM Integration Bus flow. Some of these connection methods include IBM MQ, JMS, HTTP, and web services.

IBM Integration Bus has message flow nodes that are designed to allow interoperability with other IBM products such as IBM MQ File Transfer Edition.

## IBM Integration Bus web services



© Copyright IBM Corporation 2015

Figure 1-18. IBM Integration Bus web services

WM666 / ZM6661.0

### Notes:

IBM Integration Bus can be used for connectivity to web services and web service appliances such as IBM DataPower.

As shown in the figure, SOAP messages contain the data that is passed between the web service provider and the service requester. SOAP messages are expressed in XML and can be delivered over HTTP or JMS transport. IBM Integration Bus supports both protocols. So, in a message flow, web services can be requested and aggregated. A message flow can be deployed as a web service, listening to incoming SOAP requests on HTTP or JMS.

IBM Integration Bus can use the IBM DataPower appliance to handle its web services security (WS-Security) processing by providing HTTP/HTTPS encryption and decryption. DataPower SOA appliances from IBM are purpose-built, easy-to-deploy network appliances that help simplify, secure, and accelerate your XML and web service deployments while extending your SOA infrastructure. They can do XML schema validation, XSL transformations (XSLT), web service security, transformations between disparate message formats, and more.

Web services can be looked up dynamically in IBM WebSphere Service Registry and Repository.



## IBM Integration Bus and publish/subscribe support

- IBM Integration Bus uses publish/subscribe to notify applications of significant events that occur in integration nodes
- IBM Integration Bus supports IBM MQ and MQTT for publish/subscribe
  - Choose the publish/subscribe broker based on processing requirements and your existing architecture
- A built-in MQTT broker is provided with IBM Integration Bus
  - MQTT publication is enabled by default for all events that are generated by the integration node except for business events
- If IBM MQ is installed, you can use the IBM MQ publish/subscribe broker
  - IBM Integration Bus connectivity can be published as an IBM MQ publication
  - IBM MQ queue manager delivers the publication to all subscribing applications that match the topic, and other options that are specified on their subscriptions

© Copyright IBM Corporation 2015

Figure 1-19. IBM Integration Bus and publish/subscribe support

WM666 / ZM6661.0

### Notes:

IBM Integration Bus supports two types of publish/subscribe broker: MQTT and IBM MQ. You can choose which type to use based on your processing requirements and your existing architecture.

A built-in MQTT broker is provided with IBM Integration Bus, and MQTT publication is enabled by default for all events that the integration node generates, except for business events. You can specify an alternative MQTT broker if you choose not to use the built-in MQTT broker.

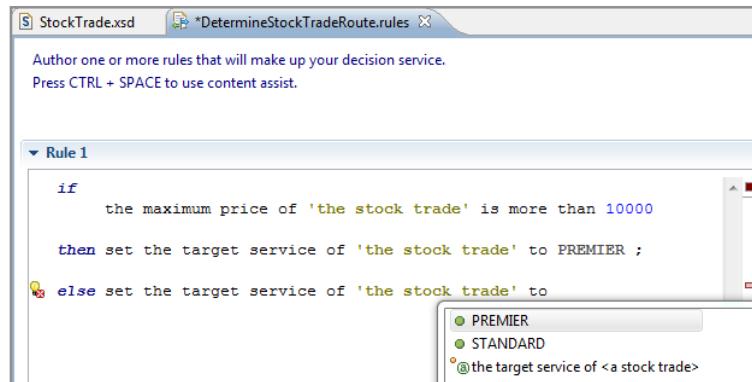
If you installed IBM MQ, you can use the IBM MQ publish/subscribe broker that is provided with IBM MQ.

There are two situations where you can use publish/subscribe in an IBM Integration Bus message flow:

- 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

## Decision management in IBM Integration Bus

- Provides business insight during integration data flows
- DecisionService node
  - Identifies inputs to business rules from in-flight data
  - Starts built-in rule engine to run business logic
  - Captures rules output for downstream processing
- User can create rules in IBM Integration Toolkit or optionally refer to business rules on external IBM Operational Decision Manager decision server
- Embedded rules engine acts on business rules in the flow
  - Rule runs in the same operating system process as integration data flow
  - Rule update notification ensures consistent rule execution



© Copyright IBM Corporation 2015

Figure 1-20. Decision management in IBM Integration Bus

WM666 / ZM6661.0

### Notes:

The DecisionService node in IBM Integration Bus implements business rules to provide operations like routing, validation, and transformation.

 **Note**

The IBM Integration Bus license entitles you to use this component only through the DecisionService node and only for development and functional test. 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.



## IBM Integration Bus runtime security

- Identifies who is authorized to submit a message to a message flow
- IBM Integration Bus Runtime Security Manager controls
  - Allows end-to-end processing on behalf of the identity in the message
  - Specifies identity authentication, mapping, authorization, and propagation
  - Administrator configures by using security profiles
- Uses centralized security provider
  - LDAP for authentication and authorization
  - IBM Tivoli Federated Identity Manager for authentication, authorization, and mapping
- Choose between file-based authorization or IBM MQ queue-based authorization
- Can be offloaded to an IBM DataPower appliance
- Specifies resources that are accessible to that message flow

© Copyright IBM Corporation 2015

Figure 1-21. IBM Integration Bus runtime security

WM666 / ZM6661.0

### Notes:

Security is an important consideration for both developers of IBM Integration Bus applications, and for system administrators that are configuring IBM Integration Bus authorities.

When you are designing an IBM Integration Bus application, it is important to consider the security measures that are needed to protect the information in the system. When a message arrives at an input node, a security profile is used to indicate whether runtime security is configured.

The security manager of the integration node is called to read the security profile. The security profile specifies the combination of authentication, authorization, and mapping that is done with the identity of the message, and by what external security provider.

At subsequent processing nodes in the message flow, it might be necessary for an identity to be used to access a resource such as a database. The identity that is used to access such a resource continues to be a proxy identity, either the identity of the integration nodes or an identity that is configured by using a command. The resource is accessed by using the appropriate proxy identity.

Two external security providers are supported so that the integration node can participate in a centralized security framework: LDAP for authentication and authorization and Tivoli Federated Identity Manager for authentication, mapping, and authorization.



## Operational management and performance

- Record and replay
  - 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
- Global caching
  - Store data that you want to reuse by using the embedded global cache or an external WebSphere eXtreme Scale grid
- High availability
  - Multi-instance integration nodes with IBM MQ or an existing high-availability manager

© Copyright IBM Corporation 2015

Figure 1-22. Operational management and performance

WM666 / ZM6661.0

### Notes:

IBM Integration Bus also includes many features for operational management, such as record and replay, and performance such as global caching.

For audit or problem determination purposes, you can record data to a database, view it, and replay it. The messages are stored in a database, which can be Oracle, Microsoft SQL Server, or DB2.

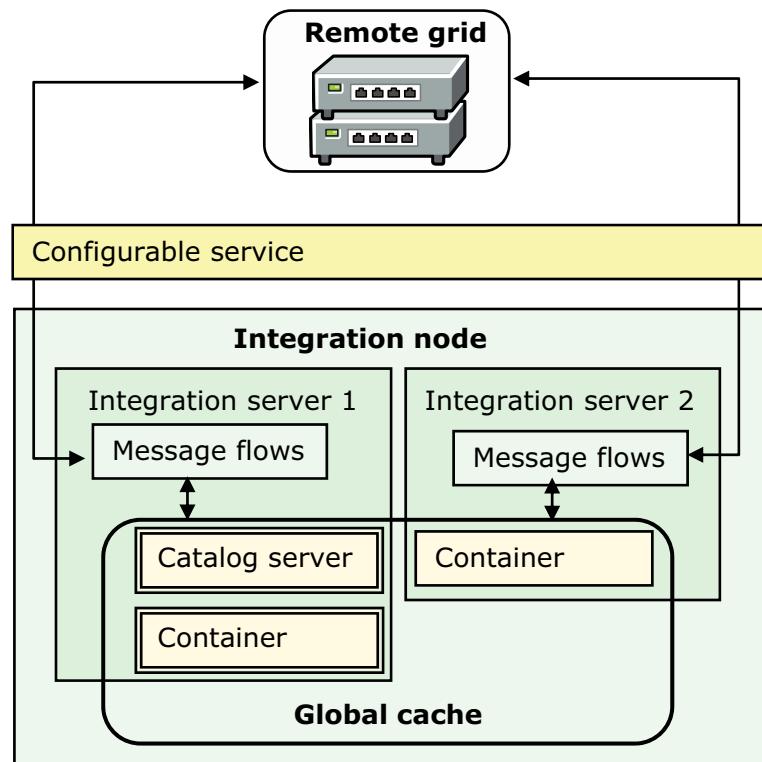
After you record the data, you can view it by using the web interface. You can replay a message to an IBM MQ queue by using the web interface or IBM Integration API. You can also replay the message to another message flow for testing or problem determination.

IBM Integration Bus contains an embedded global cache to store data that you want to reuse. As an option you can WebSphere eXtreme Scale for the cache.

With IBM Integration Bus, you can use a multi-instance integration node so that the integration node is available in the same location as an IBM MQ multi-instance queue manager in a high availability IBM MQ configuration. The multi-instance integration node can either dynamically start in all locations where the multi-instance queue manager runs, or can be set as an IBM MQ Service dependency.

## Connectivity to WebSphere eXtreme Scale grids

- Can connect to external WebSphere eXtreme Scale grid
- 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 2015

Figure 1-23. Connectivity to WebSphere eXtreme Scale grids

WM666 / ZM6661.0

### Notes:

In addition to the grid that is available as the embedded global cache within IBM 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 screenshot shows the IBM Integration Bus in the Cloud web interface. The top navigation bar includes the WebSphere Education logo and the IBM logo. The main title is "IBM Integration Bus in the Cloud". The dashboard displays "My Cloud" information for "MyGoldNode" (Octo-core, 32GB RAM, Monthly charge: \$1000.00) and "MyTrialNode" (43/90 days remaining, Monthly charge: \$0.00), totaling a monthly charge of \$1000.00. A modal window titled "Ready to upgrade or purchase more Integration Nodes?" with a "Click here." button is open. Another modal window titled "My Users" lists "me (Ike)" as a "User" and "administrator" as an "Integration Node". A third modal window titled "Create integrations to deploy into IBM Integration Bus in the Cloud." offers to "Download the IBM Integration Toolkit" and "Meet our developer community". Below these, a section titled "My Integration Nodes" shows "MyGoldNode" with "hostname:2222" and a "Launch web" button, accompanied by a callout bubble: "Click here to start managing your new Integration Node and deploying integrations through the web!". A summary section titled "Compare my Integration Nodes" lists "MyGoldNode" and "MyTrialNode". A yellow callout box highlights the following points:

- IBM Integration Bus offering in an IBM administered cloud environment
  - Helps eliminate typical inhibitors to start Integration Bus projects
  - Allows users to focus on developing solutions
  - Within the constraints of a cloud environment, developers can use the same development tools for both cloud and on-premise software and assets that are generated can be deployed to either environment

© Copyright IBM Corporation 2015

Figure 1-24. IBM Integration Bus in the Cloud

WM666 / ZM6661.0

## Notes:

IBM Integration Bus in the Cloud helps eliminate typical inhibitors to start Integration Bus projects, such as capital expenditures, hardware availability, and the skills for managing an Integration Bus environment. It also allows users to focus on developing solutions rather than installing, configuring, and managing software.

Within the constraints of a cloud environment, developers can use the same development tools for both cloud and on-premises software, and the assets that are generated can be deployed to either environment.



## Using IBM Integration Bus to provide a REST API

- Provides a way to receive JSON over HTTP/HTTPS and expose a REST API
  - Create a REST API in the IBM Integration Toolkit
  - Administer REST APIs as an IBM Integration Bus construct in the IBM Integration web user interface
- IBM Integration Toolkit REST API project
  - Import Swagger V2.0 JSON file to create the REST API project in the IBM Integration Toolkit
  - Defines a metadata format that is based on JSON schema to describe the REST APIs, their parameters, and the messages that are exchanged
  - Original JSON files are included in the project
  - Can use Swagger user interface to test and generate client code bindings

The screenshot shows a software interface titled "Create a new REST API". It displays a table of operations and their corresponding methods and paths. The operations listed are: getPetById, deletePet, updatePetWithForm, partialUpdate, addPet, updatePet, findPetsByStatus, findPetsByTags, uploadFile, createUser, createUsersWithArrayInput, createUsersWithListInput, updateUser, and deleteUser. The methods are color-coded: GET (green), DELETE (red), POST (blue), PATCH (orange), PUT (yellow), and others in grey. The paths are also color-coded. At the bottom of the dialog are buttons for "?", "< Back", "Next >", "Cancel", and "Finish".

Operation	Method	Path
getPetById	GET	/pet/{petid}
deletePet	DELETE	/pet/{petid}
updatePetWithForm	POST	/pet/{petid}
partialUpdate	PATCH	/pet/{petid}
addPet	POST	/pet
updatePet	PUT	/pet
findPetsByStatus	GET	/pet/findByStatus
findPetsByTags	GET	/pet/findByTags
uploadFile	POST	/pet/uploadImage
createUser	POST	/user
createUsersWithArrayInput	POST	/user/createWithArray
createUsersWithListInput	POST	/user/createWithList
updateUser	PUT	/user/{username}
deleteUser	DELETE	/user/{username}

© Copyright IBM Corporation 2015

Figure 1-25. Using IBM Integration Bus to provide a REST API

WM666 / ZM6661.0

### Notes:

Representational State Transfer (REST) is a software architecture style that consists of guidelines and practices for creating scalable web services.

Swagger is a simple yet powerful representation of a RESTful API. Swagger provides a framework for describing, producing, using, and visualizing RESTful APIs.

With IBM Integration Bus V10, you can receive JSON over HTTP/HTTPS and expose a REST API. You can also use the IBM Integration Toolkit graphical data mapper capabilities for the transformation of unmodeled data structures, such as JSON messages.



## IBM Integration Bus Industry Packs

- **IBM Integration Bus Healthcare Pack** provides prebuilt patterns and connections that enable rapid clinical application and device integration for more connected healthcare systems
- **IBM Integration Bus Manufacturing Pack** helps to integrate heterogeneous IT and operational manufacturing systems and make information flow more quickly and reliably
- **IBM Integration Bus Retail Pack** accelerates the development and deployment of integration between retail applications and systems, and enables the transformation and enrichment of data

© Copyright IBM Corporation 2015

Figure 1-26. IBM Integration Bus Industry Packs

WM666 / ZM6661.0

### Notes:

The IBM Integration Bus Industry Packs provide targeted solutions and help to get you started quickly. The Industry Packs are described in more detail in course WM676, IBM Integration Bus V10 Application Development II.



## Conversion from WebSphere Enterprise Service Bus

- Built-in conversion tools for WebSphere Enterprise Service Bus source
  - Preserves structural wiring between primitives of a mediation flow
  - No minimum version requirement of WebSphere Enterprise Service Bus source
  - Remaining manual tasks are provided in a task list
- WebSphere Enterprise Service Bus customers can obtain a transfer license to migrate to IBM Integration Bus, and to create and run integration servers in Scale mode

The screenshot shows the 'Configure global conversion options' step of a conversion wizard. The top bar indicates steps 1, 2, and 3 completed. The main area has a header: 'Configure global conversion options. Add extensions for those resources for which you want to use your own conversion code.' Below are two expandable sections: 'Conversion Result' (checkbox for merging results) and 'Mediation Primitive Converters' (table mapping primitives to converters). The table is as follows:

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
<b>MessageLogger</b>	<b>Subflow placeholder</b>	<b>StockQuote_MediationFlow.component</b>	<b>Placeholder converter</b>
XSLTransformation	Map	StockQuote_MediationFlow.component	Built-in converter

© Copyright IBM Corporation 2015

Figure 1-27. Conversion from WebSphere Enterprise Service Bus

WM666 / ZM6661.0

### Notes:

IBM Integration Bus provides a universal integration capability that addresses a wide range of integration scenarios. These scenarios include web services such as SOAP and REST, messaging, database, file, ERP systems, mobile, physical devices, email, custom systems and more.

IBM Integration Bus supplies a WebSphere Enterprise Service Bus conversion tool that accelerates the conversion of development artifacts that are created for WebSphere Enterprise Service Bus in WebSphere Integration Developer or IBM Integration Designer to IBM Integration Bus development artifacts.



## Supported migration paths

- You can migrate to IBM Integration Bus Version 10.0 from:
  - WebSphere Message Broker Version 7.0.0.5 or later
  - WebSphere Message Broker Version 8.0
  - IBM Integration Bus Version 9.0
- You can migrate only to a full edition of IBM Integration Bus Version 10.0, Remote Adapter Deployment, Express Edition, or Standard Edition

**Note:** IBM Integration Bus Version 10.0 does not include IBM Integration Explorer. There is no migration process for IBM Integration Explorer or WebSphere Message Broker Explorer

© Copyright IBM Corporation 2015

Figure 1-28. Supported migration paths

WM666 / ZM6661.0

### Notes:

IBM Integration Bus makes it simple to move applications from WebSphere Message Broker. All development assets such as message flows, ESQL and Java code, message models, and maps import directly into IBM Integration Bus.

You also have many options for moving your brokers and execution groups to integration nodes and integrations servers, including flexible co-existence options.

## Unit summary

Having completed this unit, you should be able to:

- Describe the features and functions of IBM Integration Bus
- Describe the business value of IBM Integration Bus
- Describe the IBM Integration Bus architecture and components
- Identify the IBM Integration Bus editions

© Copyright IBM Corporation 2015

Figure 1-29. Unit summary

WM666 / ZM6661.0

### Notes:



## Checkpoint questions

1. Choose all the tasks that you can do with IBM Integration Bus:
  - a. Send a message to one or multiple destinations that depend data content
  - b. Use a database to select or store message information
  - c. Transform messages so that diverse applications can understand and process them
  - d. Create a workflow with long-running or manual processes
  - e. Monitor message flows
  - f. Customize processing by using supplied, customer-programmed, or third-party plug-in nodes
  - g. Publish a message to other applications based on a topic or the content
  - h. Get data directly from various applications, triggered by events

© Copyright IBM Corporation 2015

Figure 1-30. Checkpoint questions

WM666 / ZM6661.0

### Notes:

Write your answers here:

1.

## Checkpoint answers

1. Choose all the tasks that you can do with IBM Integration Bus:
  - a. Send a message to one or multiple destinations that depend on the data content
  - b. Use a database to select or store message information
  - c. Transform messages so that diverse applications can understand and process them
  - d. Create workflow with long-running or manual processes
  - e. Monitor message flows
  - f. Customize processing by using supplied, customer-programmed, or third-party plug-in nodes
  - g. Publish a message to other applications based on topic or content
  - h. Get data directly from various applications, triggered by events

The correct answers are a, b, c, e, f, and g.

Choice "d" is incorrect; use WebSphere Business Process Manager instead.

Choice "h" is incorrect; use WebSphere Adapters instead.

© Copyright IBM Corporation 2015

Figure 1-31. Checkpoint answers

WM666 / ZM6661.0

### Notes:



# Unit 2. Application development fundamentals

## What this unit is about

This unit describes the IBM Integration Bus components and how they work together. You also learn how to import and export resources into the IBM Integration Bus development environment, view the message processing node properties, and test the message flow by using the IBM Integration Bus Flow exerciser.

## What you should be able to do

After completing this unit, you should be able to:

- Describe how IBM Integration Bus does basic message processing
- Describe the components of a message flow application and message processing nodes
- Describe the basic structure of a logical message tree
- Use patterns as a starting point for developing message flow applications
- Import resources to and export resources from the IBM Integration Toolkit
- Use the IBM Integration Toolkit Flow exerciser to test a message flow application
- Use the IBM Integration Toolkit to check the status of the integration node, integration server, and message flow application

## How you will check your progress

- Checkpoint questions
- Hands-on exercise

## References

IBM Knowledge Center

## Unit objectives

After completing this unit, you should be able to:

- Describe how IBM Integration Bus does basic message processing
- Describe the components of a message flow application and message processing nodes
- Describe the basic structure of a logical message tree
- Use patterns as a starting point for developing message flow applications
- Import resources to and export resources from the IBM Integration Toolkit
- Use the IBM Integration Toolkit Flow exerciser to test a message flow application
- Use the IBM Integration Toolkit to check the status of the integration node, integration server, and message flow application

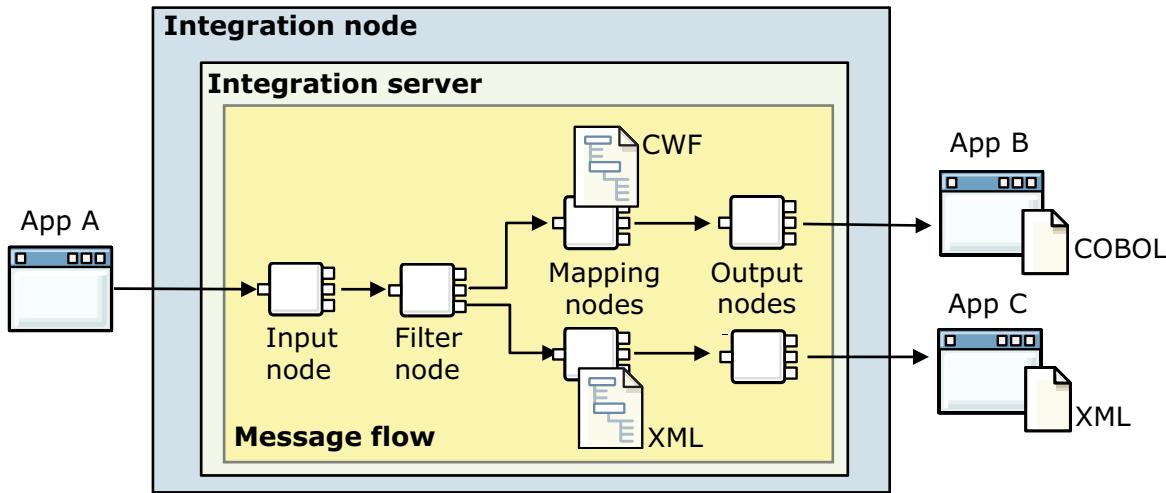
© Copyright IBM Corporation 2015

Figure 2-1. Unit objectives

WM666 / ZM6661.0

### Notes:

## Main components



- **Integration node** routes, transforms, and enriches in-flight messages as determined by message flows and message models
- **Integration server** is a named grouping of message flows
- **Message flow** is a sequence of operations
- **Message flow nodes** define the operations
- **Message models** define structure and properties of the data

© Copyright IBM Corporation 2015

Figure 2-2. Main components

WM666 / ZM6661.0

### Notes:

In IBM Integration Bus, an integration node centralizes routing and transformation functions that message flows define.

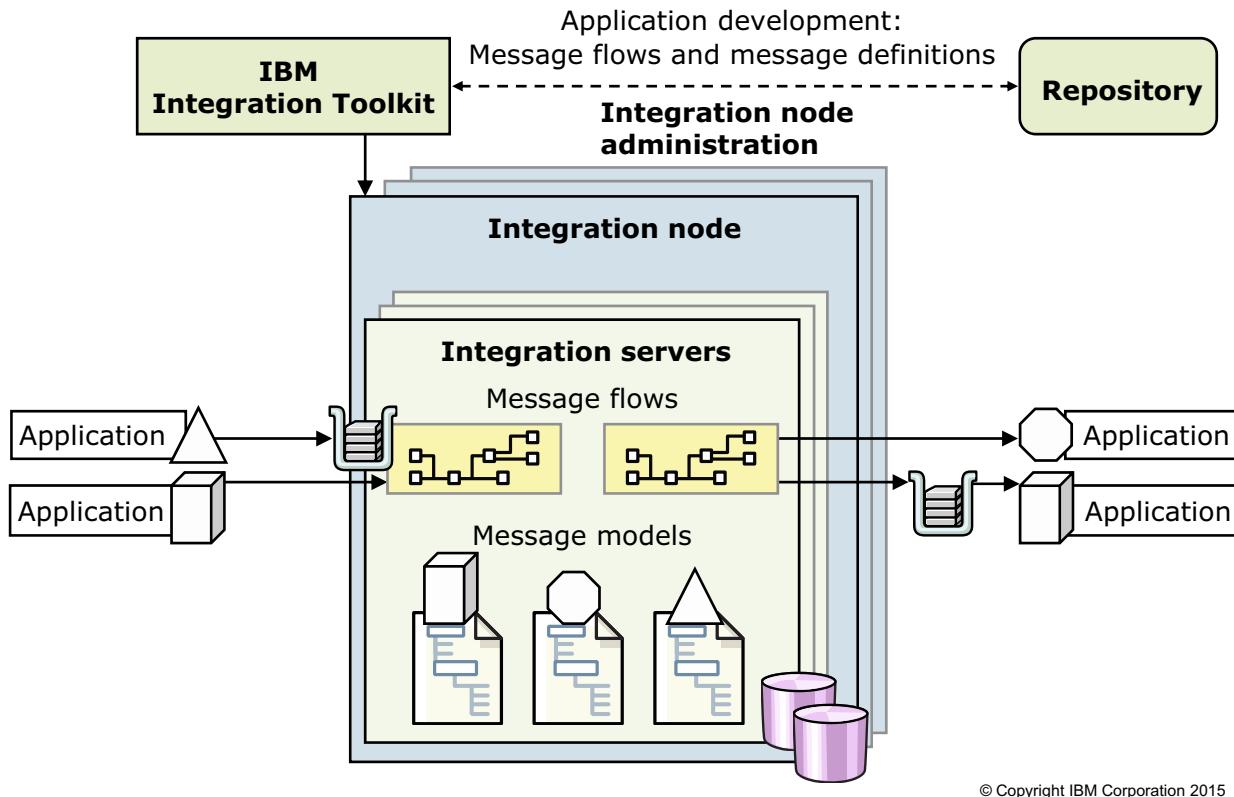
The *message flow* defines the sequence of operations on a message by a series of *message processing nodes*. The actions are defined in terms of the message format, its content, and the results of individual actions along the message flow.

A message flow can route messages from sender to recipient based on the content of the message. However, notice that the direction of the flow is, as it was with basic messaging, from the source application to the destination application. In other words, it is assumed that the destination wants, expects, and handles all messages sent to it.

Messages can also be transformed from one format to another before being delivered, perhaps to accommodate the different requirements of the sender and the recipient.

The message flow can also transform messages by modifying, combining, adding, or removing data fields. Information can be mapped between messages and databases. More complex manipulation of message data can be achieved by writing code, for example in Extended SQL (ESQL) or Java, within configurable nodes.

## Main components interaction



© Copyright IBM Corporation 2015

Figure 2-3. Main components interaction

WM666 / ZM6661.0

### Notes:

The main components of the IBM Integration Bus are the IBM Integration Toolkit, integration nodes, integration servers, message flows, and message models.

The IBM Integration Toolkit is an integrated development environment. Development artifacts can be stored in and managed by an external repository.

The integration node is the runtime engine for the message flows that are created in the IBM Integration Toolkit.

Each integration node contains one or more processes that are called *integration servers*. An integration server can contain one or more message flows.

Some message flows need message models or schemas. The integration node uses the message models and schemas to parse and optionally, validate the message content and construct predefined messages.

## IBM Integration Bus integration node

- Routes, transforms, and enriches in-flight messages as determined by message flows and message models
- Can be many integration nodes, each running on separate systems to:
  - Provide protection against failure
  - Separate the work
- Requires an IBM MQ queue manager for some functions
- Must have administrator access rights to create
- One integration node that is named **TESTNODE\_UserName** is automatically created for development with IBM Integration Toolkit

© Copyright IBM Corporation 2015

Figure 2-4. IBM Integration Bus integration node

WM666 / ZM6661.0

### Notes:

An integration node is a system service on Windows, a daemon process on UNIX, or a started task on z/OS. It controls processes that run message flows.

You can create integration nodes on every operating system that IBM Integration Bus supports.

There can be many integration nodes, and each can be running on a different system. This architecture provides protection against failure, and can separate work across different divisions in a business.

For some operations, the integration node requires access to an IBM MQ queue manager and in some cases, a set of special SYSTEM queues.

On all operating systems, you must have administrator rights to create an integration node.

An integration node that is named **TESTNODE\_UserName** is automatically created the first time that you start the IBM Integration Toolkit. This integration node is provided for development and testing.

## Integration servers

- Named grouping of message flows are assigned to an integration node
- Run in separate address spaces or as unique processes to provide isolation and scalability
- Each integration server is a separate operating system process, which provides isolated runtime environments for a set of deployed message flows
- Within integration servers, configurable thread pools are allocated for deployed message flows
- Every integration server within an integration node must have unique name
- One integration server that is named **default** is automatically created on the **TESTNODE\_UserName** integration node

© Copyright IBM Corporation 2015

Figure 2-5. Integration servers

WM666 / ZM6661.0

### Notes:

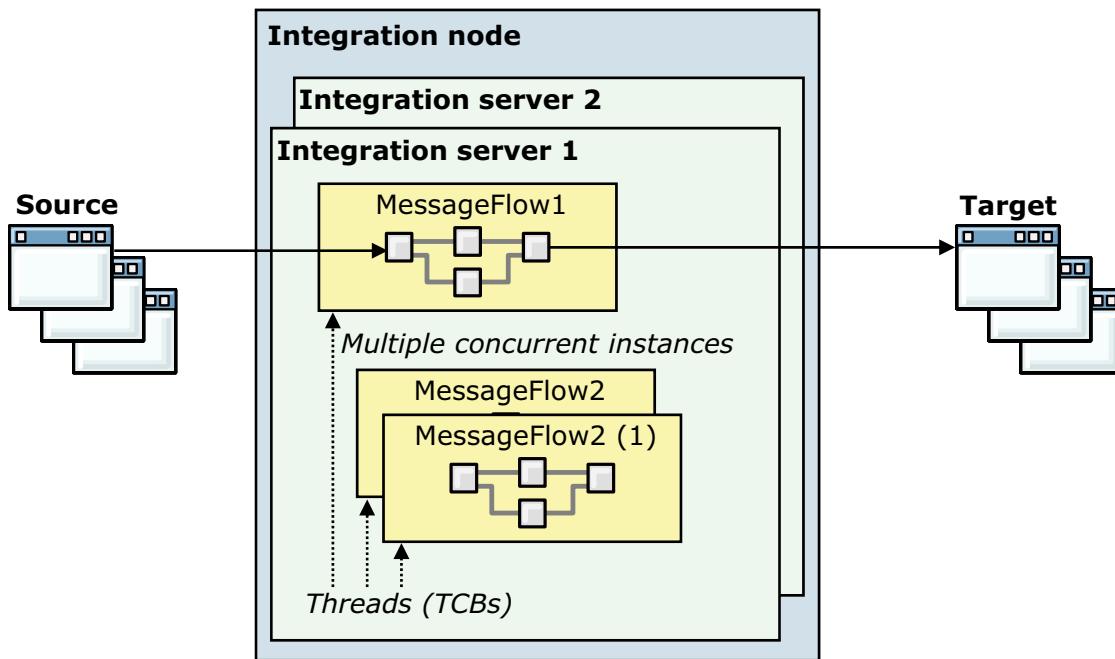
An *integration server* is a named grouping of message flows that are assigned to an integration node. Each integration server is started as a separate operating system process, providing an isolated runtime environment for a set of deployed message flows.

Within an integration server, the assigned message flows run in different thread pools. You can specify the size of the thread pool (that is, the number of threads) that are assigned for each message flow by specifying the number of instances of each message flow.

Each integration server should have a unique name. Some applications truncate the integration server name at 30 characters. If the first 30 characters of two integration servers are the same, they are considered to be the same integration server. Restrict the names of integration servers to 30 characters to avoid duplicate integration server names.

An integration server that is named **default** is automatically created on the **TESTNODE** integration node that first time that you start the Integration Toolkit.

## IBM Integration Bus runtime environment



IBM Integration Bus integration node runs on AIX, Windows, z/OS, HP-UX, Linux, Solaris, and Ubuntu

© Copyright IBM Corporation 2015

Figure 2-6. IBM Integration Bus runtime environment

WM666 / ZM6661.0

### Notes:

At run time, applications send messages to the integration node and integration servers. The integration node routes each message by using the rules that are defined in message flows and message models, and transforms the data into the structure that the receiving application requires.

An integration node can have more than one integration server. Each integration server can run one or more message flows. The limiting factors for integration servers and message flows are system resources such as memory, processor speed, and disk space.

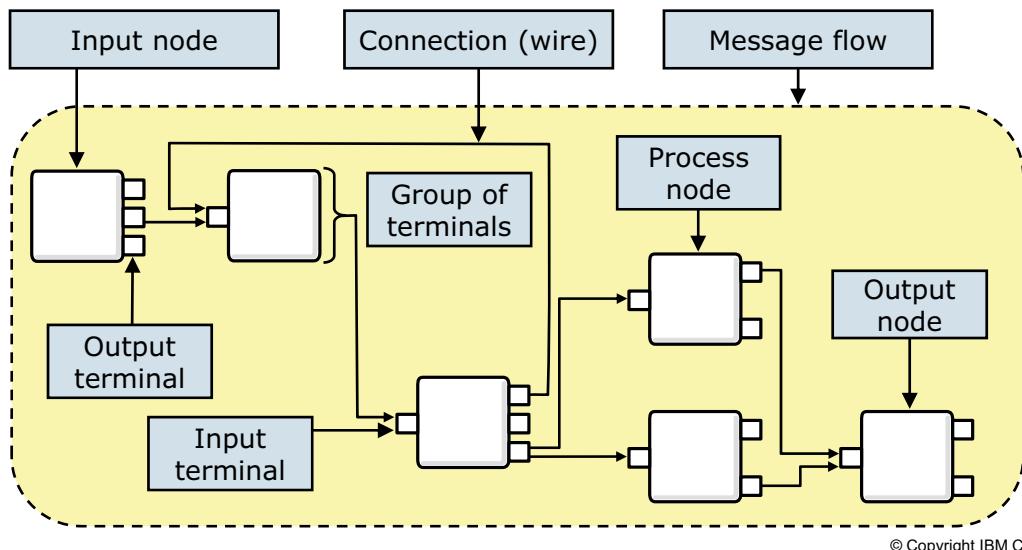
Message flows run as threads (task control blocks, or TCBs, on z/OS) within an integration server. When a message flow is deployed to an integration server, the integration node automatically starts one instance of the message flow. When you deploy a message flow to an integration server, you can specify the number of extra threads that the integration server can use to service the messages for the flow. These additional threads are created only if there are sufficient input messages. You can specify up to 256 threads for each message flow.

The same message flow can be assigned to one or more integration servers. Running the same message flow in multiple integration servers increases throughput, but there is no way to influence the order in which messages are processed.

Each instance waits to retrieve a message from the input node, and runs in parallel with other instances that retrieve a message from other input nodes.

## Message flows

- A message flow is a sequence of operations
  - Composed of message flow nodes that are connected to one another
  - Must include an input node for the source of data to process
  - Typically includes one or more output nodes that deliver data to its destinations



© Copyright IBM Corporation 2015

Figure 2-7. Message flows

WM666 / ZM6661.0

### Notes:

The figure shows the components of a message flow.

A message flow starts with at least one input node, which acts as the source of messages for the message flow to process. Often there is only one input node, but there can be multiple input nodes, each pointing to a different source.

Message flow processing and transport nodes are the intermediate nodes where messages are transformed and enriched.

Output nodes place messages on a transport or another destination, although it is possible to end a flow without an output node.

Input and output terminals are the connectors between nodes in a flow. The terminals of the nodes are “wired” together. Wires provide a path for messages to flow from one node to another. A terminal can have multiple connections, which result in a duplication of the message, or multiple different messages. The order in which any of the output paths are taken is random, but the paths are run sequentially. Each path is followed until it ends, and then the next path is taken.



Note

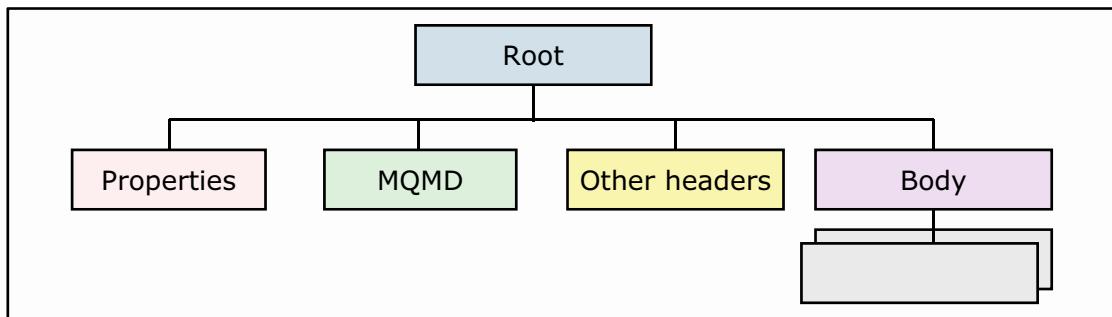
Multiple connections to the same input terminal are **not** message aggregation.

A typical message flow is said to be “stateless,” which means each instantiation of a message flow is treated as an independent transaction, unrelated to any previous request. Later in the course, you see how multiple transactions can be “related” to one another.

By setting the appropriate properties, you can make message flows transactional so that a runtime error backs out any intermediate processing. You can also cause the integration node to process particular parts of a message flow on multiple threads, for parallel processing. You also can define explicit error handling logic, so that runtime errors are handled according to your specific needs, beyond the default error handling that IBM Integration Bus provides.

## Message model and tree

- IBM Integration Bus routes and manipulates data after converting it into a logical tree
  - Input node of the message flow converts incoming data from a stream of bits into a logical tree structure through a *parsing* operation
  - Output node of message flow creates outgoing data (in the bit stream) from the tree structure by a *serialization* operation
- A logical message tree contains the original data, plus other information about the runtime environment, message transport properties, and other “control” information
- Transport type determines the logical tree headers that are included in the tree



Example logical tree for an IBM MQ message

© Copyright IBM Corporation 2015

Figure 2-8. Message model and tree

WM666 / ZM6661.0

### Notes:

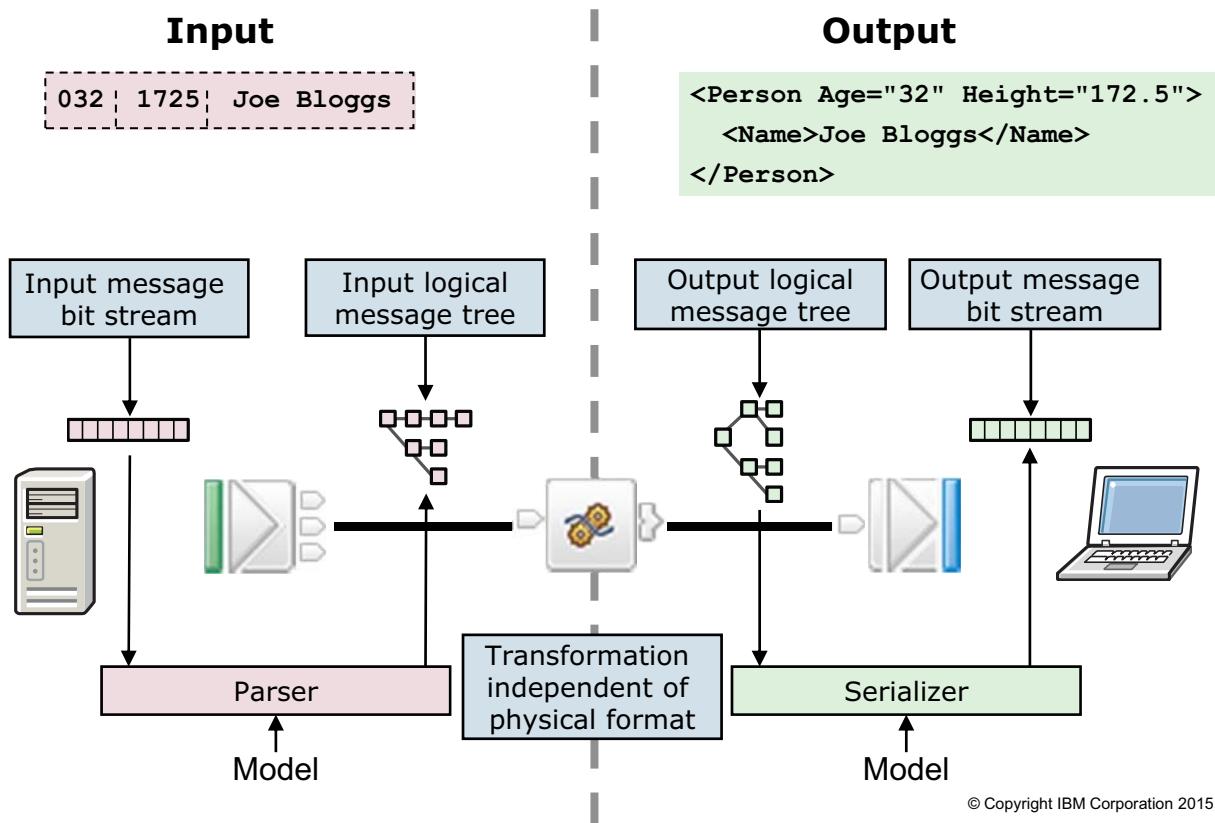
To simplify message transformation and routing, IBM Integration Bus separates the physical format of the data from the logical structure of the data. The logical message tree (or “message assembly”) is the method that IBM Integration Bus uses to describe the structure of the message properties, header, and body.

The tree structure begins with a node called Root. Under Root, other nodes and subtrees are created. These nodes and trees contain the original message body itself, along with information about the message transport that was used to bring in the message. Information about the runtime environment is also part of the tree. If a runtime error occurs, a special tree that is called an Exception List that contains information about the error, is constructed and attached to the Root tree node.

Any part of the logical message tree is accessible by message processing nodes. This capability allows your message flows to make processing decisions that are based on the context of the runtime environment and other factors.

Because the logical message follows a tree structure, it is convenient to use XPath expressions to traverse and modify the message. You can specify XPath expressions in several message processing nodes.

## Message parsing overview



© Copyright IBM Corporation 2015

Figure 2-9. Message parsing overview

WM666 / ZM6661.0

### Notes:

An IBM Integration Bus message flow typically receives messages in a defined format, transforms them, and outputs them in a different format.

All message flow nodes that operate on data require it to be in a logical tree. When the data is brought in from an external source, such as a flat file or message transport (such as IBM MQ), the input node that reads the message converts it to the logical tree structure. This operation is called *parsing*.

Before a message can be sent out from a message flow, it must be converted back to a bit stream. An output node converts the message by reversing the parsing process (sometimes called *serialization*).

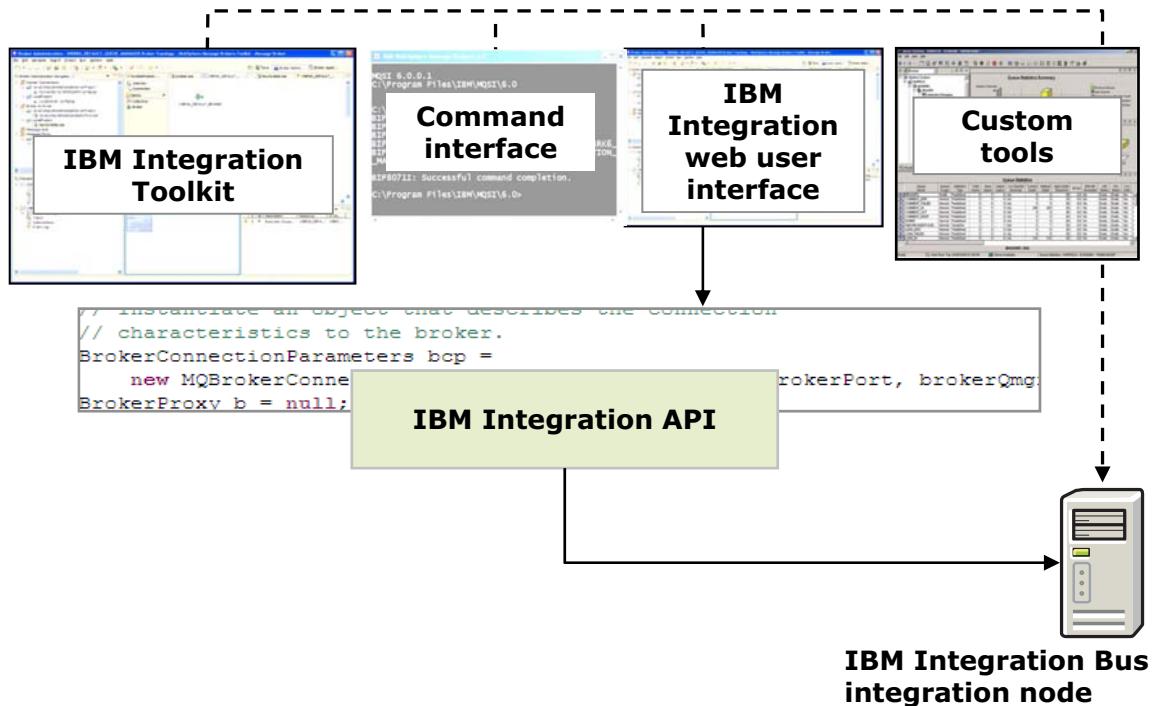
The example in the figure shows a COBOL data structure that is transformed into an XML document. In the example, the COBOL input message contains an age, height, and name field that arrives in the form of an input bit stream. Before the message flow can process it, it is converted into an input logical message tree for transformation by the message flow. A logical message tree is an integration node data structure that reflects the logical structure of the message that the processing nodes use. The structure is independent of the transformation logic.

In the example, logic is applied in a Compute node to transform the message and create an output logical message tree. The data is converted into an output bit stream that represents the message as an XML document.

The component of the integration node responsible for converting a bit stream into a logical message tree is a parser.



## IBM Integration Bus administration tools



© Copyright IBM Corporation 2015

Figure 2-10. IBM Integration Bus administration tools

WM666 / ZM6661.0

### Notes:

This figure shows that the IBM Integration Toolkit, IBM Integration web user interface, commands, and third-party tools communicate with an integration node by using the IBM Integration API.

These applications all use the Integration API to control all aspects of the IBM Integration Bus environment. You can write custom applications to manage the IBM Integration Bus environment with the Integration API.

## Command utilities

- IBM Integration Toolkit and integration node runtime commands
- Some commands can be run only on the computer on which the integration node is running
- User must be a member of the security group **mqbrkrs**, or must be the same user ID that is running the integration node
- Must run `mqsiprofile` script to set up command environment or run **IBM Integration Console**
- Supported on Linux, UNIX, Windows, and z/OS
  - z/OS requires a product such as SDSF to allow mixed case on the command
  - On Windows, components are services and can be started automatically

© Copyright IBM Corporation 2015

Figure 2-11. Command utilities

WM666 / ZM6661.0

### Notes:

You can complete many of the administrative tasks that are supported in the IBM Integration Toolkit and web user interface by using a command interface. You can also use these commands to write scripts to complete such tasks as deploying applications to production integration servers on a schedule, for example.

IBM Integration Bus commands require a special command environment. You must run the `mqsiprofile` script to set up the command environment on Linux and UNIX.

On Windows, you can start the IBM Integration Bus command environment (the IBM Integration Console) from a menu. For example, on Windows, you can start the IBM Integration Console by clicking **Start > Programs > IBM Integration Bus > IBM Integration Console**.

Some commands, such as `mqsistart`, access the local components directly. Other commands, such as `mqsicreatebar`, need access to IBM Integration Toolkit workspace resources and can run only on Windows or Linux with the IBM Integration Toolkit.

In most cases, the user that issues commands must be a member of the **mqbrkrs** group.

## Basic IBM Integration Bus commands

**iib** Complete tasks that are associated with the installation of IBM Integration Bus

Example: `iib verify all`

**mqsilist** List the IBM Integration Bus components on the system

Examples: `mqsilist`

`mqsilist IntNode`

`mqsilist IntNode -e IntServer`

**mqsistart** Start components

Example: `mqsistart Component`

**mqsistop** Stop components; use the optional `-i` flag to force an immediate stop when the component is an integration node

Examples: `mqsistop Component`

`mqsistop IntNode -i`

**Note:** Always check the system log (syslog) after a start or stop.

© Copyright IBM Corporation 2015

Figure 2-12. Basic IBM Integration Bus commands

WM666 / ZM6661.0

### Notes:

Commands can be used to check the existence of IBM Integration Bus components on a system, and to start or stop a component. The figure lists some of the basic commands that can be used to control and monitor the integration node.

You use the `iib` command to complete tasks that are associated with the installation of IBM Integration Bus.

The `mqsilist` command lists the IBM Integration Bus components. It has some optional flags that are not listed in the figure. For example, the `-d` option can be used to select the detail to include in the results:

- **0** returns only the integration node name and the names of the associated queue managers.
- **1** returns a one line summary of each resource (the default).
- **2** returns detailed information about each resource.

You can use the `mqsistart` and `mqsistop` commands to start and stop an integration node or integration server.



## IBM Integration Toolkit

- Eclipse-based desktop development environment for message flow and message model design, deployment, and testing
  - A *perspective* is a collection of views, toolbar icons, and menus, which are grouped to accomplish a specific type of work
  - A *view* supports editors and presents information
  - *Editors* are used to create and modify message flows, message models, and maps
- Tip:** If you accidentally close a view or editor, click **Window > Reset perspective** or **Window > Show view** to restore the view.
- Requires a workspace
  - Location on file system for resources that are used to develop applications
  - Must provide a workspace name each time IBM Integration Toolkit is started
  - Only one user at a time can open a workspace
- Can be used with source code and version control systems

© Copyright IBM Corporation 2015

Figure 2-13. IBM Integration Toolkit

WM666 / ZM6661.0

### Notes:

The IBM Integration Toolkit is an Eclipse-based integrated development environment.

The Eclipse environment has *perspectives*, *views*, and *editors*. A perspective controls initial view visibility, layout, and action visibility.

The IBM Integration Toolkit provides standard perspectives for general resource navigation, online help, and team support tasks. More perspectives are supplied by other plug-ins.

Each perspective has its own views and editors that are arranged for presentation on the screen. Several different types of views and editors can be open at the same time within a perspective.

Views provide information about the currently selected object. For example, if you select a node in a message flow, the properties of the node are shown in another view. Views have a simpler lifecycle than editors. Modifications that are made in a view (such as changing a property value) are generally saved immediately, and the changes are reflected immediately in other related parts of the user interface.

With editors you can open, edit, and save objects much like file-system-based tools. When active, an editor can contribute actions to the menus and toolbar. The IBM Integration Toolkit provides a

message flow editor and message model editor for development. Other editors are supplied by plug-ins or applications.

The applications, folders, and files that exist in the IBM Integration Toolkit are called resources. They are collectively referred to as the *workspace* and they are saved in the local file system.



## Team development

- Each developer works in their own IBM Integration Toolkit
  - Periodically releases changes to team code
  - Can update workbench with team code any time
- Team repository
  - Any repository that Eclipse supports can be used
  - Directly supported repositories: CVS and Rational ClearCase LT
  - Repository options can include change control and history, file locking, and security
  - A single development repository can support multiple integration nodes

© Copyright IBM Corporation 2015

Figure 2-14. Team development

WM666 / ZM6661.0

### Notes:

With IBM Integration Bus, each developer works in their own Integration Toolkit instance. The artifacts that the Integration Toolkit creates are maintained as files so a source code management system or team repository can be used to manage the artifacts.

A source code management system or team repository is not part of the Integration Toolkit, but it directly supports any Eclipse repository. For example, Concurrent Version System (CVS) is a simple open source software configuration management system. It implements version control, parallel development, and lifecycle integration. Individual developers can use CVS, as can large, distributed teams.



Figure 2-15. IBM Integration Toolkit Welcome page

WM666 / ZM6661.0

### Notes:

The Welcome page, which is shown in the figure, is displayed the first time that the Integration Toolkit starts. The Integration Toolkit is started on Windows by clicking **Start > All Programs > IBM Integration Bus > IBM Integration Toolkit**.

The Welcome page is also accessed at any time by clicking **Welcome** from the **Help** menu. The Welcome topics are **Get Started**, **What's New**, and **Language Pack**. The **Get Started** topic includes links for Integration Bus tutorials.

Close the Welcome page to access the Application Development perspective and begin message flow and message model development.



## Default configuration for development

- IBM Integration Toolkit creates a complete local unit test configuration the first time that it starts
  - Integration node that is named `TESTNODE_UserName`
  - Integration server that is named `default`
- Requires administrator privileges and a local user ID on Windows
- Already created on the exercise image for this course

© Copyright IBM Corporation 2015

Figure 2-16. Default configuration for development

WM666 / ZM6661.0

### Notes:

The Integration Toolkit provides a complete local unit test configuration the first time that it starts for message flow testing.

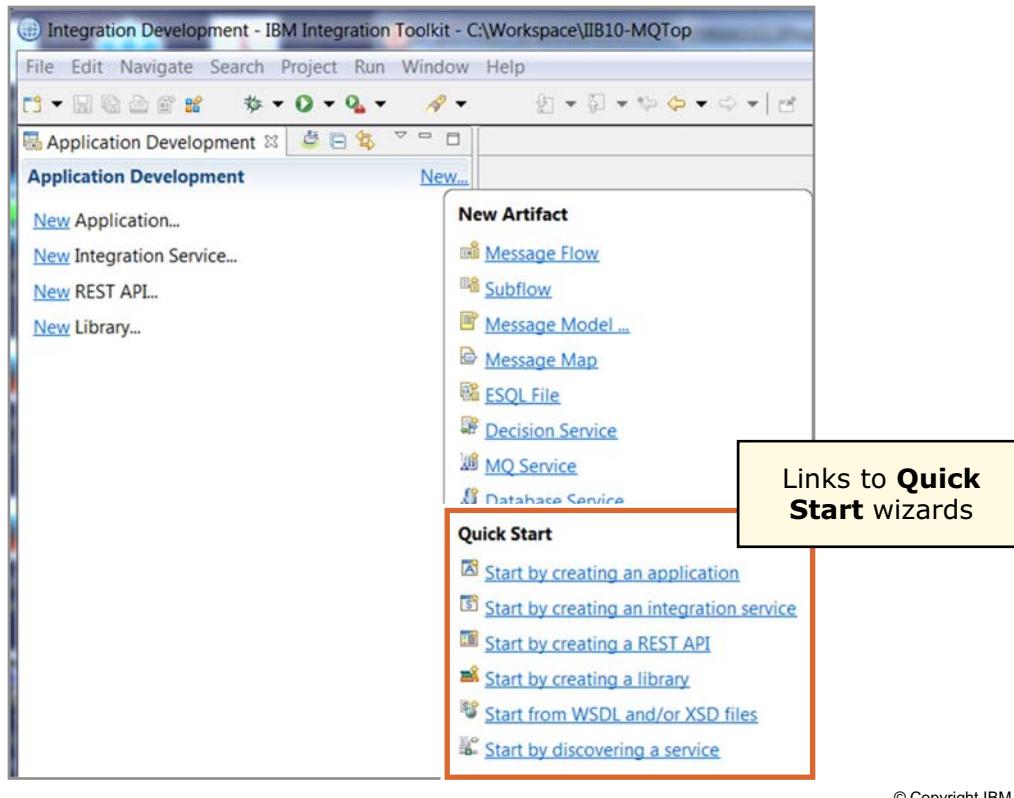
The default configuration includes an integration node and an integration server on port a queue manager listener, and an integration server.

The default configuration uses port 4414 for the HTTP listener for the IBM Integration web user interface. Verify that no other application uses this port.

You must have administrator access rights on the system where the Integration Toolkit runs.



## Integration Development perspective at start



© Copyright IBM Corporation 2015

Figure 2-17. Integration Development perspective at start

WM666 / ZM6661.0

### Notes:

The figure shows the Integration Development perspective that is displayed the first time that it is started. It contains links to Quick Start wizards and predefined message flow patterns.

You can use Quick Start wizards to rapidly build an IBM Integration Bus application with little or no coding. The Quick Start wizards include:

- Start by creating an application
- Start by creating an integration service
- Start by creating a REST API
- Start by creating a library
- Start from a WSDL, XSD file, or both
- Start by discovering a service

The IBM Integration Toolkit accelerates your development by helping you create a message flow. You can:

- Use supplied patterns that encapsulate a tested approach to solving a common architecture, design, or deployment task in a particular context.

- Use existing WSDL files or XML schema definition files as a basis for message set and message flow development.
- Create an application “from scratch” by creating application or library in which to store your development artifacts. The Integration Toolkit helps you define the application or library and the components you need. From there, you can create a message flow and any other components you need.

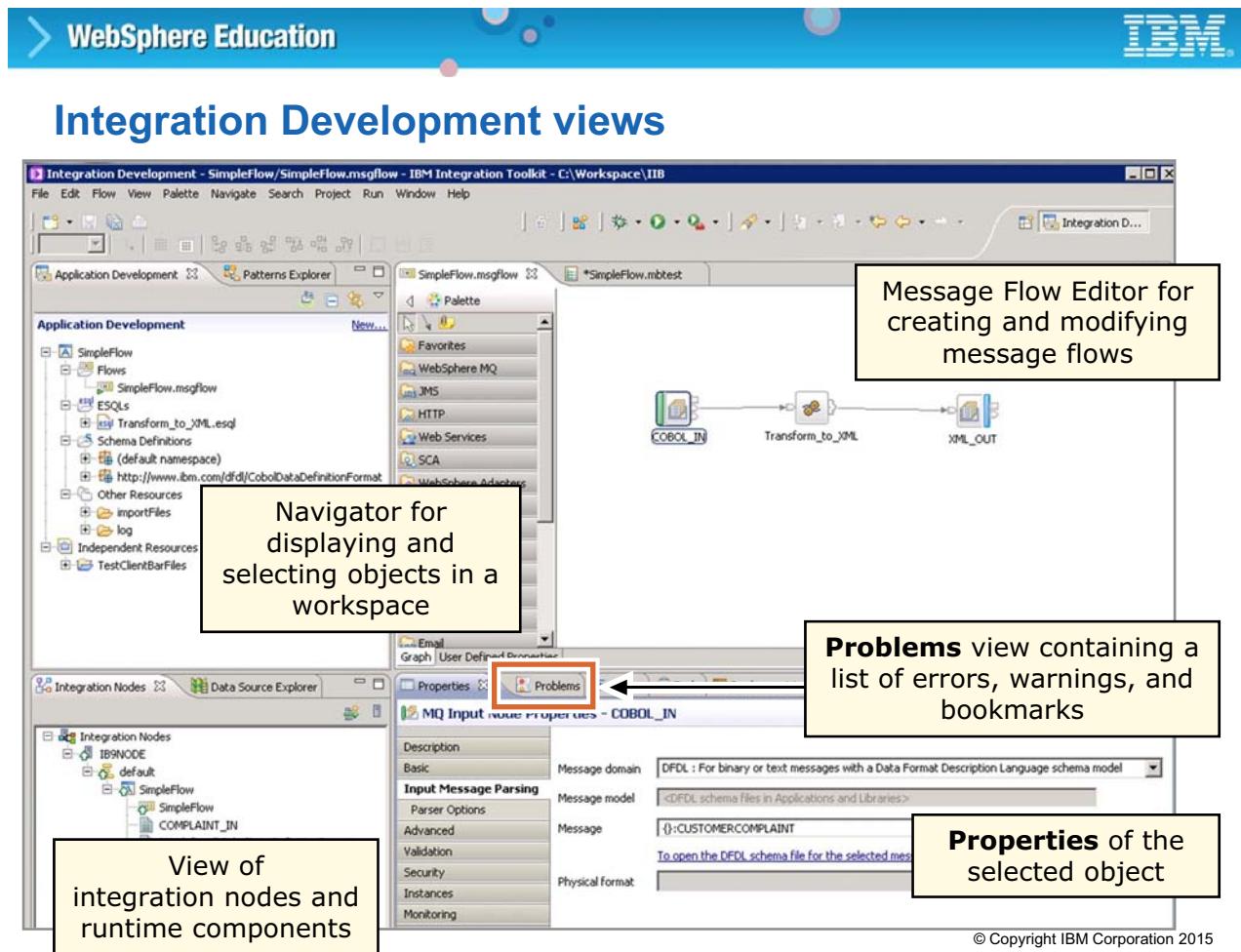


Figure 2-18. Integration Development views

WM666 / ZM6661.0

## Notes:

The figure shows some of the most common views that are available in the Integration Development perspective in the Integration Toolkit:

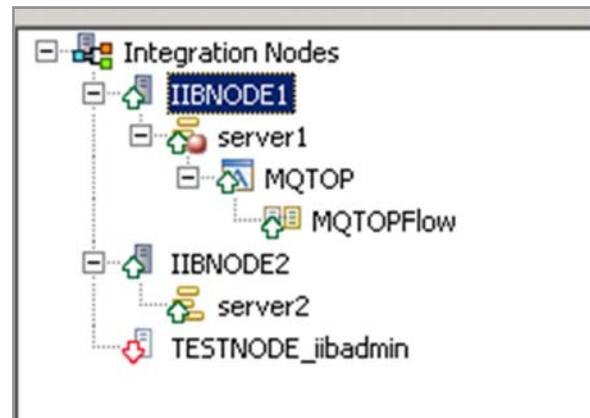
- The Message Flow Editor view contains the message flow nodes in a palette and a canvas for developing message flows and wiring message nodes.
- The Application Development navigator is used to view and select objects in a workspace. Right-clicking most objects in the navigator starts a menu. Double-clicking most objects opens the object in an editor.

Some views are provided in the lower area of the Integration Development perspective.

- The **Integration Nodes** view provides the status of the integration nodes, integration servers, and message flows.
- The **Properties** view provides the configuration properties for the selected object.
- The **Problems** view contains the list of current error or warning messages.

## Integration Nodes view

- Allows application developer to:
  - Create and delete integration nodes
  - Connect to remote integration nodes
  - Create and delete integration servers
  - Deploy message flows to an integration server
  - Remove deployed artifacts from integration server
  - Start and stop integration nodes, integration servers, and message flows
  - View the status of the integration nodes, integration servers, and deployed objects
  - Stop Flow exerciser recording
  - Configure debug port and start the flow debugger for integration server



© Copyright IBM Corporation 2015

Figure 2-19. Integration Nodes view

WM666 / ZM6661.0

### Notes:

Developers can use the **Integration Nodes** view for some basic administration, such as stopping and starting the integration node, deploying message flows, and determining what flows and components are deployed in an integration server.

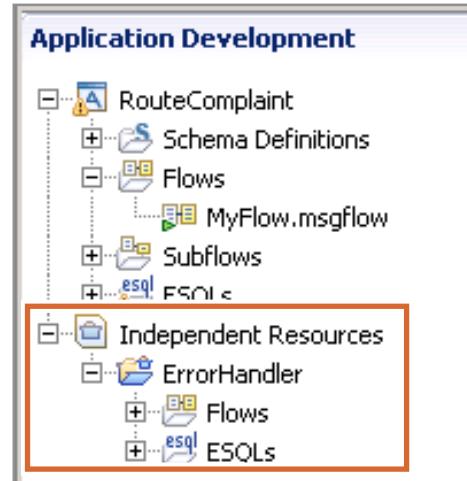
Administration tasks in the **Integration Nodes** view are limited to the tasks listed on the figure. Full access to integration node, integration server, and message flow administration is provided through the IBM Integration web user interface and IBM Integration Bus commands.



## Projects, applications, and libraries (1 of 2)

- **Integration project**

- A specialized container in which you create and maintain all the resources that are associated with one or more message flows
- Contains a single message flow and its resources, or groups message flows and resources, in a single project to organize your message flow resources
- Shown in the **Independent Resources** folder in the **Application Development** view
- Add resources to an integration project and deploy the resources that are contained in that Integration project



© Copyright IBM Corporation 2015

Figure 2-20. Projects, applications, and libraries (1 of 2)

WM666 / ZM6661.0

### Notes:

Within the Integration Toolkit, you can organize all resources in various ways. These resources are the “building blocks” that comprise a message flow application solution, and include message flows, message models or sets, other reusable components, and references to other application solution components. With this approach, you can build reusable components that can become part of multiple applications.

The three primary containers that you use to organize Integration Bus resources are projects, applications, and libraries.

Each of these containers is represented at the operating system level by groups of related files. The Integration Toolkit manages the logical relationships between the files. Altering the files outside the Integration Toolkit can damage the project. You must use the Integration Toolkit to package and move resources. Do not move resources by moving their associated files in the file system.

An *Integration project* is a specialized container in which you create and maintain all the resources that are associated with one or more message flows. An integration project can contain:

- Message flows
- Subflows
- Message maps
- ESQL files
- Database definitions
- BAR files

A project can contain a single message flow and its resources. You can also group related message flows and resources in a single project to provide an organizational structure to your message flow resources.

 WebSphere Education 

## Projects, applications, and libraries (2 of 2)

- **Application**

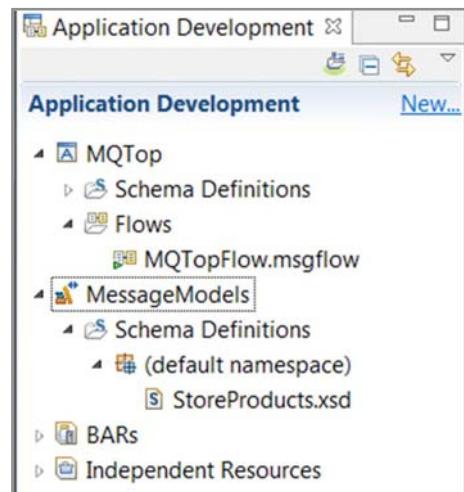
- Container for all the resources that are required to create a complete solution
- Can include message flows, message definitions (DFDL and XSD files), JAR files, XSL stylesheets, and WebSphere Adapters files

- **Library**

- Logical grouping of related reusable resources (message models, maps, and subflows) that an application uses
- Can be shared or static

- **BAR file**

- Unit of deployment to the integration node that contains integration projects, applications, and libraries



© Copyright IBM Corporation 2015

Figure 2-21. Projects, applications, and libraries (2 of 2)

WM666 / ZM6661.0

### Notes:

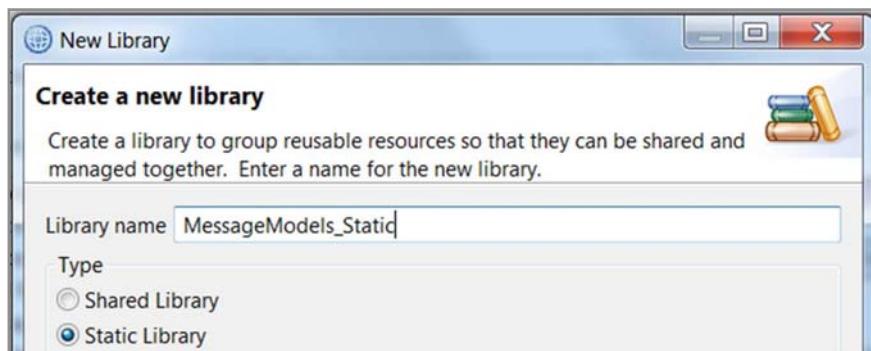
An *application* can contain any Integration Bus resource. It can also contain references to message flow dependencies, such as a Java project or message model. It can also reference libraries of reusable components.

In general, an application contains the main message flows and all the components that are required for a self-contained solution. You can deploy multiple applications to an integration server, and the visibility of a resource is restricted to the application in which it is contained.

A *library* is a logical grouping of related code, data, or both. A library contains references to reusable resources, such as a message model or map. A library can also refer to a resource that is contained in another library. Libraries can be embedded in an application to create a stand-alone copy for deployment. A message flow that is not part of an application can also access a library at run time at the integration server level. Use multiple libraries to group related resources (for example, by type or function). Libraries can be static or shared.

## Static libraries

- Application gets its own copy of the library and the resources that it contains
- To update a library, every application that uses the library must be repackaged and redeployed with the updates
- During packaging and deployment process, the library hierarchy is ignored and the libraries are flattened into a list



© Copyright IBM Corporation 2015

Figure 2-22. Static libraries

WM666 / ZM6661.0

### Notes:

One or more applications can reference a static library.

Changes that are made to a static library in the Integration Toolkit are available to all applications that reference that library. However, when the applications are packaged into a BAR file and deployed, each application has its own copy of the library and the resources that are contained in it. If you update a static library, you must repackage and redeploy each application that references that library.

## Shared libraries

- Applications use the artifacts in the deployed shared library
  - Applications automatically recognize updates to shared library
  - Application deployment fails if a required shared library is not already deployed
  - Cannot remove a shared library from an integration server while deployed applications are referencing it
- Identified as a shared library project type in the Integration Toolkit
- Shared libraries are deployed directly to the integration server
- Applications must explicitly reference the shared libraries
- Shared library hierarchy is preserved at run time
- Message flows are not allowed in a shared library
- Shared libraries do not have a running state; they cannot be started or stopped and no runtime threads are assigned to shared libraries

© Copyright IBM Corporation 2015

Figure 2-23. Shared libraries

WM666 / ZM6661.0

### Notes:

You might want to develop a set of common resources and make them available to multiple applications. If you want to deploy and manage just one copy of those common resources, use a shared library.

A shared library can be deployed directly to an integration server. Any application can reference the resources in that deployed shared library. If that shared library is updated, all referencing applications pick up the changes immediately.

A shared library must be deployed with or before an application that references it.

- If you deploy resources by dragging them onto an integration server, deploy the shared library first.
- If you deploy resources by adding them to a BAR file, ensure that you include any shared libraries that your application references.

Similarly, you cannot remove a deployed shared library while deployed applications are referencing it.

Shared libraries can reference other shared libraries, but not static libraries. Similarly, a static library cannot reference a shared library. An integration project cannot reference a shared library.



## Application, library, and project maintenance

- Applications, libraries, and projects can be:
  - Created
  - Updated from the repository
  - Imported or exported to or from the file system, compressed file, or arbitrary Eclipse container
- Create *active working sets* to filter the Application Development perspective to projects, applications, or libraries that are related to a particular scope or application
  - Working set is created automatically when you create an application or library

© Copyright IBM Corporation 2015

Figure 2-24. Application, library, and project maintenance

WM666 / ZM6661.0

### Notes:

Unless specified differently, all artifacts are created in the workspace directory of the Integration Toolkit installation folder. The `.metadata` directory of a workspace directory stores important information about the workspace structure, such as a project reference or a resource property.

A project is either open or closed in the Integration Toolkit. When a project is closed, it cannot be changed in the Integration Development perspective and it cannot be referenced from other projects, applications, or libraries. The resources of a closed project are not shown in the Integration Development perspective, but they are in the local file system. Closed projects require less memory, and closing a project at broker archive build time can reduce the build time.

You can also create working sets to filter the resources in the Application Development navigator to show only those resources that belong to a specific scope or application.

WebSphere Education 

## Creating a Working set

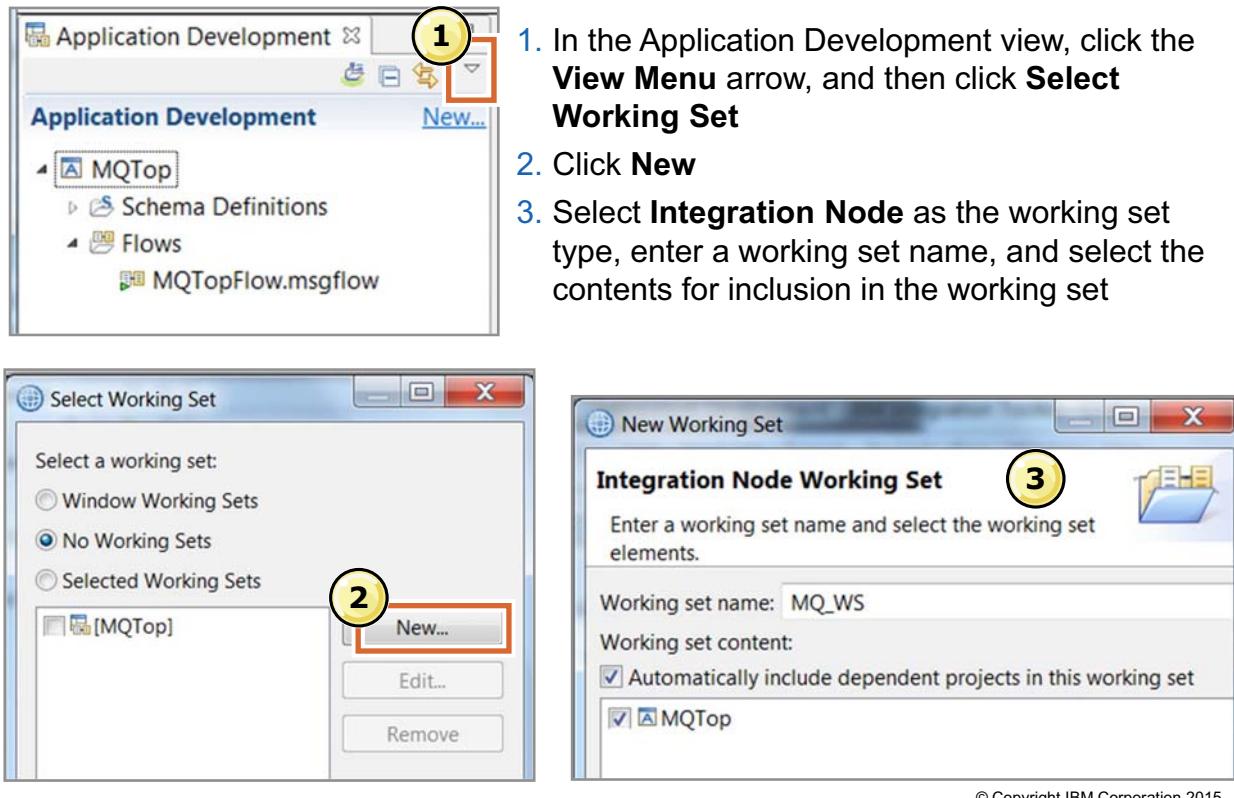


Figure 2-25. Creating a Working set

WM666 / ZM6661.0

### Notes:

A working set can be created to group related applications, projects, and libraries. The active working set is displayed at the top of the Application Development navigator.

A working set is created automatically when you create an application or library. When you select a working set, square brackets enclose these working sets, for example [MQTOP].

To create a working set:

1. Click the **View Menu** arrow in the Application Development navigator and then click **Select Working Set**.
2. On the **Select Working Set** window, you can select from existing working sets, or create one. Click **New**.
3. Select **Integration Node** for the working set type, click **Next**, and then enter a working set name and select the resources to include in the working set.

After you create a working set, only the artifacts that are associated with that working set are shown in the Application Development navigator. Showing fewer components simplifies the view of the workspace.

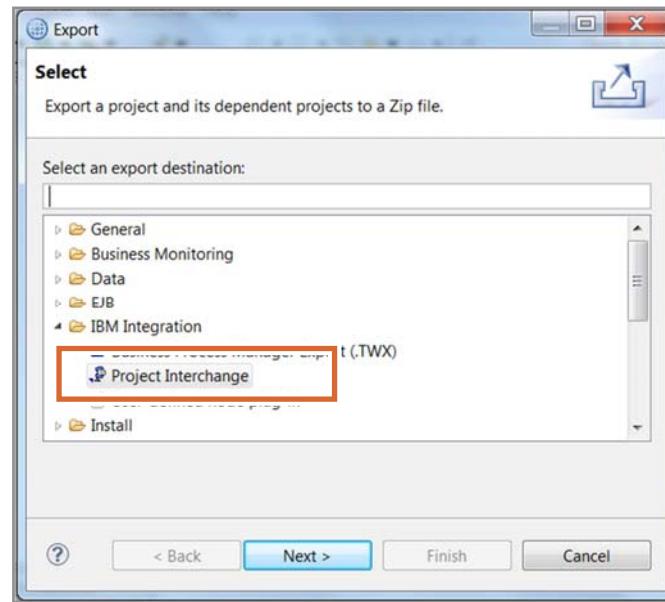
To show only the resources in a working set:

1. Click the arrow on the toolbar of the Application Development navigator and then click **Select Working Set**.
2. Select the relevant working set. If you select the working set that is created automatically for an application or library (as indicated by brackets), the Application Development navigator shows only the selected container and its contents.

WebSphere Education

## Exporting to create a Project Interchange file

- Create a Project Interchange file to export resources from the workspace
- **File > Export > IBM Integration > Project Interchange** copies resources from the workspace to:
  - The file system
  - A compressed file
  - Any other file container that Eclipse supports
- Eclipse operation
  - No transformation
  - No IBM Integration Bus involvement



© Copyright IBM Corporation 2015

Figure 2-26. Exporting to create a Project Interchange file

WM666 / ZM6661.0

### Notes:

Projects can be imported or exported. Exporting is the best choice for transferring files from a workspace to ensure that all components are included and categorized correctly.

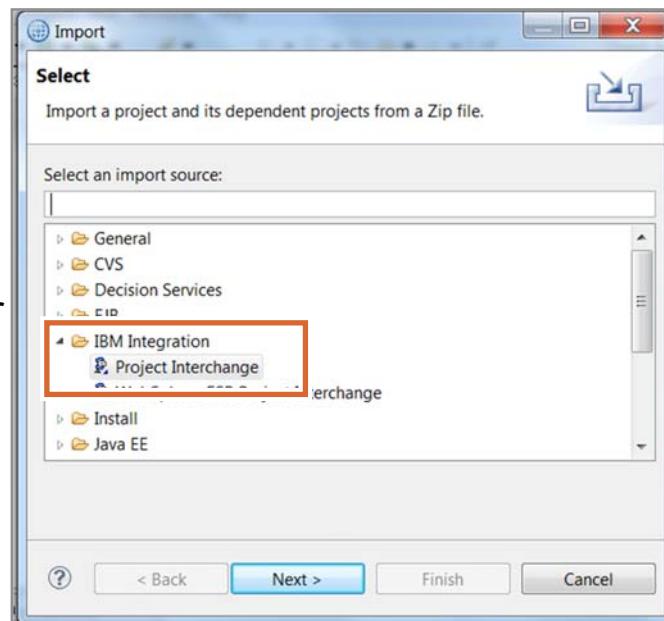
To export a project, application, or library to a Project Interchange file in the Integration Toolkit:

1. Click **File > Export**.
2. Under the **IBM Integration** folder, click **Project Interchange**, and then click **Next**.
3. Select the objects to include in the export file, enter the name and location of the project interchange file, and then click **Finish**. The project interchange file is created as a compressed file.



## Importing a Project Interchange file

- **File > Import > IBM Integration > Project Interchange**  
copies resources from Project Interchange file into the existing workspace
- Work with resources from previous versions of WebSphere Message Broker or IBM Integration Bus by importing a project interchange file into IBM Integration Bus V10



© Copyright IBM Corporation 2015

Figure 2-27. Importing a Project Interchange file

WM666 / ZM6661.0

### Notes:

You must use the IBM Integration Toolkit import and export functions to move application, library, or project resources to ensure that all references correctly reflect the new organization.

As shown in the figure, there are many import options. Most of the import operations in this course use Project Interchange files to import exercise objects into your workspace.

## Broker schemas (namespaces)

- Integration project subdirectory that provides a way to organize objects into a namespace to prevent duplicate names in the workspace or in the integration node
- Defined as the relative path from the project source directory to the flow name
- Can add more structure by using a period (.) in broker schema

Example: Namespace `MQ80.Demo` in project `MQ80_FLOWS` and workspace of `IIBDev` resolves to path: `IIBDev\MQ80_FLOWS\MQ80\Demo`

- When moving or copying flows into another schema:
  - Associated files (`*.esql`, `*.map`) are not automatically moved or copied
  - References must be repaired manually
- Broker schema is propagated to the integration node with deployment

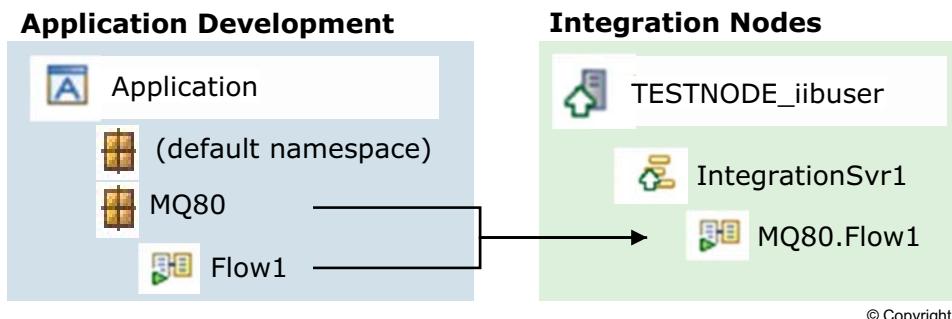


Figure 2-28. Broker schemas (namespaces)

WM666 / ZM6661.0

### Notes:

Integration Bus objects are identified by using a combination of a namespace and a name, referred to as a fully qualified name.

Fully qualified names make it easier to identify and locate objects, and to correct broken references. For example, if an object is renamed, all references to that object are broken. If an object is substituted for another object with the same name, all the broken references are corrected. This concept, called *name linking*, is important when working in a team environment. It means that you can share files in a repository and concurrently modify, add, and delete objects in the message flow application. When you integrate the various parts of the application, you can detect and resolve broken references to objects that were previously moved, renamed, or deleted.

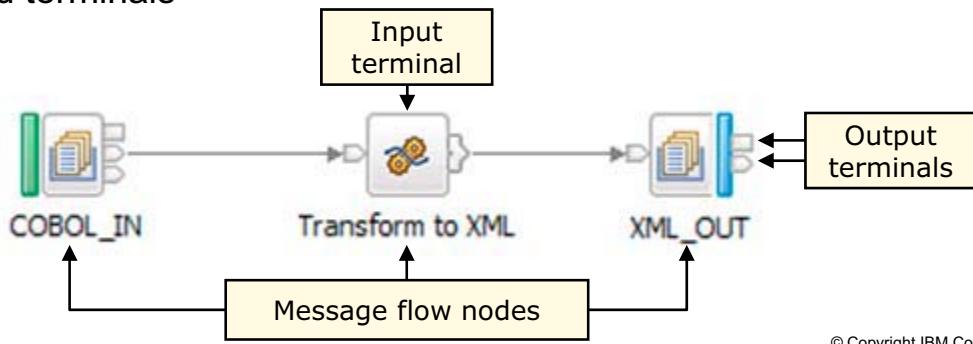
A *broker schema* provides a namespace for objects that are created in the Integration Toolkit. The broker schema is defined as the relative path from the project source directory to the flow name.

Each broker schema represents a unique name scope. So, you can create two different message flows that share a name within two different broker schemas. The broker schemas ensure that the two message flows are recognized as separate resources. The two message flows with the same base name, are considered unique.

When you first create a message flow project, a default broker schema that is named `default` is created within the project. The `default` broker schema is not displayed in the Integration Toolkit unless other broker schemas are defined.

## Message flow nodes

- Message flow nodes complete actions in a message flow
  - Transform or enrich the incoming message
  - Provide routing control for the movement of the message through the message flow (based on message content or other factors)
  - Provide other specialized functions such as retrieving or sending a message by using a message transport, accessing a database, and aggregating multiple messages
- Most message flow nodes operate on an incoming logical message, and then pass the message to the next message flow node
- Messages enter and leave a node through input and output points called terminals



© Copyright IBM Corporation 2015

Figure 2-29. Message flow nodes

WM666 / ZM6661.0

### Notes:

A message flow node is a self-contained module that completes an action in the message flow. Most message flow nodes act on an incoming message, either by modifying the message or by using the message content to change the routing in the message flow.

Some types of nodes alter the flow of messages within a message flow, by querying the contents of the message, or by doing operations such as retrieving rows from a relational database. Other nodes access message transports to send or receive messages, such as through IBM MQ, web services, JMS, or files. Other nodes exchange data with enterprise information systems.

In most instances, a message enters the node through an input terminal, and exits the node through one or more output terminals. Not all nodes have all terminals. Some have only output terminals but no input terminals, while others have only an input terminal. Some nodes have multiple output terminals, and send the message through one or more terminals, depending on the processing that occurs within the node. With some nodes, you can add terminals to allow multiple routing paths.

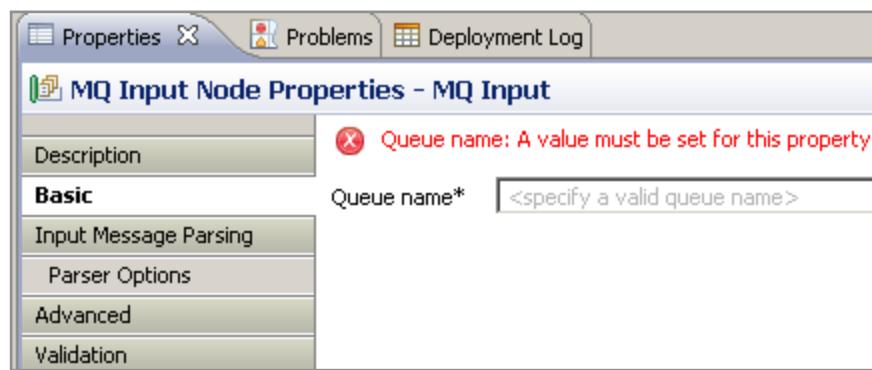
To connect message flow nodes together, you create a connection that is called a wire. Wiring two nodes together provides a path on which a message can flow. When you are wiring nodes in a message flow, be careful that you are wiring the correct nodes and terminals. Wiring the wrong

terminals results in a message flow that does not process messages as expected. It is also possible to send a message through a terminal that is unwired, which discards the message with no warning from the integration node.



## Message flow node properties

- Control how a message flow node operates
  - Mandatory:** You must set values for these properties at design time
  - Optional:** You can set values for these properties at design time; otherwise, a default is used
  - Configurable:** You can override values in the deployment file or at run time
  - Promoted:** Elevates the property to the message flow level instead of the individual node level; the property value can then be changed at the message flow level



© Copyright IBM Corporation 2015

Figure 2-30. Message flow node properties

WM666 / ZM6661.0

### Notes:

With most message flow nodes, you do not write any code or embed any logic. Instead, you control how a node operates by modifying its properties. You can construct message flows more quickly and with fewer errors by modifying node properties than by writing code. Some nodes require that you write code; these nodes are examined in detail later in this course.

Multiple types of properties are associated with a message flow node. You configure some properties at design time. Other properties are configured when you deploy the message flow to the integration node. You can also configure some properties at run time, after deployment, for greater flexibility. Run time properties are used to modify how the message flow operates without having to re-edit the message flow and then redeploy it.

Node properties are accessed in the **Properties** view. This view has multiple subordinate tabs, each of which has one or more sets of properties.

- Mandatory properties** are those properties that you must set before you can deploy the message flow to the integration node, or the message flow node cannot operate. In the example that is shown on the figure, the MQ Input node **Queue name** property is mandatory because the queue from which the node retrieves a message must be specified. Unset mandatory properties prevent a message flow from being deployed.

- **Optional properties** are those properties that you can configure at design time, but if you do not, default values are used.
- **Configurable properties** and **Promoted properties** are used to alter the properties of message flow nodes at the level of the BAR file or at the integration server or integration node level (where the message flow runs). For example, a message flow might access a particular database while in development, but a different one when the message flow is running in production. With configurable and promoted properties, you can modify the database information for all message flow nodes that access the database. By making one change, you override all references to the database used in development to the database that the message flow uses in production, without re-editing all those nodes. Configurable properties and promoted properties are covered later in the course.

## Message processing nodes overview

- Many integrated process nodes
  - Input (various transport protocols)
  - Output (various transport protocols)
  - Routing inside flow
  - Transformation
  - Database operations
  - Fan-out and fan-in (aggregation) of messages\*
  - Timer\*
  - File
  - Debugging and exception handling
- Support for vendor-written and user-written plug-in nodes

**\*Note:** Only available if you specify an IBM MQ queue manager on the integration node

© Copyright IBM Corporation 2015

Figure 2-31. Message processing nodes overview

WM666 / ZM6661.0

### Notes:

The figure lists some of the most common types of nodes that IBM Integration Bus supports.

If the standard nodes provided with IBM Integration Bus do not provide the required function, it is possible to have user or vendor-written nodes included in the Integration Toolkit. These nodes can then be used in a message flow.

To help you locate nodes when designing message flows, nodes are categorized in drawers in the Integration Toolkit Message Flow editor.

## Message transport nodes

- IBM MQ
  - MQInput
  - MQOutput
  - MQReply
  - MQGet
  - MQHeader
- JMS
  - JMSInput
  - JMSOutput
  - JMSReceive
  - JMSReply
  - JMSHeader
  - JMSMQTransform
  - MQJMSTransform
- Web Services
  - SOAPInput
  - SOAPReply
  - SOAPRequest
  - SOAPAsyncRequest
  - SOAPAsyncResponse
  - SOAPEnvelope
  - SOAPExtract
  - RegistryLookup
  - EndpointLookup
- Windows .NET
  - .NETInput
- HTTP
  - HTTPAsyncRequest
  - HTTPAsyncResponse
  - HTTPInput
  - HTTPReply
  - HTTPRequest
  - HTTPHeader
- IBM MQ Managed File Transfer
  - FTEInput\*
  - FTEOutput\*

**\*Note:** Only available if you specify an IBM MQ queue manager on the integration node

© Copyright IBM Corporation 2015

Figure 2-32. Message transport nodes

WM666 / ZM6661.0

### Notes:

This figure lists the IBM Integration Bus message transport nodes. The nodes are grouped by their type, which is also the drawer name in the Message Flow editor palette.

Some nodes, such as the IBM MQ Managed File Transfer node, require that an IBM MQ queue manager is specified for the integration node.

## Message transport-related nodes

- **File**
  - FileInputStream
  - FileOutputStream
  - FileRead
  - CDInput\*
  - CDOOutput\*
- **TCP/IP**
  - TCPIPClientInput
  - TCPIPClientOutput
  - TCPIPClientReceive
  - TCPIPServerInput
  - TCPIPServerOutput
  - TCPIPServerReceive
- **WebSphere Adapters**
  - JDEdwardsInput
  - JDEdwardsRequest
  - PeopleSoftInput
  - PeopleSoftRequest
  - SAPInput
  - SAPReply
  - SAPRequest
  - SiebelInput
  - SiebelRequest
- **SCA**
  - SCAInput
  - SCAResponse
  - SCAAsyncRequest
  - SCAAsyncResponse
  - SCAResponse
- **Other external systems**
  - CORBARequest
  - CICSRequest
  - IMSRequest

**\*Note:** Only available if you specify an IBM MQ queue manager on the integration node

© Copyright IBM Corporation 2015

Figure 2-33. Message transport-related nodes

WM666 / ZM6661.0

### Notes:

This figure lists the nodes that are used to connect to external sources of messages. These nodes are not message transports in the strictest sense, but they do provide a way to send and receive messages for processing within a message flow.

- CDInput and CDOOutput nodes connect to IBM Sterling Connect:Direct
- SCA nodes connect to WebSphere Process Server
- TCPIP nodes connect to TCP/IP input and output streams, as a client or as a server
- CORBA nodes connect to CORBA Internet Inter-ORB Protocol (IIOP) applications
- CICSRequest node connects to CICS applications, by sending Distributed Program Link (DPL) requests over TCP/IP-based IP InterCommunications protocol
- IMSRequest node is used to send a request to run a transaction on a local or remote IMS system, and wait for a response
- WebSphere Adapters nodes interact with various enterprise information systems such as SAP.

## Other message processing nodes (1 of 2)

- **Routing**
  - Filter
  - Route
  - Label
  - RouteToLabel
  - Publication
  - AggregateControl\*
  - AggregateReply\*
  - AggregateRequest\*
  - Collector\*
  - Resequence\*
  - Sequence\*
- **Transformation**
  - Compute node
  - JavaCompute
  - XSLTransform
  - Mapping
  - .NETCompute
- **Database**
  - Database
  - DatabaseInput
  - DatabaseRetrieve
  - DatabaseRoute
- **Construction**
  - Input
  - Output
  - Throw
  - Trace
  - TryCatch
  - FlowOrder
  - Passthrough
  - ResetContentDescriptor

**\*Note:** Only available if you specify an IBM MQ queue manager on the integration node

© Copyright IBM Corporation 2015

Figure 2-34. Other message processing nodes (1 of 2)

WM666 / ZM6661.0

### Notes:

**Routing** nodes control how messages flow through a message flow. They act as decision points and points where you can alter where a message branches. These nodes are also used for aggregating multiple responses.

**Transformation** nodes modify the contents of the message.

**Database** nodes interact with relational database management systems. You can do any database operation such as select, insert, update, and delete. You can also run stored procedures. You use the contents of the logical message tree to complete database operations, and the results of database queries can be used to modify the message tree.

**Construction** nodes control the flow of messages within a message flow. They are used to construct subflows, manage explicit runtime error handling, and do miscellaneous operations on the message tree.



## Other message processing nodes (2 of 2)

- **Email**
  - EmailInput
  - EmailOutput
- **Validation**
  - Validate
- **Security**
  - SecurityPEP
- **Timer**
  - TimeoutControl\*
  - TimeoutNotification\*
- **Business Decisions**
  - DecisionService

**\*Note:** Only available if you specify an IBM MQ queue manager on the integration node

© Copyright IBM Corporation 2015

Figure 2-35. Other message processing nodes (2 of 2)

WM666 / ZM6661.0

### Notes:

The Integration Toolkit provides many miscellaneous message processing nodes. These nodes are used to do such tasks as:

- Sending and receiving mail by using SMTP and POP servers
- Validating the XML content of a message
- Calling the message flow security manager during message processing
- Initiating timer-driven message flow operations that are based on calendar dates and times

## Using patterns for simplified development

- A pattern is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context
- Creates top-down, parameterized connectivity solutions
- Reduces common problems in flow development
- Communicates good practices to the IBM Integration Bus community
- Complements existing bottom-up construction for custom connectivity
- Reduces time-to-value for solution development
- Catalog of IBM Integration Bus patterns is available on OT4I GitHub Pattern Repository
- Patterns can be user-defined

© Copyright IBM Corporation 2015

Figure 2-36. Using patterns for simplified development

WM666 / ZM6661.0

### Notes:

A start-from-the-beginning approach to design in IBM Integration Bus can be intimidating to new users, and it is not the fastest way to develop a solution. It is much easier to start with a template and customize it.

Many message flows show common behaviors, such as providing web services, message routing, and putting a message on a queue. A major feature of IBM Integration Toolkit is patterns-based development. A patterns-based model gives you the ability to start creating solutions from predefined templates, which are referred to as *patterns*, for top-down modeling.

A pattern is a tested solution to a specific IBM Integration Bus application problem. By generalizing and abstracting the solution to the problem, a pattern is created. To allow the pattern to fit various application circumstances, the pattern is built by using configurable parameters.

A catalog of patterns is available on the OT4I GitHub Pattern Repository. After you download the pattern, you use the pattern wizard to help you configure the pattern to your specific needs.

After you configure the appropriate parameters, the pattern instance is generated. Generating a pattern instance creates the customized artifacts and components that are needed to implement your solution.



## Pattern concepts

- Pattern parameter values customize the generated pattern instance
- Design-time parameters cannot be modified after a pattern instance is generated
- Runtime properties
  - Affect behavior of flow during processing
  - Can be modified after development with the BAR file editor
- Regeneration overwrites any changes; it does not merge

© Copyright IBM Corporation 2015

Figure 2-37. Pattern concepts

WM666 / ZM6661.0

### Notes:

Patterns-based development takes advantage of common practices by standardizing message flows. It eases some of the anxiety that new users can experience when presented with so many node choices, and it gives a quick start to message flow design. A pattern reduces common problems by providing templates that solve common problems and promote good practices.

A pattern is production-ready and exemplifies good practices.

The IBM Integration Toolkit gives you access to patterns to assist you with development. You can also create user-defined patterns.

A *pattern instance* is a copy of predefined template that solves a problem, such as file processing or service virtualization.

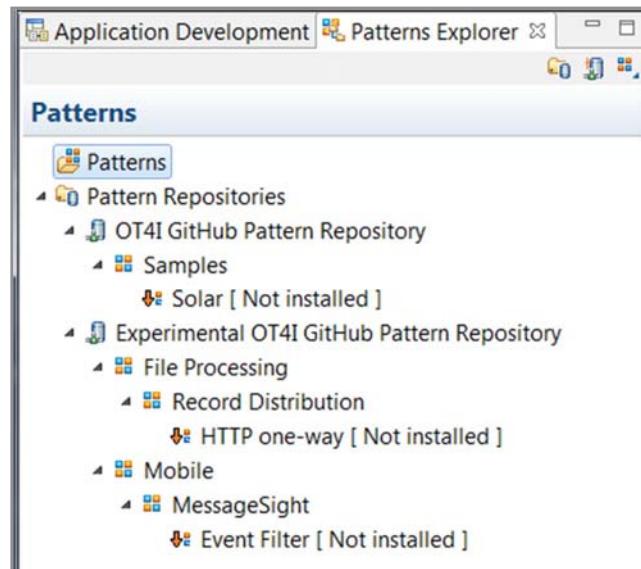
Each pattern has values that are known as *pattern parameters*. Pattern parameters are parameters that customize and configure a pattern. The pattern parameters that you configure depend on the particular pattern, and on the options that you enable for that pattern. An example of a pattern parameter is a queue name from where messages are read.

It is possible to regenerate an IBM Integration Bus application from a pattern; however, regenerating does not merge the new version with the old version. It overwrites the existing generated application.



## Patterns Explorer

- Pattern categories group similar solutions together
  - Describes a class of solutions
  - Leaf nodes are the patterns
- Pattern specification describes each pattern
- Each pattern name is appended with:
  - **[Not Installed]** if the pattern is not installed in the IBM Integration Toolkit
  - **[Installed]** if the pattern is already installed



© Copyright IBM Corporation 2015

Figure 2-38. Patterns Explorer

WM666 / ZM6661.0

### Notes:

The Patterns Explorer lists the available patterns in the IBM Integration Toolkit and the GitHub Pattern Repository.

- The **Patterns** folder contains patterns that are downloaded and installed in the Integration Toolkit.
- The **Pattern Repositories** folder lists the patterns that are available in the GitHub Pattern Repository.



## Getting patterns from the GitHub repository

1. Right-click **OT4I GitHub Pattern Repository** under **Pattern Repositories**, and click **Connect** to display the categories of the patterns that are available in the repository.
2. Expand a pattern category to list the patterns that are available in that category.
3. Right-click the pattern to download and click **Download and Install**. This menu option is available only if the pattern is not already installed.
4. When the security warning is displayed, click **OK**.
5. The pattern is downloaded, installed, and listed as **[Installed]** in the Patterns Explorer.
6. The pattern is listed under the **Patterns** folder in the Patterns Explorer.

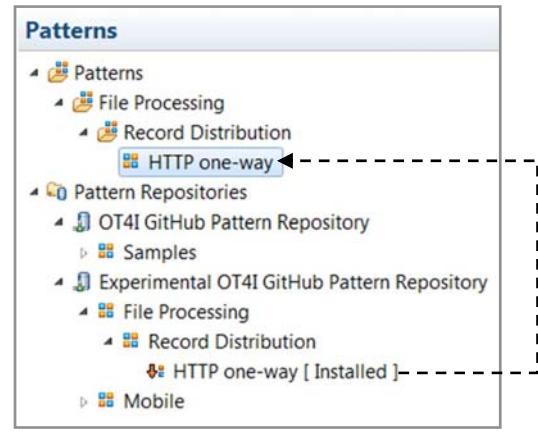


Figure 2-39. Getting patterns from the GitHub repository

WM666 / ZM6661.0

### Notes:

This figure shows the steps for getting patterns from the GitHub Pattern Repository.



## Generating a pattern instance (1 of 2)

**Record Distribution to HTTP: one-way pattern**

Use the Record Distribution to HTTP: one-way pattern to connect file-based and transactional integration.

Use this pattern in the following situations:

- You have batches of transactions that are received as files from local or remote systems.
- Your transactions cannot be sent as individual transactions. For example:
  - The source systems support file-based integration capabilities only.
  - The business requirements mandate that transactions are batched; for example, the collection of all transactions within a working day.
  - You must conform to existing working practices; for example, you must use files that contain EDI transactions.
- Your target systems are designed to accept individual transactions. Such systems might take input from near real-time systems and from batch files, and therefore present a transactional interface.
- You have batch files that contain transactions that must be distributed to one or more systems.
- If any failures occur, it must be acceptable for the entire file to be reprocessed.

**1** Review the pattern specification

**2** Click **Create New Instance**

© Copyright IBM Corporation 2015

Figure 2-40. Generating a pattern instance (1 of 2)

WM666 / ZM6661.0

### Notes:

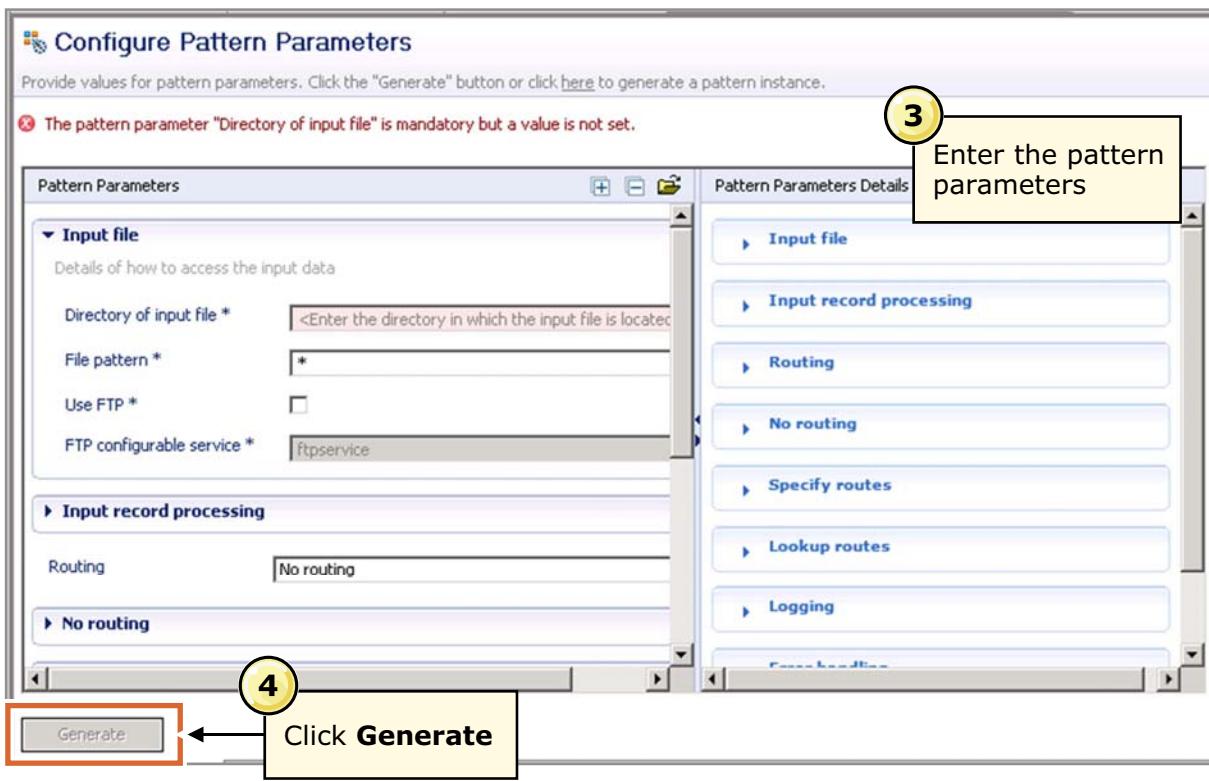
Each pattern includes a pattern specification that describes the pattern.

As shown in the figure, the next steps are to review the pattern specification to determine whether the pattern addresses your needs.

In the example in the figure, the **Record Distribution to HTTP: one-way pattern** is selected. The specification identifies when to use the pattern and the problem that it solves. In this example, the pattern reads records from a file and routes each record to an HTTP output node.

After reviewing the pattern specification, click **Create New Instance**.

## Generating a pattern instance (2 of 2)



© Copyright IBM Corporation 2015

Figure 2-41. Generating a pattern instance (2 of 2)

WM666 / ZM6661.0

### Notes:

As the figure shows, the final steps for creating a pattern instance are for you to enter the pattern instance parameters, and then to click **Generate**.

Clicking **Generate** creates an application that contains pattern artifacts such as message flows and scripts. Service information that describes the parameter usage, is included with the pattern.



## Configuring pattern parameters

**Configure Pattern Parameters**

Provide values for pattern parameters. Click the "Generate" button or click [here](#) to generate a pattern instance.

**Pattern Parameters**

- Sample**
- Input file** (selected)
 

Details of how to access the input data

  - Input Directory \*
  - Filename pattern \*
  - Use FTP \*
  - FTP configurable service
- Routing**

Specify the routing type

  - Routing \*
  - Single Destination**

**Pattern Parameters Details**

- Input file**
- Routing**
- Single Destination**
- Specify Routes**

- Parameters are logically grouped into sections
- An asterisk (\*) indicates mandatory parameters
- A red "X" indicates missing parameters
- Fields are watermarked and prepopulated
- Detailed help is available for each pattern parameter

© Copyright IBM Corporation 2015

Figure 2-42. Configuring pattern parameters

WM666 / ZM6661.0

### Notes:

Pattern parameters are logically grouped into sections to distinguish between configuration and runtime properties, such as error handling.

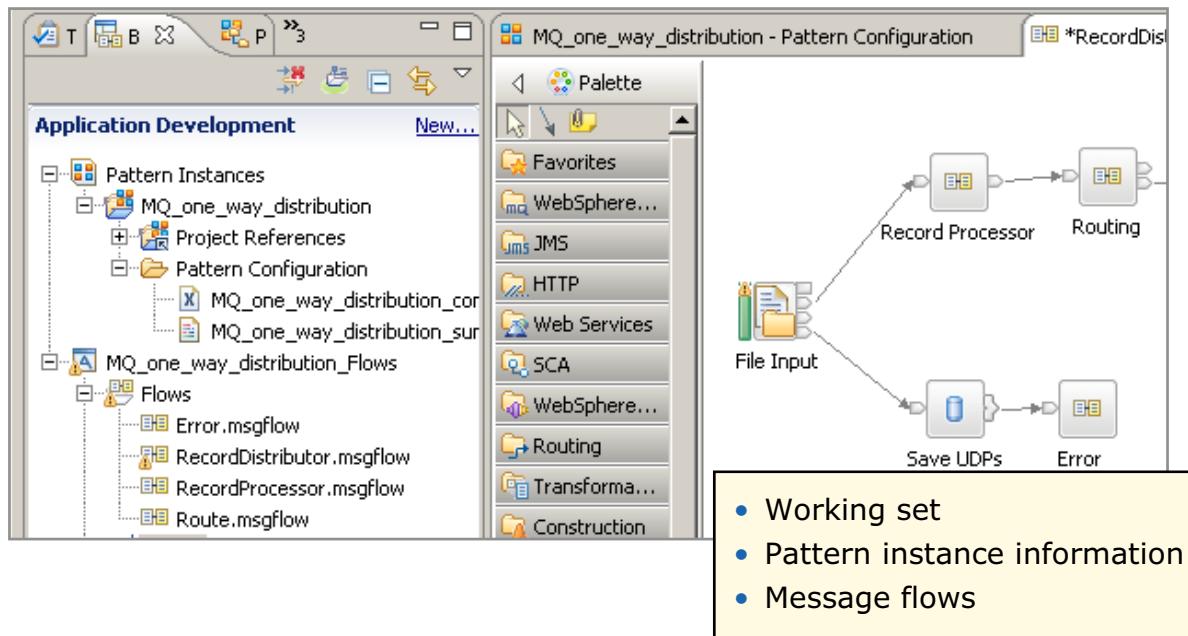
In some cases, parameters can be prepopulated with available message models, for example.

Patterns have both mandatory and optional parameters. An asterisk (\*) identifies mandatory parameters. A red X indicates missing parameters.

After you specify all the required parameters (and any optional ones you want to use), click **Generate** to generate the pattern instance.



## Generated artifacts



© Copyright IBM Corporation 2015

Figure 2-43. Generated artifacts

WM666 / ZM6661.0

### Notes:

Generating the pattern creates a working set, a **Pattern Instances** project, and an application.

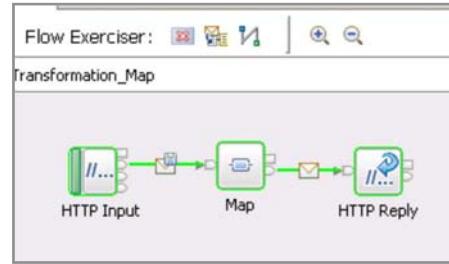
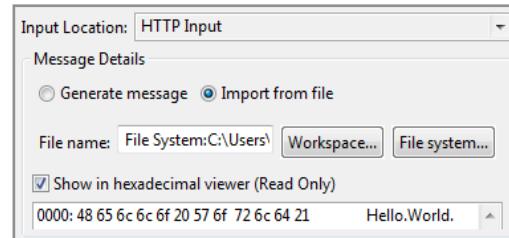
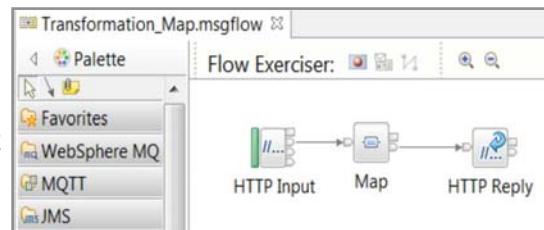
The **Pattern Instances** project contains references to all other projects in the workspace that relate to the pattern instance. The **Pattern Instances** also contains a pattern instance configuration file that stores the pattern parameter values. This configuration file stores the pattern parameters that you configured, so that you can review them later.

Generating a **Pattern Instances** project also creates an IBM Integration Bus application that typically contains message flows and other Integration Bus resources that implement the pattern.



## Integrated testing with IBM Integration Toolkit

- Unit and regression test
  - Verify flow behaviors and migration
  - Fix and refactor behavior during development
  - Continuous Integration with regression test
  
- Client and direct injection options
  - Import, view, and edit test data
  - Inject messages over transports
  - Capture mock inputs for later replay
  - Build regression suites from test cases
  
- Observe captured data paths
  - Move back and forth
  - View all parts of the message assembly
  - View, start, and stop data recording
  - Select from multiple injected messages in single data capture session



© Copyright IBM Corporation 2015

Figure 2-44. Integrated testing with IBM Integration Toolkit

WM666 / ZM6661.0

### Notes:

To check that a message flow or integration service is processing messages as expected, you can send messages to the flow by using the Flow exerciser in the Integration Toolkit. You can then use the Flow exerciser to show the path that each message took, and view the structure and content of the logical message tree at any point in a message flow.

You can send messages to the message flow by using one of the following options:

- Use an external tool or client to form and send one or more input messages to the flow. After the flow processes the input messages, click the **View path** icon in the Flow exerciser toolbar to highlight message paths on the flow.
- If you are using an integration service, or your message flow contains MQInput, HTTPInput, or SOAPIInput nodes, click the **Send message** icon in the Flow exerciser toolbar. Then, you can use the **Send Message** dialog box to create an input message or select an existing input message or recorded message and send it to the flow.



## IBM Integration Toolkit Flow exerciser

- Send messages to the flow by using the Flow exerciser or an external client
- Show the path that each message took
- View the structure and content of the logical message tree at any point in a message flow
- Save recorded messages for replay
- Recording mode is enabled and disabled on an integration server
  
- Must have the following components:
  - Integration service, stand alone message flow, or an application that includes a message flow
  - Integration node and integration server that are accessible from the IBM Integration Toolkit

**Note:** The Flow exerciser cannot be used with a message flow that is defined in a REST API.

© Copyright IBM Corporation 2015

Figure 2-45. IBM Integration Toolkit Flow exerciser

WM666 / ZM6661.0

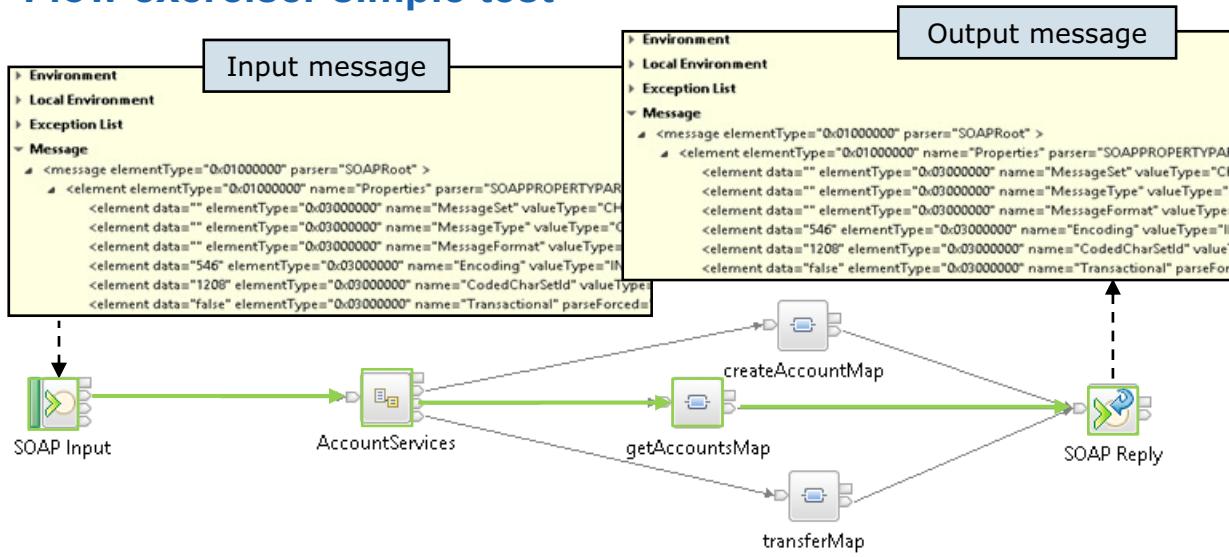
### Notes:

Before you use the Flow exerciser, ensure that you have the following components:

- A resource such as an integration service, a stand-alone message flow, or an application that includes a message flow.
- An integration node and associated integration server that are accessible from the IBM Integration Toolkit.



## Flow exerciser simple test



1. Create and send a message to the input node of the message flow
2. Highlight the message path on the message flow and any subflows that are associated with the message flow
3. Display the content of the logical message tree for a message that passed through a connection in the message flow
4. Save the content of the logical message tree as a recorded message that you can send to the message flow later

© Copyright IBM Corporation 2015

Figure 2-46. Flow exerciser simple test

WM666 / ZM6661.0

### Notes:

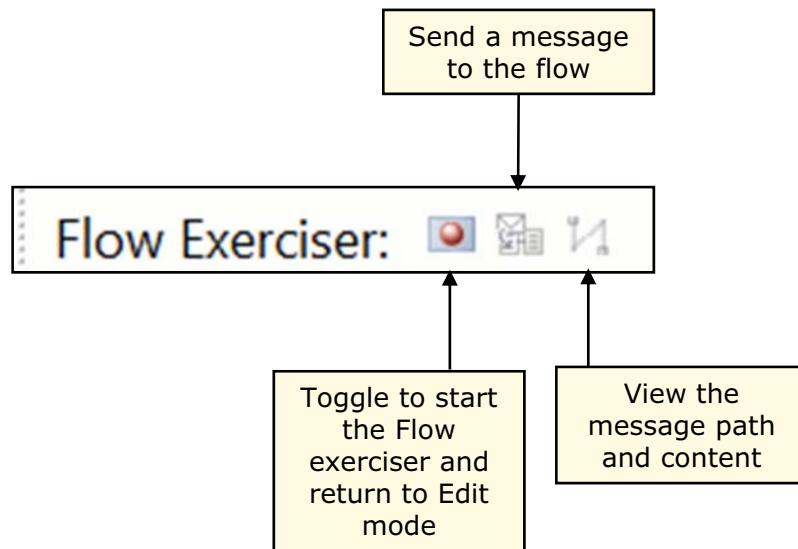
This figure lists the basic steps for using the Flow exerciser for testing a message flow.

After you record the run of a flow (as shown by the highlighted paths in green), you can save any input messages for testing the flow later. When you send (inject) a message to the flow, the flow is initiated directly; it does not need an external transport to send an input message to the node.

Saved messages use the internal XML format, and can be viewed under **Other Resources** in the Application Development view. The internal XML format for any recorded message can also be viewed by clicking **Copy unformatted message** from the menu.



## Flow exerciser toolbar



© Copyright IBM Corporation 2015

Figure 2-47. Flow exerciser toolbar

WM666 / ZM6661.0

### Notes:

The Flow exerciser toolbar in the Message Flow editor, contains three icons.



## Running the Flow exerciser

1. Open the message flow or integration service description in the Integration Toolkit
2. Click the **Start Flow** exerciser icon
3. Send messages by using one of the following options:
  - If you are using an integration service or the message flow contains MQInput, HTTPInput, or SOAPInput node, click the **Send message** icon
  - If you are using an external tool or client to send one or more input messages, click the **View path** icon to highlight message paths on the flow after the input messages are processed
4. Click a highlighted connection to view the data that passed through the connection
5. To return the message flow to edit mode, click the **Return flow to edit mode** icon

© Copyright IBM Corporation 2015

Figure 2-48. Running the Flow exerciser

WM666 / ZM6661.0

### Notes:

This figure lists the steps for running the Integration Toolkit Flow exerciser.



## Viewing message paths and connections

- If you send more than one message to the flow, inspect each highlighted connection to see which messages passed through that connection
- If you send a single message to the flow and the message passes through a connection multiple times, the logical message tree is captured as a separate message instance each time the message passes through a connection
- By default, a maximum of 200 message instances are displayed in the message flow

© Copyright IBM Corporation 2015

Figure 2-49. Viewing message paths and connections

WM666 / ZM6661.0

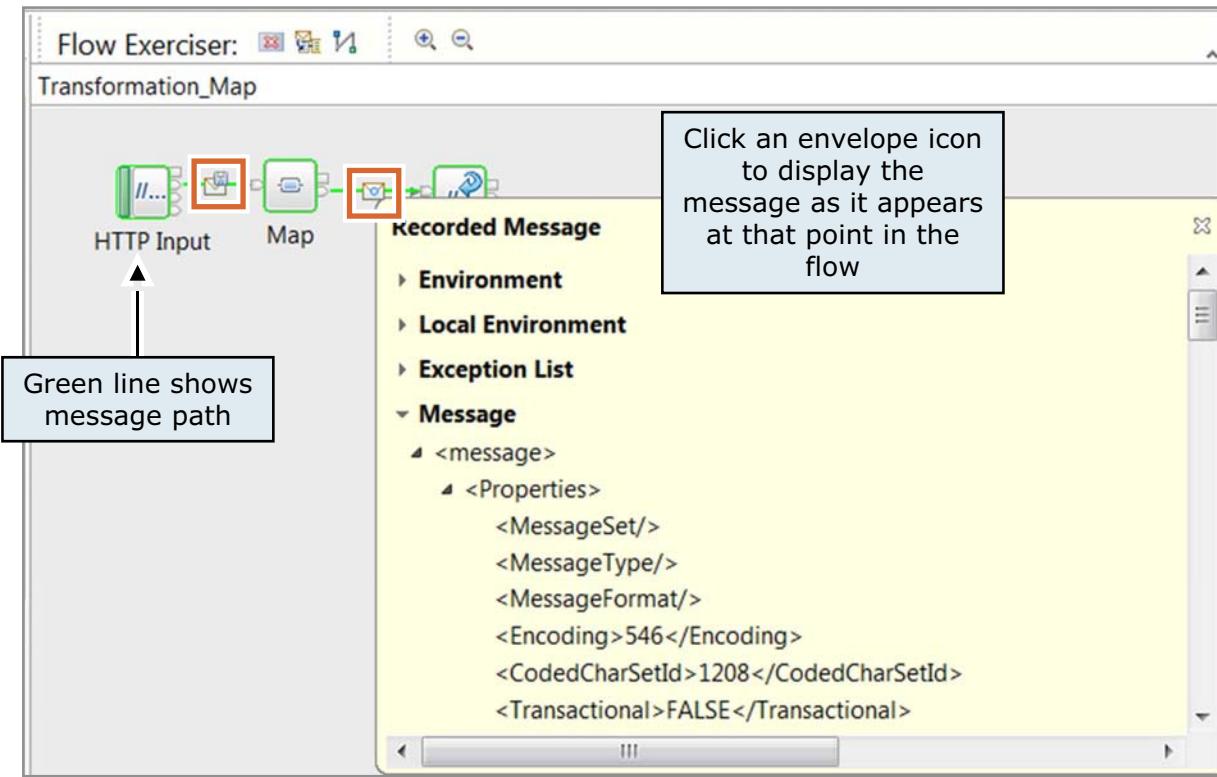
### Notes:

If the number of message instances that are captured exceeds the number that is configured in the preferences, you are prompted to choose whether to view the configured number of recorded messages, or whether to view all the recorded messages:

- If you opt to view the configured maximum number of recorded messages, you might not see whole sequences of messages.
- If you opt to view all the messages, there might be a performance impact.



## Flow exerciser example



© Copyright IBM Corporation 2015

Figure 2-50. Flow exerciser example

WM666 / ZM6661.0

### Notes:

The figure shows an example of the Flow exerciser. The green line shows the message path. You can click an envelope icon to display the message as it appears at that point in the message flow.

## Unit summary

Having completed this unit, you should be able to:

- Describe how IBM Integration Bus does basic message processing
- Describe the components of a message flow application and message processing nodes
- Describe the basic structure of a logical message tree
- Use patterns as a starting point for developing message flow applications
- Import resources to and export resources from the IBM Integration Toolkit
- Use the IBM Integration Toolkit Flow exerciser to test a message flow application
- Use the IBM Integration Toolkit to check the status of the integration node, integration server, and message flow application

© Copyright IBM Corporation 2015

Figure 2-51. Unit summary

WM666 / ZM6661.0

### Notes:



## Checkpoint questions

1. Which two statements are true for IBM Integration Bus?
  - a. Each message flow runs as a separate process in the integration node.
  - b. A message flow runs as a thread within one or more integration servers.
  - c. The integration node offers various ways to scale message throughput (parallel processing).
  - d. It stores all data about integration servers and deployed flows in an integration node database, which must be explicitly created.
  
2. Which two statements are true for the IBM Integration Toolkit?
  - a. It is the single point of control for all IBM Integration Bus development, administrative, and management tasks.
  - b. It runs in the same environments as the integration node.
  - c. It is not necessary to have an IBM MQ server (queue manager) installed on the developer workstation.
  - d. A user can deploy the same message flow to multiple integration nodes.

© Copyright IBM Corporation 2015

Figure 2-52. Checkpoint questions

WM666 / ZM6661.0

### Notes:

Write your answers here:

1.

2.

## Checkpoint answers

1. Which two statements are true for IBM Integration Bus?
  - a. Every message flow runs as a separate process in the integration node.
  - b. A message flow runs as a thread within one or more integration servers.
  - c. The integration node offers various ways to scale message throughput (parallel processing).
  - d. It stores all data about integration servers and deployed flows in an integration node database, which must be explicitly created.

**Answer:** b and c.

2. Which two statements are true for the IBM Integration Toolkit?
  - a. It is the single point of control for all IBM Integration Bus development, administrative, and management tasks.
  - b. It runs on the same environments as an integration node.
  - c. It is not necessary to have the IBM MQ server (queue manager) installed on the developer workstation.
  - d. A user can deploy the same message flow to multiple integration nodes.

**Answer:** c and d.

© Copyright IBM Corporation 2015

Figure 2-53. Checkpoint answers

WM666 / ZM6661.0

### Notes:

## Exercise 1

Importing and testing a message flow



© Copyright IBM Corporation 2015

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

10.1

Figure 2-54. Exercise 1

WM666 / ZM6661.0

### Notes:



## Exercise objectives

After completing this exercise, you should be able to:

- Import an IBM Integration Bus project interchange file
- Use the Message Flow editor to examine the message flow components and properties
- Test the message flow by using the IBM Integration Toolkit Flow exerciser

© Copyright IBM Corporation 2015

Figure 2-55. Exercise objectives

WM666 / ZM6661.0

### Notes:

See the *Student Exercise Guide* for detailed instructions.



# Unit 3. Creating message flow applications

## What this unit is about

In this unit, you learn how to manually define an IBM Integration Bus application. You learn how to add and connect message flow nodes, define message flow node properties, and test the message flow application. You also learn how to use the IBM Integration web user interface to monitor the integration node, integration server, and message flow application at run time.

## What you should be able to do

After completing this unit, you should be able to:

- Create a message flow application
- Add nodes to a message flow
- Package and deploy message flow applications and resources
- Use the IBM Integration web user interface to monitor the integration node, integration server, and message flow

## How you will check your progress

- Checkpoint
- Hands-on exercise

## References

- IBM Knowledge Center
- For information about the IBM Integration web user interface status fields, see the IBM Integration API documentation:  
[www.ibm.com/support/knowledgecenter/SSMKHH\\_10.0.0/com.ibm.etools.mft.doc/bz90370\\_.htm](http://www.ibm.com/support/knowledgecenter/SSMKHH_10.0.0/com.ibm.etools.mft.doc/bz90370_.htm)



## Unit objectives

After completing this unit, you should be able to:

- Create a message flow application
- Add nodes to a message flow
- Package and deploy message flow applications and resources
- Use the IBM Integration web user interface to monitor the integration node, integration server, and message flow

© Copyright IBM Corporation 2015

Figure 3-1. Unit objectives

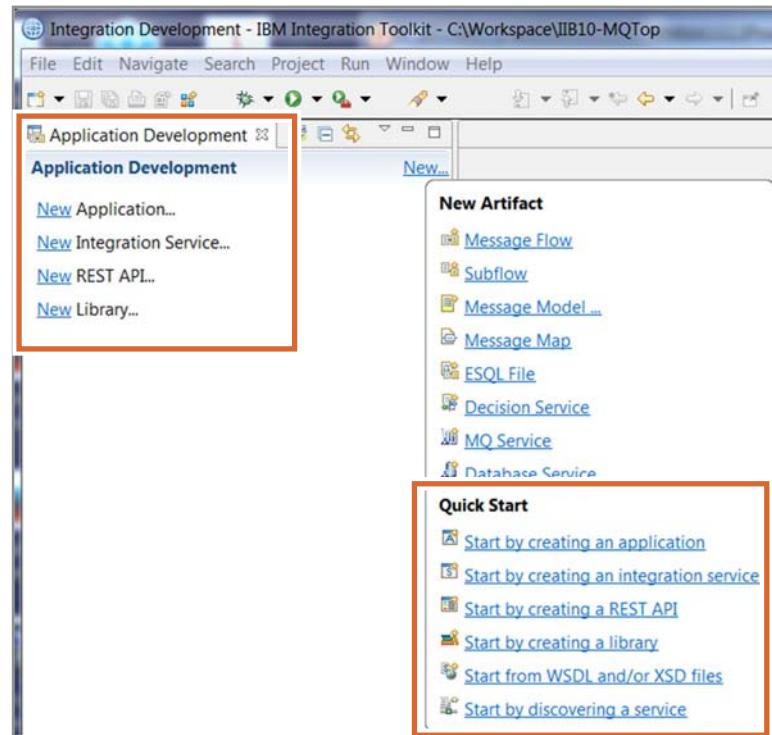
WM666 / ZM6661.0

### Notes:



## Quick starts

- Start building message flow applications
  - By creating an application
  - By creating an integration service
  - By creating a REST API
  - By creating a library
  - From a WSDL or XSD file
  - By discovering a service



© Copyright IBM Corporation 2015

Figure 3-2. Quick starts

WM666 / ZM6661.0

### Notes:

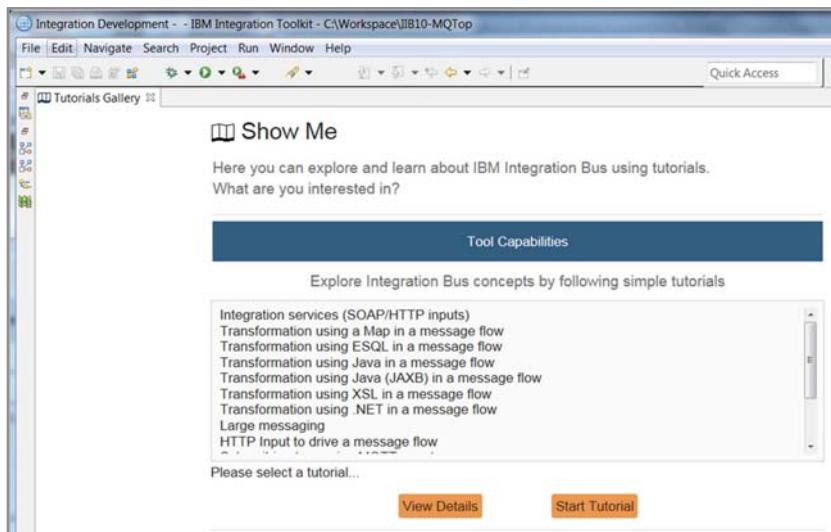
The IBM Integration Toolkit accelerates your development by helping you create a message flow application. You can:

- Use supplied patterns that encapsulate a tested approach to solving a common architecture, design, or deployment task in a particular context.
- Use existing WSDL or XSD files as a basis for message set and message flow development.
- Create an application “from scratch” by creating application or library in which to store your development artifacts. The Integration Toolkit helps you define the application or library and the components you need. From there, you can create a message flow and any other components you need.
- Create a REST API by importing a Swagger document that describes the resources and operations that you want in the REST API.
- Create an integration service, which is a specialized application with a well-defined interface and structure.



## IBM Integration Bus Tutorials Gallery

- Introduce the features that are available in IBM Integration Bus and how to use them
  - End-to-end IBM Integration Bus message flow applications
  - Demonstrate a specific feature of IBM Integration Bus such as file processing, message transformation, and modeling
- Access from the **Welcome** page or by clicking **Help > Tutorials Galley**



© Copyright IBM Corporation 2015

Figure 3-3. IBM Integration Bus Tutorials Gallery

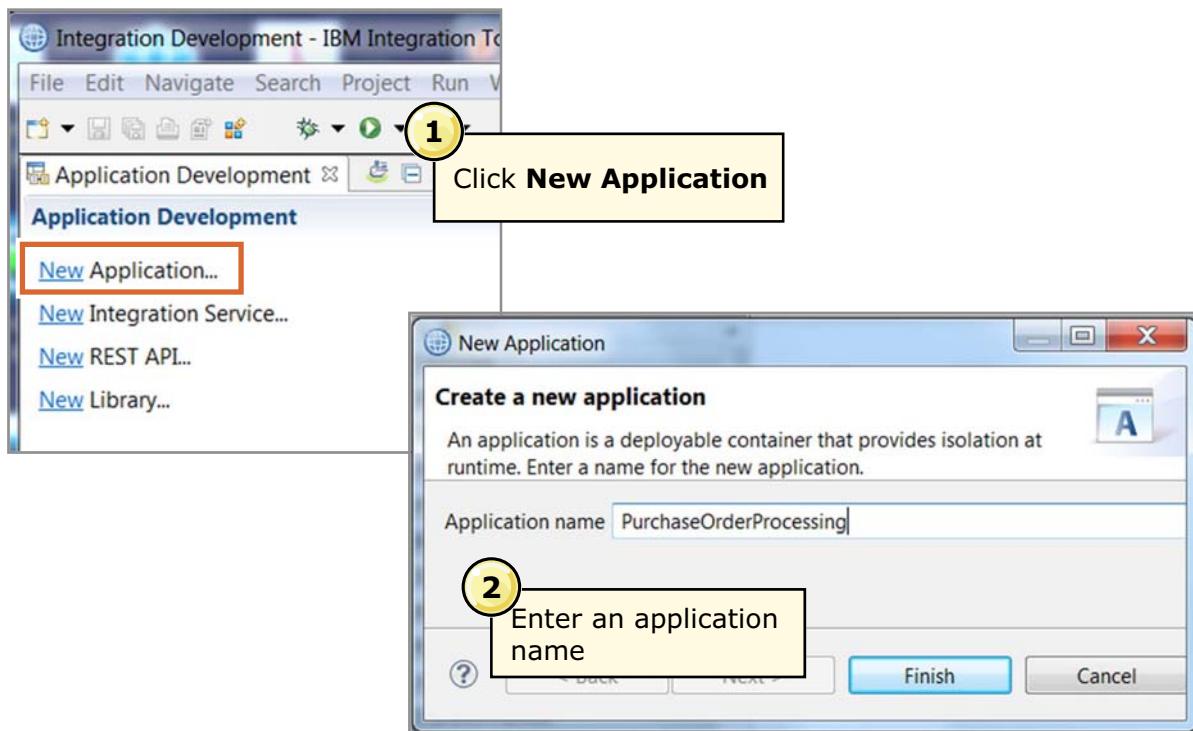
WM666 / ZM6661.0

### Notes:

The IBM Integration Toolkit provides many simple tutorials. Unlike a pattern, which acts as a configurable template that you can customize to your needs, the tutorials provide a complete working example of an IBM Integration Bus solution.



## Creating an application



© Copyright IBM Corporation 2015

Figure 3-4. Creating an application

WM666 / ZM6661.0

### Notes:

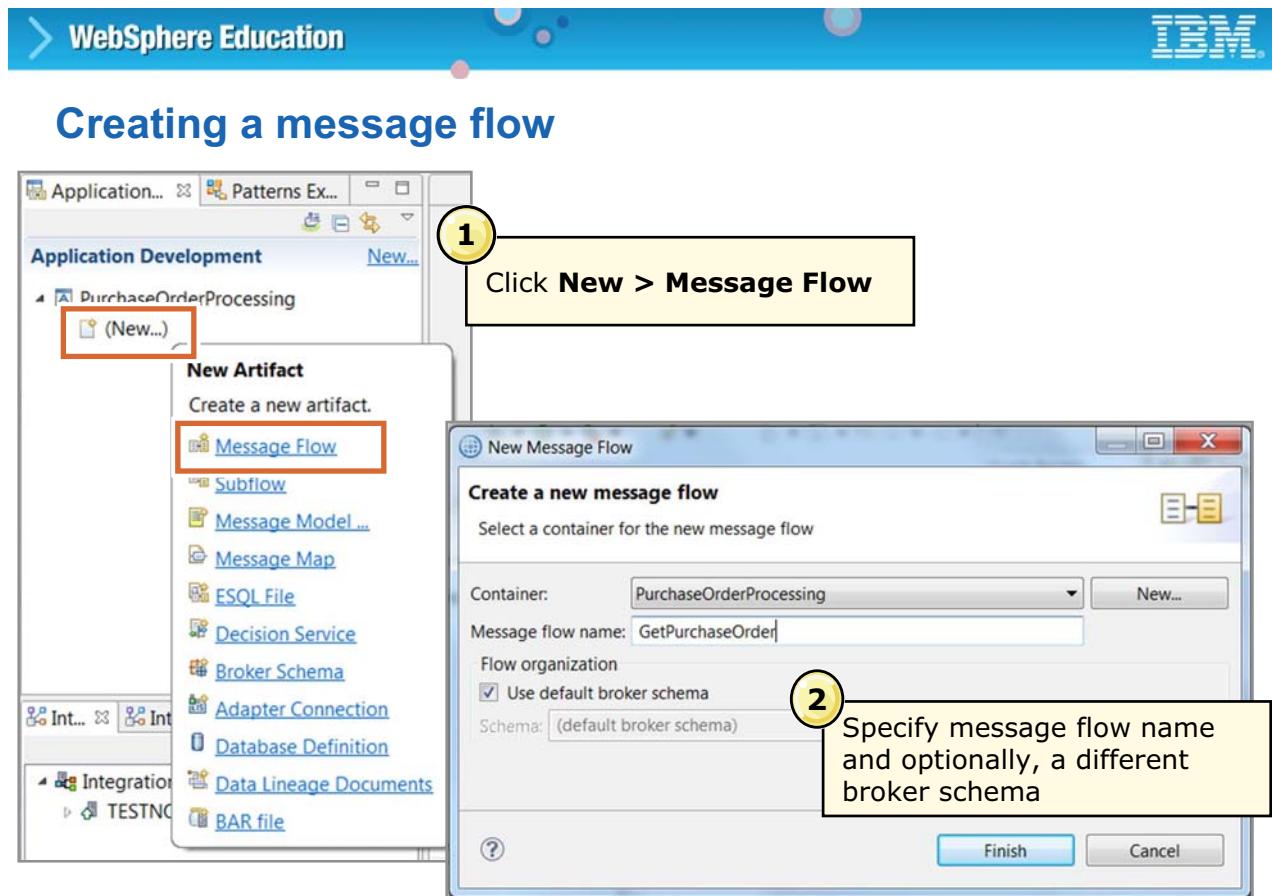
If you cannot find a pattern that matches your solution requirements, you can create an application and then define the message flow and resources.

An application is a container for all the resources that are required to create a solution. An application can contain IBM Integration Bus resources, such as flows, message definitions, and JAR files.

The order in which you create resources is flexible. You can begin by creating an application, then create resources to add to it, or you can begin by creating libraries of resources, then create an application that refers to them.

To create an application in the Integration Toolkit:

1. Click the **New Application** link in the Application Development view.
2. Enter an application name and then click **Finish**.



© Copyright IBM Corporation 2015

Figure 3-5. Creating a message flow

WM666 / ZM6661.0

## Notes:

After you create the application, you can create the message flows that specify how to process messages in the integration node.

To create a message flow in an application:

1. Click the **New > Message Flow** link under the application in the Application Development view.
2. Enter a message flow name.

Optionally, you can specify a new broker schema name that reflects the resources that the schema contains. For example, if you want to use this schema for message flows for retail applications, you specify a schema name of `Retail`.

When you click **Finish**, the message flow is added to the application in the Application Development view and the message flow opens in the Message Flow editor.

The next step is to add the nodes to the message flow.



## Adding nodes to the message flow

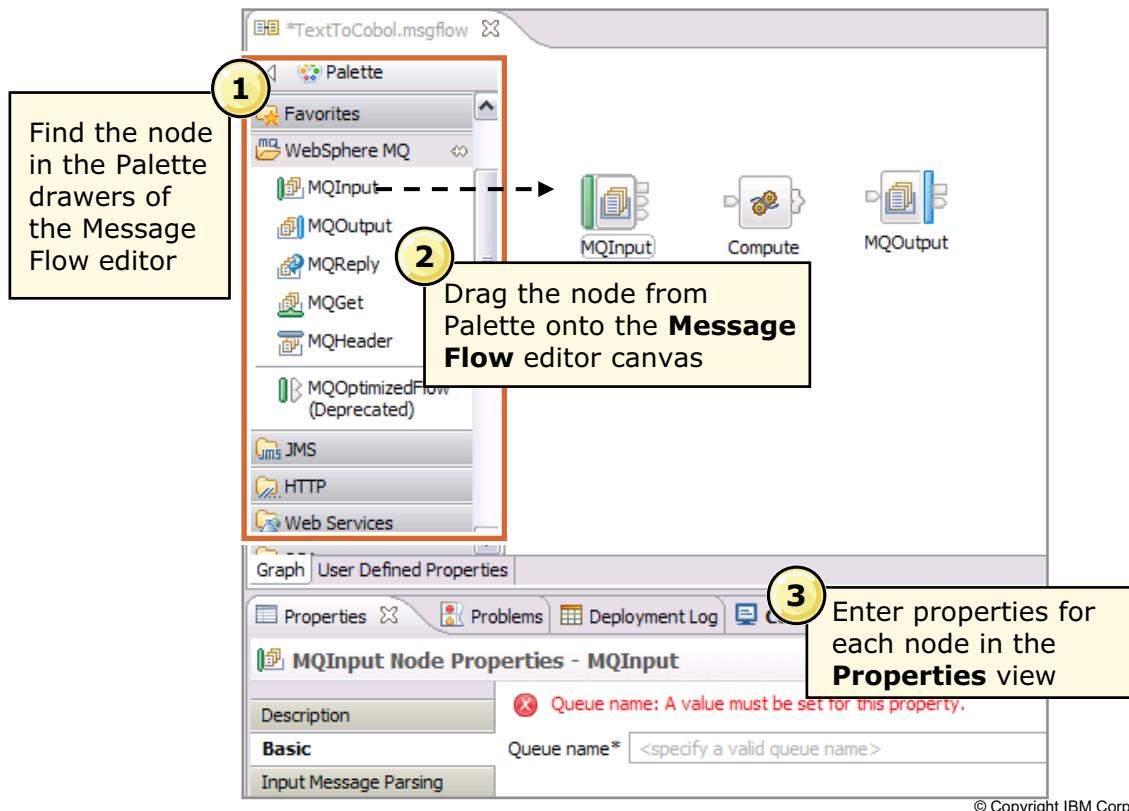


Figure 3-6. Adding nodes to the message flow

WM666 / ZM6661.0

### Notes:

The Message Flow editor Palette categorizes the Integration Bus processing nodes in *drawers* by type and function. For example, the Palette contains a **WebSphere MQ** drawer that contains the nodes that act specifically on IBM MQ messages such as MQInput, MQOutput, and MQGet.

You add nodes to a message flow by dragging them from the Palette to the canvas of the Message Flow editor. You can also click a node in the Palette, position the mouse cursor to the location you want in the canvas, and then click to place the node.

As each node is added to the Message Flow editor canvas, the **Properties** view updates to show the properties that can be defined for the node. Most nodes have multiple properties that are categorized on *tabs* in the **Properties** view. The screen capture in the figure shows three of the property tabs on the MQInput node: **Description**, **Basic**, and **Input Message Parsing**.

Some properties are mandatory and require a value. Mandatory properties are designated with an asterisk (\*). You should always check all the property tabs for a node to ensure that you supplied a value for the mandatory properties.



## Wiring nodes



- Use one of the following methods to create a connection:
  - Right-click the node, click **Create connection**, and then select the source and target nodes and terminals
  - Click the source terminal and then click the target terminal
- Hover the cursor over terminals to identify the terminal name
- Hover the cursor over a wire to identify the wire connection source and target
- Directional lines indicate the direction that a message travels

© Copyright IBM Corporation 2015

Figure 3-7. Wiring nodes

WM666 / ZM6661.0

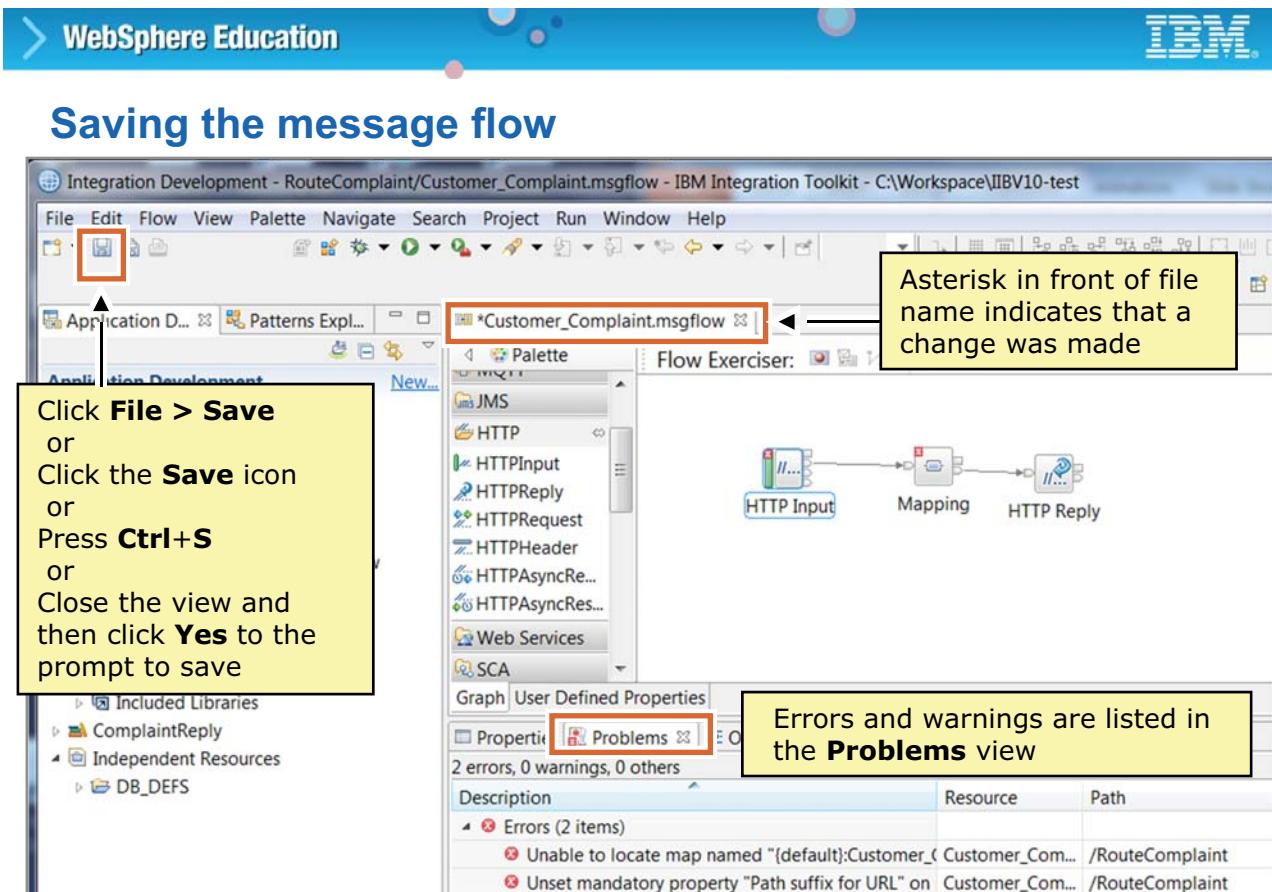
### Notes:

After you place the nodes on the Message Flow editor canvas (or as you are doing so), the next step is to wire the terminals to other nodes to create a flow. A wire creates a path through which a message can flow.

To wire two nodes together, click the source terminal and then click the target terminal. You can also right-click the source node and then click **Create connection** to select the nodes and terminals by name.

Use caution when wiring nodes. If you wire the wrong terminal from a node, unexpected results occur, including incorrect processing and possible loss of messages. If a node propagates a message through a terminal that is unwired, the message is lost without warning from the integration node.

After wiring the nodes, hover the cursor over the new wire to be sure that the correct terminals were selected. You can also use the **Outline** view at the bottom of the Application Development perspective to see all the nodes and their connections.



© Copyright IBM Corporation 2015

Figure 3-8. Saving the message flow

WM666 / ZM6661.0

## Notes:

Integration Bus objects are stored as files on the file system. Any time that you modify a file in the Integration Toolkit ensure that you save the file so that you do not lose your changes.

When you save a message flow file, the Message Flow editor does some basic validation on the message flow file. For example, the Message Flow editor verifies that all mandatory properties have a value. If the message flow references an external resource, such as a map or message model, the Message Flow editor verifies that those references are available. Any error or warnings that are found during the validation process are listed in the **Problems** view.



## Deploying solutions

- After a message flow is developed, the flow and all related components must be moved (deployed) to an integration server so that it can be run
1. Create a BAR file that contains the resources to deploy
  2. Ensure that the integration node and integration server are running
  3. Configure queues and DSNs as needed by the message flow nodes
  4. Deploy any shared libraries that the application uses
  5. Deploy the BAR file to the integration server by using:
    - IBM Integration Toolkit
    - IBM Integration web user interface
    - The `mqsideploy` line command
    - IBM Integration API
  6. Monitor the **Deployment log** for the status

© Copyright IBM Corporation 2015

Figure 3-9. Deploying solutions

WM666 / ZM6661.0

### Notes:

To run a message flow, you must move it to an integration server. This process is called *deployment*.

When you use the Integration Toolkit Flow exerciser to test the flow, the application is deployed automatically. If you are not using the Flow exerciser or need more control over the deployment process, you must create a BAR file and deploy it manually.

The BAR file contains the resources for the application. Resources can include message flows, message models, and ESQL code.

After the BAR file is created, you deploy it to an integration server. During deployment, the BAR file is unpacked and the message flow components are placed into the integration server libraries.

This figure summarizes the steps for deploying Integration Bus solutions. The next pages describe these steps in more detail.

## Creating a BAR file

- Create a BAR file to hold the deployable components by using:
  - IBM Integration Toolkit by clicking **File >New > BAR file**
  - The `mqsicreatebar` or `mqsipackagebar` command on Windows and Linux
- If using `mqsipackagebar`, you must use the `mqsicreatebar` command first to compile any message sets and Java code
- Add a message flow (defined in a `.msgflow` file) or a compiled message flow (`.cmf`)
  - Select **Compile and in-line resources** to add flows as a compiled message flow

© Copyright IBM Corporation 2015

Figure 3-10. Creating a BAR file

WM666 / ZM6661.0

### Notes:

As is the case with many operations in IBM Integration Bus development, creating and managing BAR files can be achieved in several ways. For example, you can create BAR files by using the Integration Toolkit or by using the `mqsicreatebar` or `mqsipackagebar` commands.

The commands can be included in scripts to automate some tasks. For example, you can use the `mqsipackagebar` command to create deployable BAR files on PCs that do not have the IBM Integration Toolkit installed. Although in this course you do not use the commands for creating BAR files, you should know about these commands.

When you add a message flow to a BAR file, you can add the flow as a `.msgflow` file, or as a compiled message flow (defined in a `.cmf` file). You cannot add the same message flow to a BAR file as both a `.cmf` file and a `.msgflow` file.

Deploying all the resources in the compiled form requires that all the subflows are implemented in `.msgflow` files. As a result, all the flow, subflow, and ESQL code is in-lined into the parent `.cmf` file. This method does not allow the same flexibility as the source deployment, but it does provide an unambiguous runtime behavior.



**Important**

If the BAR file contains a mixture of resources that are compiled and resources that are not compiled, you might see unexpected results.

## BAR file contents

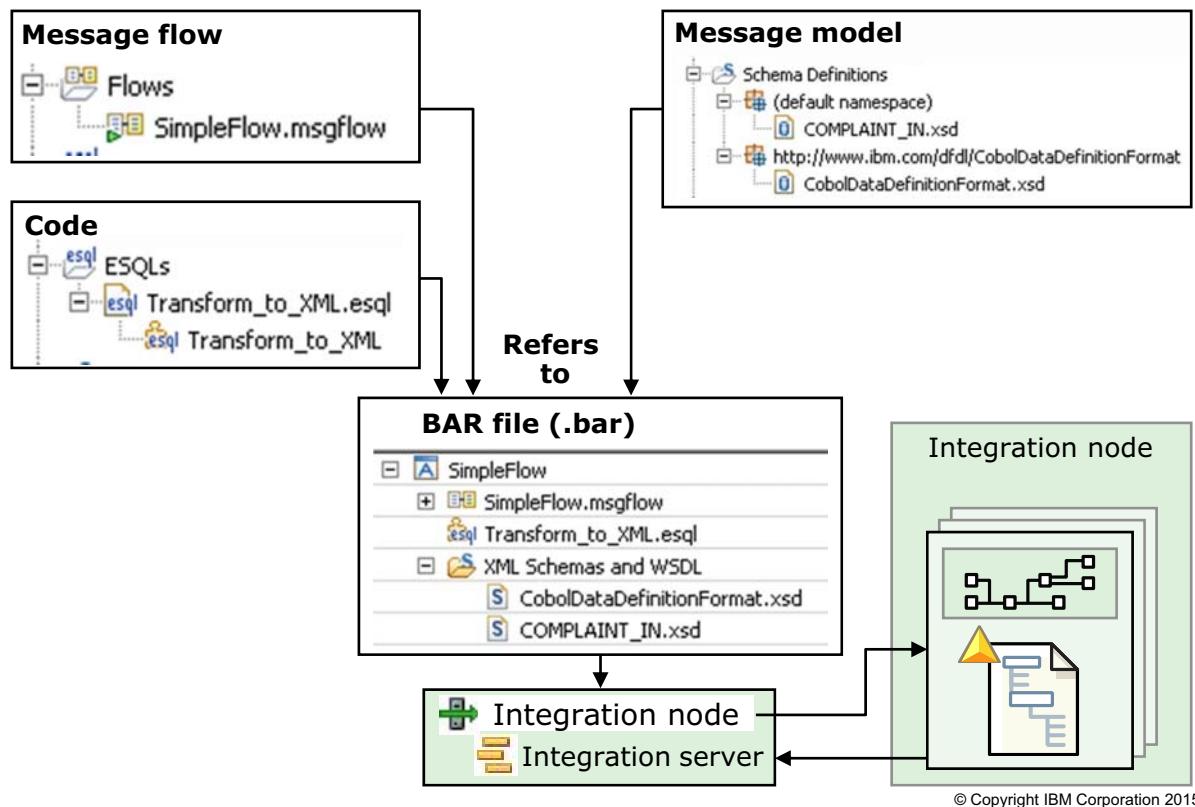


Figure 3-11. BAR file contents

WM666 / ZM6661.0

### Notes:

The BAR file can contain many different files such as:

- A message flow (.msgflow) file for each message flow
- Message model schemas
- XML files (.xml) and stylesheets (.xsl files) for use with the XSLTransform node
- ESQL files for use with the Compute node or Java files for use with JavaCompute node

The BAR file also contains the `broker.xml` file. This file is called the *deployment descriptor*. This file, in XML format, is in the `META-INF` folder of the `.zip` file and can be modified by using a text editor or shell script. The deployment descriptor is described in more detail later in this course.



## Creating the BAR file in the Integration Toolkit

- From the IBM Integration Toolkit:
  - Click **File > New > BAR File**
  - Specify the name of the BAR file
  - Add components to the BAR file by using the editor, or drag components from the Application Development view
- By default, the BAR file is created in the **BarFiles** project
  - To change this value in the IBM Integration Toolkit, click **Window > Preferences**, expand **Integration Development**, and then select **Build BAR**

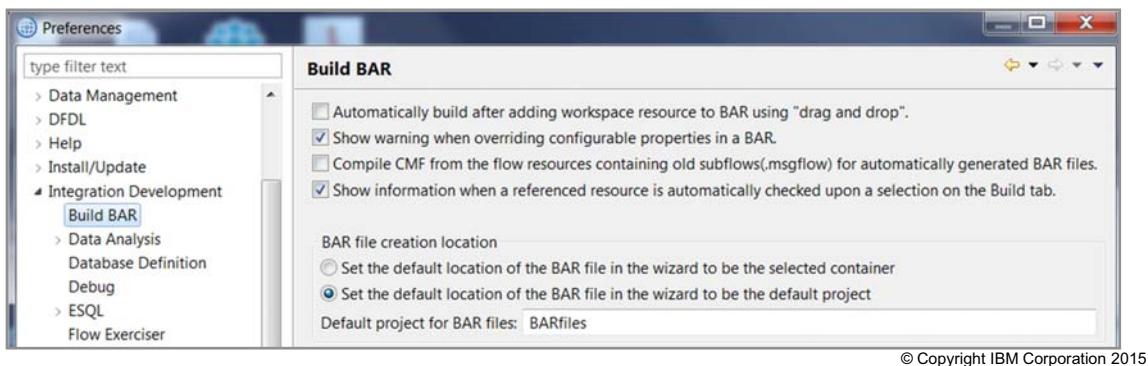


Figure 3-12. Creating the BAR file in the Integration Toolkit

WM666 / ZM6661.0

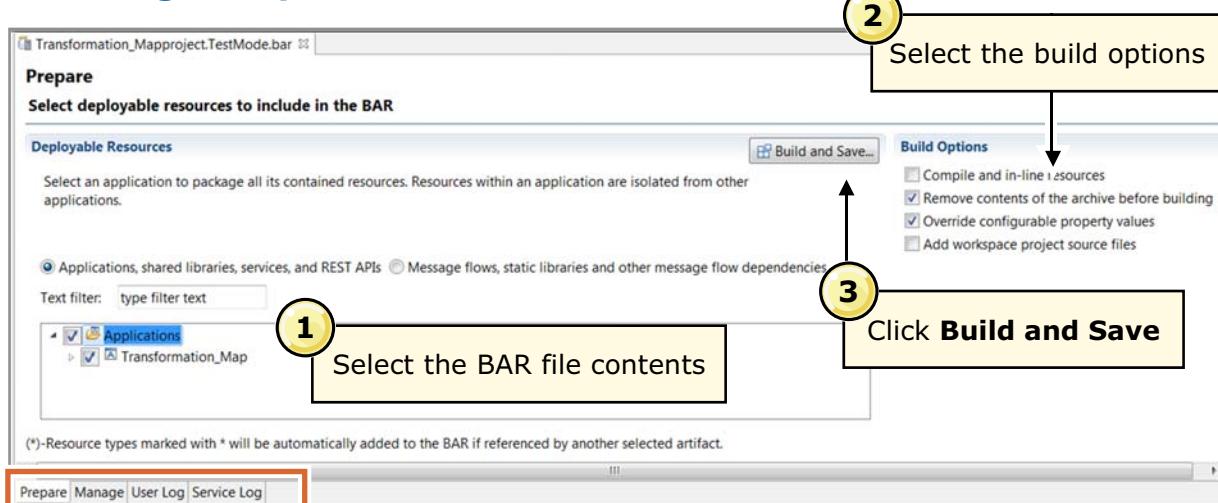
### Notes:

The figure describes one of the ways to create a BAR file in the IBM Integration Toolkit.

By default the BAR file is created in a special **BarFiles** project. You can change the default project for BAR files in the Integration Toolkit **Integration Development > Build BAR** preferences.



## Adding components to the BAR file



- **Prepare** view build options include:
  - **Remove contents of the archive before building** to remove all existing contents of the BAR file before building the new BAR file
  - **Override configurable properties values** to override the values from the message flow
  - **Add workspace project source files** to include source files in the BAR file

© Copyright IBM Corporation 2015

Figure 3-13. Adding components to the BAR file

WM666 / ZM6661.0

### Notes:

As shown in the figure, you populate the BAR file with artifacts from the workspace on the **Prepare** tab by checking the resources that are required and clicking **Build and save**.

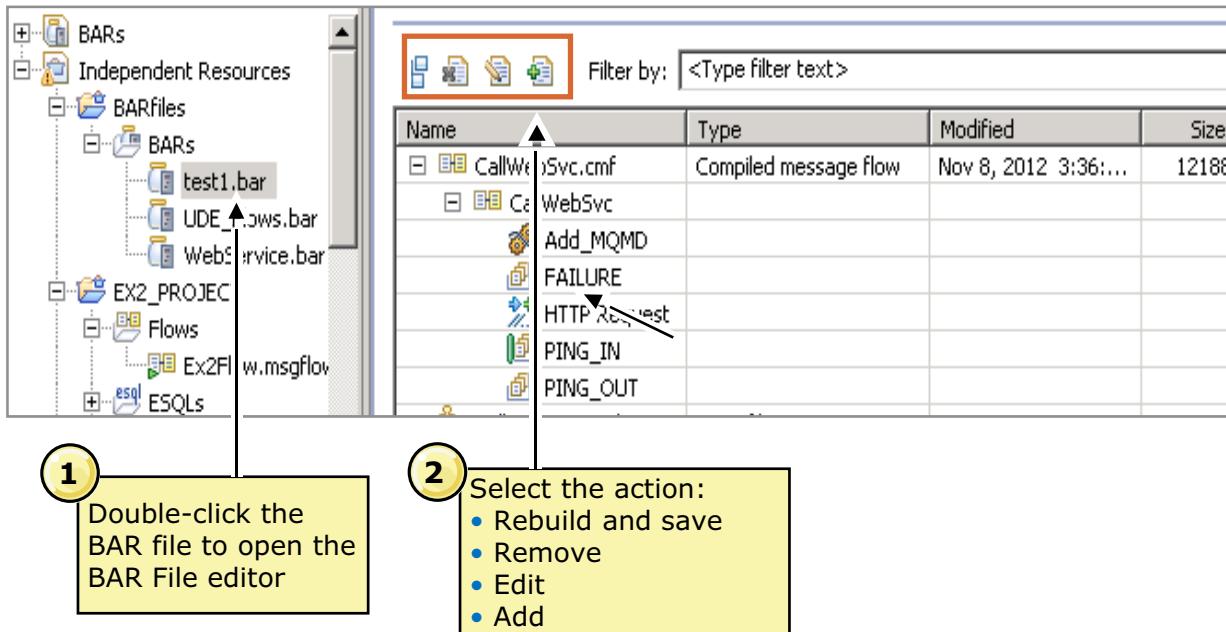
By default, the **Prepare** tab shows all the resources that are available in the workspace. It is possible to limit the list to only those resources that are contained within a particular working set. If you define a working set that contains only the resources that are needed for a particular application, you can select this working set on the BAR file builder. The BAR file builder then shows the relevant resources only.

If you elect to include source files in the BAR file, these resources are stored in a separate folder, which is named `src`. A separate folder clearly distinguishes between source components and the component that the integration node runtime environment uses.

Compile errors prevent components from being successfully deployed. If you receive a message about a failed deployment, be sure to consult the **Problems** view and correct any problems. However, components that display warning messages only are included in the BAR file.



## Modifying a BAR file



© Copyright IBM Corporation 2015

Figure 3-14. Modifying a BAR file

WM666 / ZM6661.0

### Notes:

After the BAR file is created, you can use BAR file editor to add and remove components and change the deployment options. You can also add components to the BAR file in the Application Development view by dragging them into the BAR file.

To modify a BAR file:

1. Double-click the file with the .bar extension to open the BAR File editor.

Go to the **Manage** tab. The Manage tab displays the message flows, message models, and other files that are currently in the BAR file. Use the icons to add, edit, and remove files from the BAR file.

2. Select the action. The action icons on the toolbar in the **Manage** tab are used to:

- Build a BAR file.
- Remove message flows, message definitions, or other items from the BAR file.
- Edit selected message flows, message definitions, or other items in the BAR file.
- Add message flows, message definitions, or other items to the BAR file.



## BAR file message flow properties

**Manage**  
Rebuild, remove, edit, add resources to broker archive and configure their properties

Filter by: <Type filter text>

Name	Type	Modified
DR_Manning.map	MAP file	Jun 20, 2015 46:46 AM
DB.msgflow	Message flow	Jun 20, 2015 46:46 AM
COMPLAINT_FAILURE		
COMPLAINT_IN		
COMPLAINT_OUT		

Command for packaging the BAR contents

Prepare **Manage** Ser Log Service Log

Properties Problems Deployment Log Integration Registries

**DB.msgflow**

**Configure** Configure properties of selected built resource.

Workload Management Details

Consumer Policy Set	
Consumer Policy Set Bindings	
Coordinated Transaction	<input type="checkbox"/>
Monitoring Profile Name	
Provider Policy Set	
Provider Policy Set Bindings	
Security Profile Name	

© Copyright IBM Corporation 2015

**1** Select the message flow on the BAR file editor **Manage** tab

**2** Override the configurable properties in the **Properties** view

Figure 3-15. BAR file message flow properties

WM666 / ZM6661.0

### Notes:

In the **Manage** view of the BAR File editor, you can set variables that give the BAR file more or alternative runtime properties. In this way, the content of the BAR file (that is, message flows and message definitions) can remain unchanged but the attributes can vary from one package to another.

The properties that can be modified without changing the underlying message flow itself are known as *configurable properties*. Configurable properties eliminate the need to modify message flow source as the flow progresses through the lifecycle from development to test to production.

**Manage**  
Rebuild, remove, edit, add resources to broker archive and configure their properties

Filter by: <Type filter text>

Name	Type	Modified
DB.msgflow	Message flow	Jun 20, 2015 6:46 AM
DB		
COMPLAINT_FAILURE		
COMPLAINT_IN		
COMPLAINT_OUT		
Compute Reply		

Command for packaging the BAR contents

Prepare Manage User Log Service Log

Properties Problems Deployment Log Integration Registries

**COMPLAINT\_FAILURE**

**Configure** Configure properties of selected built resource.

Workload Management

Queue manager name	
Queue name	COMPLAINT_FAILURE
Reply-to queue	
Reply-to queue manager	
Security profile	
Validate	Inherit

© Copyright IBM Corporation 2015

Figure 3-16. BAR file node properties

WM666 / ZM6661.0

## Notes:

You can also use the BAR file editor to modify some message processing node properties.

Consider the case of a simple message flow. It uses an input queue with a development name of Q\_IN. On the production system, the corresponding input queue might be named COMPLAINT\_FAILURE. In the **Manage** tab of the BAR File editor, right-clicking the MQInput node name displays the node properties where you can override the associated input queue. In this manner, an administrator can alter the message flow by replacing the input queue that the developer uses (Q\_IN) with the queue name in the production system (COMPLAINT\_FAILURE).

You can also view the properties that can be configured for your message flows by selecting **Configurable properties** from the **Filter by** list.

Be sure to save the BAR file after you change it.



## Message flow packaging and validation

- Message flow validation
  - Ensure that mandatory properties are set
  - Finds associated resources in referenced projects such as subflows, ESQL, and maps
  - Calculates values for promoted properties; promoted from node to subflow to main flow
  - Identified errors and warnings are listed in the **Problems** view
- Cannot deploy from an application, library, or project that contains errors (but can deploy with warnings)

© Copyright IBM Corporation 2015

Figure 3-17. Message flow packaging and validation

WM666 / ZM6661.0

### Notes:

The message flow is validated when you click **Build and Save** in the BAR File editor. Every resource is subject to incremental validation upon saving. Errors or warnings are detailed in the **Problems** view. In the Application Development view, you can also see the error chain up to the application level. Errors in an application prevent deployment.

Double-clicking the error in the task view often takes you to the location of the problem so that it can be fixed. The **Task list** can be filtered, for example, to show items from selected resource only.

The validator uses application properties to find dependent applications and libraries. They also find associated resources such as ESQL modules and maps. The compiler lists its results in the **Details** section of **Add to BAR** operation.

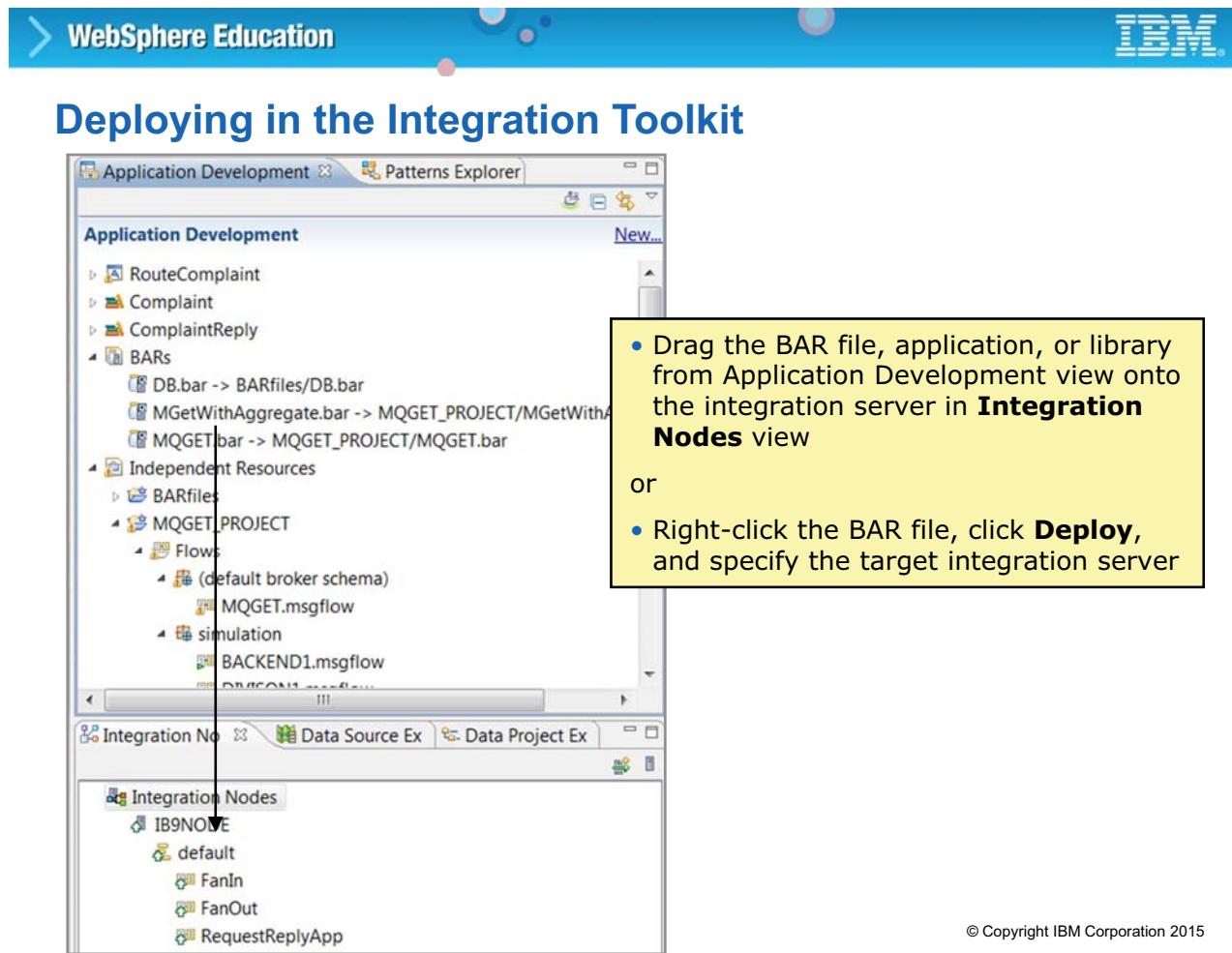


Figure 3-18. Deploying in the Integration Toolkit

WM666 / ZM6661.0

### Notes:

There are many ways to deploy the BAR file to an integration server. One way is to select the BAR file in the Application Development navigator and drag it onto the integration server in the **Integration nodes** view. Alternatively, you can right-click an integration server to select a BAR file to deploy to the selected integration server.

You can also deploy message flow applications directly onto integration servers without having to build BAR files or change perspectives.

Deployment results are displayed synchronously in a **Deployment log**. With the **Deployment Log**, you can quickly verify that message flows are deployed and working as expected.

## Deployment details

When a deployment operation is requested:

1. The integration node ends processing of application messages and stops all message flows
  2. The BAR is deployed, and the required updates are made to the integration server and integration node environment
  3. Message flows are restarted
  4. The **Deployment Log** and **Integration Nodes** view are updated
- 
- Always check the **Deployment Log** after deployment to verify that the expected actions occurred
  - If necessary, adjust timeout values for the integration node
    - Depends on network delays, integration node workload, and system load

© Copyright IBM Corporation 2015

Figure 3-19. Deployment details

WM666 / ZM6661.0

### Notes:

The figure lists the actions that occur when a BAR file is deployed to an integration server.

Be sure to check the **Deployment Log** after every deployment operation to ensure that the deployment was successful. Deployment messages are also shown in the system event log.



## Deployment Log

- Shows the result of deployment actions on an integration node
  - Name of the integration node from which deployment message originated
  - Detail of the deployment message
- Clean Log** clears all log entries from the **Deployment Log** view, but does not delete the log entries from the administration log
- Messages are automatically cleared after 72 hours

Deployment Log		Timestamp
Details		
[IB9NODE_default.bar]	has been deployed to integration server default on integration node IB9NODE	Fri Jun 07 10:57:2
BIP2871I:	The request made by user 'IIBAdmin' to 'deploy' the resource '20130607_1057_23' of type 'BAR'	Fri Jun 07 10:57:4

© Copyright IBM Corporation 2015

Figure 3-20. Deployment Log

WM666 / ZM6661.0

### Notes:

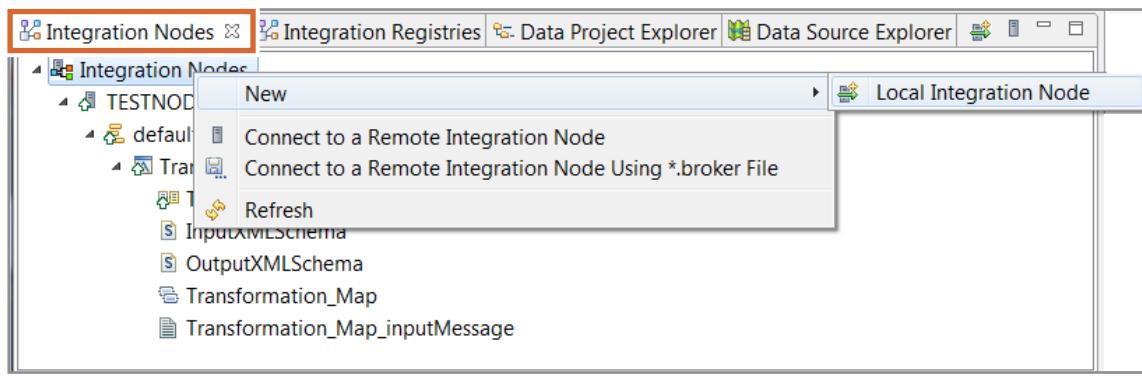
The Deployment Log shows the result of a deployment action on an integration node. The figure shows an example of a Deployment Log.

If you inadvertently close the Deployment Log view, you can reopen it from the toolbar by clicking **Window > Show View > Deployment Log**.



## Managing the integration node in the Integration Toolkit

- Create a local integration node or connect to a remote integration node
- Start and stop an integration node, integration server, or message flow
- Create and delete integration servers
- Start the IBM Integration web user interface
- Deploy applications, libraries, message flows, BAR files
- Verify the status of integration nodes, integration servers, and message flows



© Copyright IBM Corporation 2015

Figure 3-21. Managing the integration node in the Integration Toolkit

WM666 / ZM6661.0

### Notes:

You can do some basic management of integration nodes, integration servers, and message flows in the Integration Toolkit.



## Using the IBM Integration web user interface

The screenshot shows the IBM Integration web user interface. On the left, there is a navigation tree under 'TESTNODE\_watss' for 'Servers/default'. The 'Monitoring' section is expanded, and 'Admin Log' is selected. The main panel is titled 'Admin Log' and shows a table of log messages. The table has columns for 'Message', 'Source', 'Timestamp', and 'Message Detail'. The messages listed are all 'BIP2871I' type, mostly 'Administration Request' or 'Administration Result' from 'SYSTEM|D'. A green callout box on the right lists administrative tasks:

- Monitor the status of integration node, integration servers, and applications
- Create and delete integration servers
- Start and stop applications and message flows
- Start and stop statistics collection
- View the integration node Admin Log
- Deploy BAR files

© Copyright IBM Corporation 2015

Figure 3-22. Using the IBM Integration web user interface

WM666 / ZM6661.0

### Notes:

The IBM Integration web user interface is the primary interface for managing an integration node and its components.

You can complete the following administrative tasks from the IBM Integration web user interface:

- View, create, rename, start, stop, and delete integration servers
- Deploy resources
- View, start, stop, manage, and delete deployed resources
- View, create, edit, and delete configurable services
- Create, retrieve, update, and delete operational policies
- Edit some properties of integration nodes, integration servers, and message flows
- Work with statistics and accounting data for the message flows
- Work with statistics for the resources that are used by the integration servers

The IBM Integration web user interface is enabled and assigned to port 4414 by default. You can change the port when an integration node is created. You can disable the IBM Integration web user interface at any time, if necessary.



## Integration server status example

**default - Integration Server**

Overview    Resource Statistics    Statistics

▼ Quick View

Integration Server Name	default
Run Mode	running
UUID	87671332-01d2-4c13-a488-3217aa5fd106
Running	true
Short Description	
Long Description	
Record Mode	Disabled

▼ Advanced Properties

Injection Mode	Disabled
Process ID	9488
Trace Level	none
Soap Nodes use Embedded Listener	true
Thread Local Proxy Name Managers	false
Console Mode	off
HTTP Nodes use Embedded Listener	false
Inactive User Exit List	
Active User Exit List	

© Copyright IBM Corporation 2015

Figure 3-23. Integration server status example

WM666 / ZM6661.0

### Notes:

You can view the status of the integration node, integration servers, and message flow. This figure shows an example of the integration server status for the integration server that is named default. If you have administrator permissions, you can modify the configurable properties by clicking **Edit**.



## Deploy by using the IBM Integration web user interface

1. Expand the **Servers** section
2. Click the arrow beside the integration server where you want to deploy the BAR file and then click **Deploy**
3. To select a BAR file, click **Browse**
4. Go to the BAR file directory, click the BAR file, and click **Open**
5. (Optional) Override the BAR file properties
6. Click **Deploy**

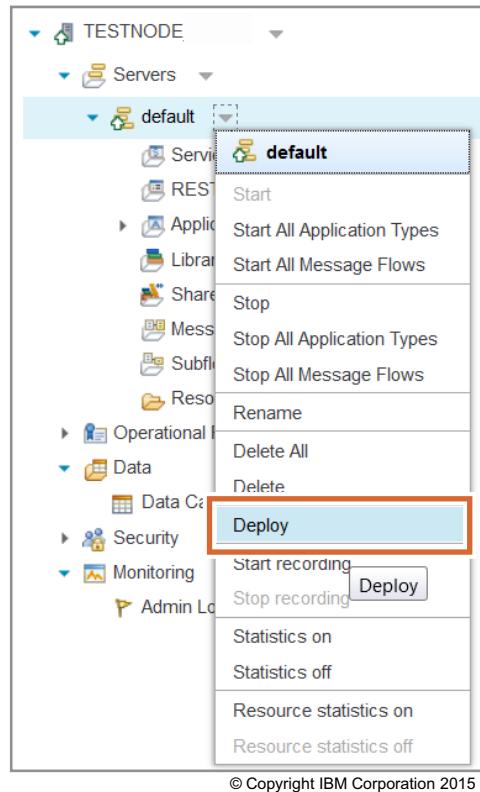


Figure 3-24. Deploy by using the IBM Integration web user interface

WM666 / ZM6661.0

### Notes:

You can deploy existing BAR files to an integration server in the IBM Integration web user interface.



## IBM Integration web user interface BAR file properties

- Shows contents of BAR file (similar to BAR File editor in the Integration Toolkit)
- Overrides** file is a properties file in which each line contains a `property-name=override pair` or `property-value=new-property-value pair`

**Deploy BAR File**  
Select a BAR file to deploy. Optionally, provide an overrides file.

BAR file:	<input type="text" value="MQTopproject.TestMode.bar"/>	<input type="button" value="Browse"/>
Deploy preview:		
Content	Value	
startMode	<unset>	
startInstancesWhenFlowStarts	<unset>	
MQTOPIN	... MQTOPIN	
queueName	<unset>	
connection	<unset>	
destinationQueueManagerName	<unset>	
queueManagerHostname	<unset>	
listenerPortNumber	<unset>	

© Copyright IBM Corporation 2015

Figure 3-25. IBM Integration web user interface BAR file properties

WM666 / ZM6661.0

### Notes:

You can override the properties in the BAR file in the IBM Integration web user interface by using an overrides file. Any property values that are specified in the overrides file are displayed in the dialog box, in place of the original property values. If you want to restore the original property values from the BAR file, click the down arrow next to **Overrides** and then click **Clear overrides file**.

For more information about the format of an overrides file, see the `mqsiapplybaroverride` command in the IBM Knowledge Center.



## Unit summary

Having completed this unit, you should be able to:

- Create a message flow application
- Add nodes to a message flow
- Package and deploy message flow applications and resources
- Use the IBM Integration web user interface to monitor the integration node, integration server, and message flow

© Copyright IBM Corporation 2015

---

Figure 3-26. Unit summary

WM666 / ZM6661.0

### Notes:



## Checkpoint questions

1. True or False: You can override all node properties in the BAR file.
2. Which of the following options is the correct method for deploying an Integration Bus application that references a shared library?
  - a. Deploy the application BAR file and then deploy the shared library
  - b. Deploy the shared library and then deploy the application BAR file
  - c. Include shared library in the BAR file with the application and then deploy the BAR file

© Copyright IBM Corporation 2015

Figure 3-27. Checkpoint questions

WM666 / ZM6661.0

### Notes:

Write your answers here:

1.

2.



## Checkpoint answers

1. True or False: You can override all node properties in the BAR file.  
**Answer:** **False.** You can override some node properties in the BAR file.
  
2. Which of the following options is the correct method for deploying an Integration Bus application that references a shared library?
  - a. Deploy the application BAR file and then deploy the shared library
  - b. Deploy the shared library and then deploy the application BAR file
  - c. Include shared library in the BAR file with the application and then deploy the BAR file

**Answer:** b.

© Copyright IBM Corporation 2015

Figure 3-28. Checkpoint answers

WM666 / ZM6661.0

### Notes:

## Exercise 2



Creating a message flow application

© Copyright IBM Corporation 2015

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

10.1

Figure 3-29. Exercise 2

WM666 / ZM6661.0

### Notes:



## Exercise objectives

After completing this exercise, you should be able to:

- Create a message flow application
- Use the IBM Integration Flow exerciser to test the message flow application
- Use the IBM Integration web user interface to check the status of the integration node, integration server, and message flow application at run time

© Copyright IBM Corporation 2015

---

Figure 3-30. Exercise objectives

WM666 / ZM6661.0

### Notes:

See the *Student Exercise Guide* for detailed instructions.

# Unit 4. Connecting to IBM MQ

## What this unit is about

In this unit, you learn how to process IBM MQ messages in a message flow. You also learn about the connection and topology options for IBM MQ.

## What you should be able to do

After completing this unit, you should be able to:

- Describe the IBM MQ connection options
- Examine the properties of the IBM MQ nodes
- Predict the location of the message if a runtime error is encountered during message flow processing
- Attach an MQEndpoint policy to one or more IBM MQ nodes in a message flow to control connection details at run time

## How you will check your progress

- Checkpoint questions
- Lab exercise

## References

IBM Knowledge Center for IBM Integration Bus

IBM training course WM207, *IBM MQ V8 System Administration*

## Unit objectives

After completing this unit, you should be able to:

- Describe the IBM MQ connection options
- Examine the properties of the IBM MQ nodes
- Predict the location of the message if a runtime error is encountered during message flow processing
- Attach an MQEndpoint policy to one or more IBM MQ nodes in a message flow to control connection details at run time

© Copyright IBM Corporation 2015

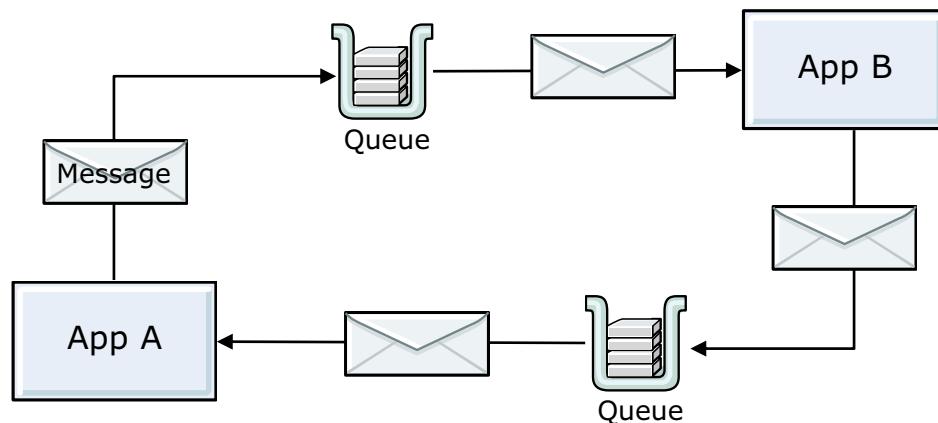
Figure 4-1. Unit objectives

WM666 / ZM6661.0

### Notes:

## IBM MQ functional overview

- Common application programming interface
- Assured message delivery
- Time-independent processing
- Application parallelism
- Faster application development



© Copyright IBM Corporation 2015

Figure 4-2. IBM MQ functional overview

WM666 / ZM6661.0

### Notes:

IBM MQ uses messages and queues to provide program-to-program communication. The communicating applications can be on the same system, or distributed across a network of IBM and non-IBM systems.

The major benefits of IBM MQ are:

- There is a common application programming interface, the Message Queue Interface (MQI) that is consistent across all the supported operating systems.
- IBM MQ can transfer data with assured delivery; messages do not get lost, even when a system failure occurs. There is no duplicate delivery.
- Communicating applications do not have to be active at the same time.
- An application is divided into discrete functional modules that communicate with each other with messages. In this way, the modules can run on different systems, be scheduled at different times, or they can act in parallel.
- Application development is faster because the developer is shielded from the complexities of the network.

## Messages

- Contain the application data or payload
- Are placed on queues, which allow programs to run independently of each other, at different speeds and times, in different locations, and without a logical connection between them
- Contain IBM MQ message descriptor (MQMD) with control information
- Can contain optional message properties that the application creates



Message descriptor (MQMD)	Message properties (Optional)	Application data
---------------------------	-------------------------------	------------------

© Copyright IBM Corporation 2015

Figure 4-3. Messages

WM666 / ZM6661.0

### Notes:

A message is any information that one application needs to communicate to another application. A message can convey a request for a service, or it can be a reply to a request. It might also report on the progress of another message; to confirm its arrival or report on an error, for example. A message can also carry information for which no reply is expected.

A message consists of two parts:

- Message descriptor
- Application data

The message descriptor contains information about the message. The sending application supplies both the message descriptor and the application data when it puts a message on a queue. Both the message descriptor and the application data are returned to the application that gets the message from the queue. The application that puts the messages on the queue sets some of the fields in the message descriptor; the queue manager sets others on behalf of the application.

## Queues

- A defined end-point destination for messages
  - *Local queue* is owned by the queue manager to which the program is connected
  - *Remote queue* is owned by a different queue manager to the one to which the program is connected



Only queues that are defined as local queues (QLOCAL) hold messages

© Copyright IBM Corporation 2015

Figure 4-4. Queues

WM666 / ZM6661.0

### Notes:

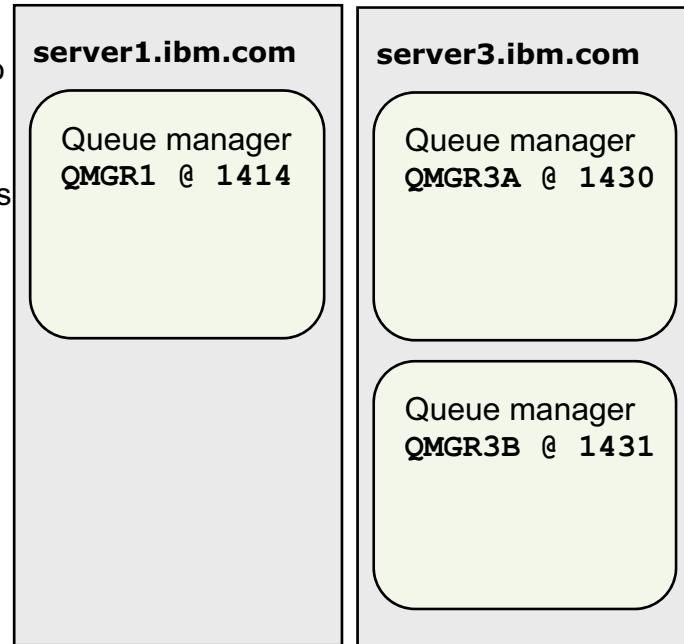
A *queue* is a place to store messages until they can be processed. This figure describes the type of queues the IBM MQ supports.

The time that a message waits to be retrieved and processed can be short. Alternatively, it can be a long time if it must wait for the receiving application to start. Either way, the ability to store a message safely is an important characteristic of a queue.



## Queue manager

- Provides messaging services to applications
- Ensures that messages are sent to the correct queue or are routed to another queue manager
- Hosts the queues and the channels that connect queue managers
- Listeners accept network requests from other queue managers, or client applications, and start associated channels
  - Queue managers that share a server must use different TCP/IP listener ports



© Copyright IBM Corporation 2015

Figure 4-5. Queue manager

WM666 / ZM6661.0

### Notes:

Queue managers are the main components in an IBM MQ messaging network. The queue managers host the other objects in the network, such as the queues and the channels that connect the queue managers together.

In a distributed environment, each queue manager is assigned a unique TCP/IP port.



The screenshot displays the IBM WebSphere MQ Explorer interface. On the left, a tree view titled "IBM WebSphere MQ" shows "Queue Managers" expanded, with "Canberra1" selected. Under "Canberra1", "Queues" is highlighted and expanded, showing sub-folders like "Topics", "Subscriptions", "Advanced", and "JMS Administered Objects". Other nodes include "Canberra2", "Queue Manager Clusters", and "file:/C:/Documents and Settings/Admin". The main pane, titled "Content", is titled "Queues" and contains a table with 15 rows of queue information. The table columns are "Queue name", "Queue type", "Definition type", "Open input count", and "Open output count". The table shows various system and administrative queues. To the right of the table, a bulleted list describes the tool's features:

- Graphical tool for configuring and controlling IBM MQ from a single console
- Configure all IBM MQ objects and resources, including JMS, and publish/subscribe
- Available on Windows and Linux (x86) only

© Copyright IBM Corporation 2015

Figure 4-6. IBM MQ Explorer

WM666 / ZM6661.0

## Notes:

IBM MQ Explorer is the graphical user interface for administering and monitoring IBM MQ objects, whether your local computer or a remote system hosts them.

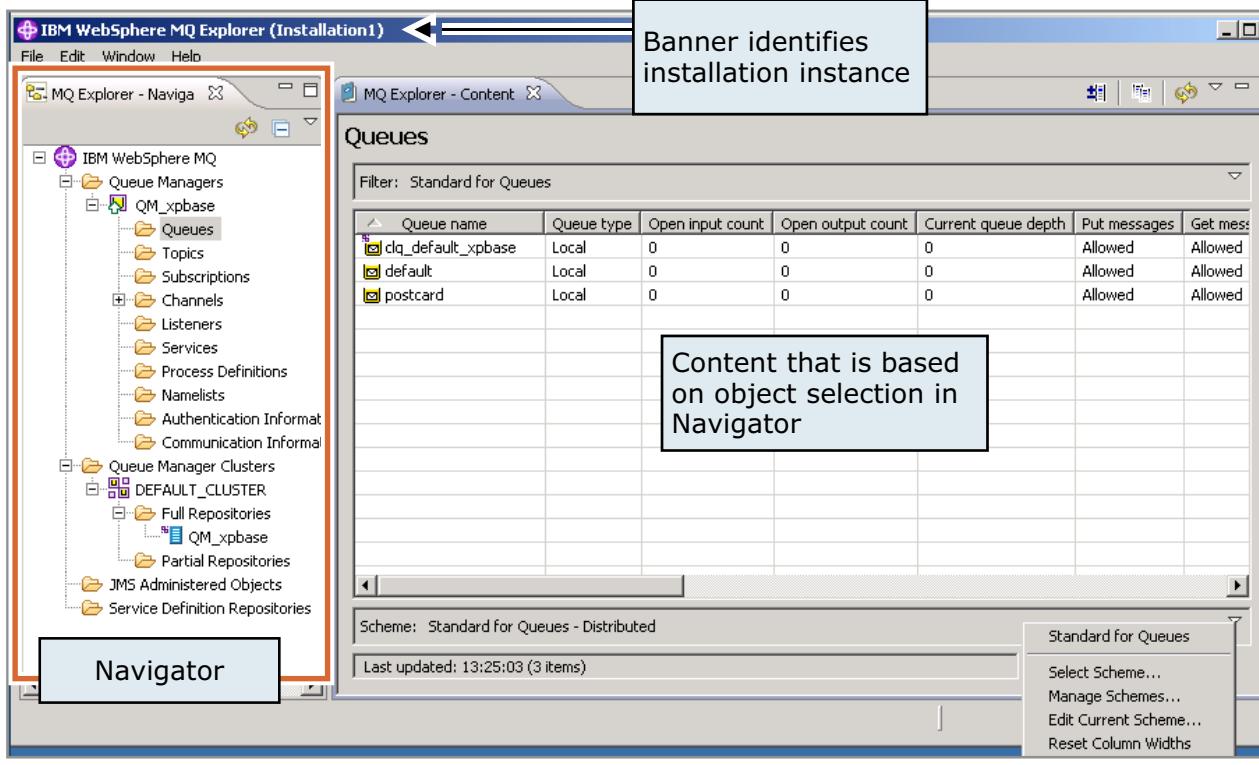
IBM MQ Explorer also supports:

- Advanced filtering
- Management of application connections
- Grouping of queue managers to simplify management
- Publish/subscribe administration

IBM MQ Explorer is built as a Rich Client Platform (RCP) application on Eclipse. It runs on Windows and Linux (x86) operating systems but can be used to remotely manage IBM MQ objects on other operating systems.



## IBM MQ Explorer default views



© Copyright IBM Corporation 2015

Figure 4-7. IBM MQ Explorer default views

WM666 / ZM6661.0

### Notes:

The figure shows the general layout of the IBM MQ Explorer user interface.

The **Navigator** view on the left contains a list of all the objects that IBM MQ Explorer manages. Status icons next to some of the objects, such as queue managers, indicate the status of the object.

The **Content** views contain more information about the object that is selected in the Navigator. For example, if the **Queues** folder under a queue manager is selected in the Navigator view, the content view contains information about the queues on that queue manager.

By using the **Scheme** icon (bottom portion of the **Queues** content view), you can edit the current scheme and select the information to display in the queue property columns. You can also change the order of the columns.



## IBM MQ connectivity options with IBM Integration Bus

- Message flow applications can access existing IBM MQ networks to get and put messages
  - MQInput, MQOutput, MQGet, and MQReply nodes can connect to local or remote queue managers
  - MQInput nodes can receive messages from multiple queues on multiple queue managers
- Globally coordinated transaction control requires that MQInput and MQOutput nodes use same local queue manager
- Policies and configurable services define links to queue managers

© Copyright IBM Corporation 2015

Figure 4-8. IBM MQ connectivity options with IBM Integration Bus

WM666 / ZM6661.0

### Notes:

IBM Integration Bus contains processing nodes that can access existing IBM MQ networks to get and put messages.

You can set a property in an IBM MQ input node and output node to globally coordinate message flow transactions with a transaction manager to ensure the data integrity of transactions. For globally coordinated transactions on distributed systems, you must specify a local queue manager on the integration node. This queue manager is the global transaction manager, and no other IBM MQ resources can be used in the message flow. On z/OS, all queue managers are globally coordinated.

You can specify a connection from an IBM MQ node to a specific local or remote queue manager by using connection properties on the node, including the destination queue manager name, host name, port, and channel. Alternatively, you can use an MQEndpoint policy to control the values of IBM MQ node connection properties at run time, or to specify an IBM MQ broker for event publication.



## Advantages of messaging with IBM MQ

- IBM MQ clients enable an application to connect remotely or locally to an IBM MQ queue manager
- Publish/subscribe support increases messaging capability from point-to-point messaging to a less coupled style of messaging
- IBM MQ clusters allow multiple instances of the same service to be hosted through multiple queue managers to enable load-balancing, and failover and simplify administration
- Secure Sockets Layer (SSL) protocol can be used to secure communication between queue managers or IBM MQ client
- IBM MQ supports a wide range of hardware and operating systems
- IBM Integration Bus license includes entitlement to IBM MQ

© Copyright IBM Corporation 2015

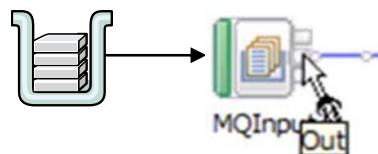
Figure 4-9. Advantages of messaging with IBM MQ

WM666 / ZM6661.0

### Notes:

Listed in the figure are some of the IBM MQ features that make communication and configuration easier.

## MQInput node



- Receives messages from a specified queue through a local or client connection to a queue manager
- Polls a queue for incoming messages that meet search criteria
- Default values for parser selection and publish/subscribe topics
- Establishes transaction mode for entire message flow
- Starts a separate thread
- Multiple MQInput nodes in a message flow identify alternative input sources
- One message is received per thread
- Three output terminals: Failure, Out, Catch

**Note:** On z/OS, only local connections to queue managers are supported

© Copyright IBM Corporation 2015

Figure 4-10. MQInput node

WM666 / ZM6661.0

### Notes:

The MQInput node retrieves a message from a specified IBM MQ message queue. The node gets the first eligible message according to match options that are specified on the node, or in overrides. At run time, the node does an “MQGet with wait” operation on the specified queue. When the message is written to the queue, the MQInput node gets the message, parses it from a bit stream into a logical message tree, and then propagates it to the next node.

The MQInput does other functions as well, such as setting the transactional properties for the message flow. It can also participate in error handling if a runtime exception occurs during processing anywhere in the message flow.

The MQInput node routes each message that it retrieves successfully to the **Out** terminal. The other terminals are used if there is a failure. Message flow behavior in the case of a failure is described later in this unit.

WebSphere Education

## MQOutput node



- Puts a message to the local or remote IBM MQ queue under transaction of message flow, or an explicit commit
- Destination mode
  - Queue name sends message to queue that is identified in **Queue name** property
  - Reply-to queue or MQReply node puts a message to **ReplyToQueue** that is identified in the MQMD
  - Destination List for dynamic routing; queue names are set by using ESQL in other nodes
- Not necessarily the endpoint of the message flow
  - OUT terminal to which the message is routed if it is successfully put to the output queue
  - Force a sequence of written messages

**Note:** On z/OS, only local connections to queue managers are supported

© Copyright IBM Corporation 2015

Figure 4-11. MQOutput node

WM666 / ZM6661.0

### Notes:

The MQOutput node puts messages to a local or remote queue.

You can configure the **Destination Mode** property of the MQOutput node to put a message to a specific IBM MQ queue or to the destinations identified in the local environment that is associated with the message.

- If you select **Queue Name** (the default), the message is sent to the queue that is named in the **Queue Name** property. If you select this option, you must identify a queue manager and specify a value for the **Queue Name** property.
- If you select **Reply To Queue**, the message is sent to the queue that is identified in the **ReplyToQ** field in the MQMD.
- If you select **Destination List**, the message is sent to the list of queues that are named in the local environment that is associated with the message.

The MQOutput node has an **Out** terminal that can be used to route the message if it was successfully put to the output queue.



## MQReply node

- Send a response to the originator of the input message
  - Puts the output message to the queue that the MQMD **ReplyToQ** field identifies in the message header
- On distributed systems, you can configure the MQReply node to put a message to an IBM MQ queue on a local or remote queue manager
- **Transaction Mode** property defines whether the message is written under sync point

© Copyright IBM Corporation 2015

Figure 4-12. MQReply node

WM666 / ZM6661.0

### Notes:

The MQReply node sends a response to the queue that the **ReplyToQ** field of the input message header identifies.

On distributed systems, you can configure the MQReply node to put a message to a queue on any local or remote queue manager. You can either specify the queue manager explicitly by setting the **MQ Connection** properties on the MQOutput node, or specify an MQEndpoint policy on the **Policy** tab.

On z/OS, only local connections to queue managers are supported. You must have a queue manager that is specified for the integration node, but you can also connect to other local queue managers on an MQReply node by using server bindings for the connection.

Similar to the MQInput node, the Transaction Mode property and message persistence determine whether the message is written under sync point control.



## MQGet node

- Allows IBM MQ messages to be retrieved in the middle of a flow
  - Removes unnecessary flows
  - Saves extra parsing and sync points, and failure modes
- IBM MQ is not a database
  - Optimized for a one-time write and a destructive read
  - A more natural fit for many scenarios such as request and reply, and SOAP over IBM MQ
- Full range of MQGET options possible through node configuration for both static and input message defined behavior

© Copyright IBM Corporation 2015

Figure 4-13. MQGet node

WM666 / ZM6661.0

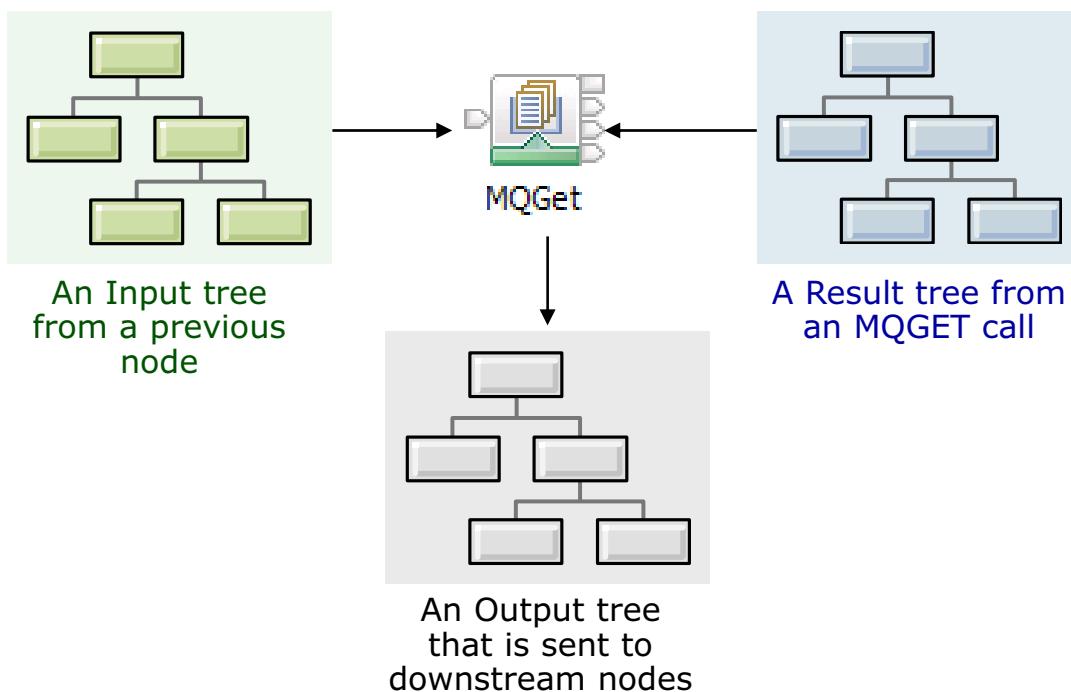
### Notes:

An MQGet node can be used anywhere within a message flow, unlike an MQInput node, which can be used only as the first node in a message flow.

When using a synchronous request/reply pattern, the request message is sent by using an MQOutput node, and the reply would then be received in the same message flow with an MQGet node.

A synchronous flow might not be appropriate in cases where the request/reply processing might be long-running, since processing of other incoming messages are not initiated until the current request is completed. In these cases, the message flow would be better designed asynchronously, with separate request and reply flows. Any required correlation context must be saved to a storage medium in the request flow and restored during the reply flow. A possible solution is to use a queue to store the required correlation context and use an MQGet node in the reply flow to retrieve it. This solution would prevent, for example, a longer running update request from blocking fast-running inquiry requests.

## MQGet node processing outline



© Copyright IBM Corporation 2015

Figure 4-14. MQGet node processing outline

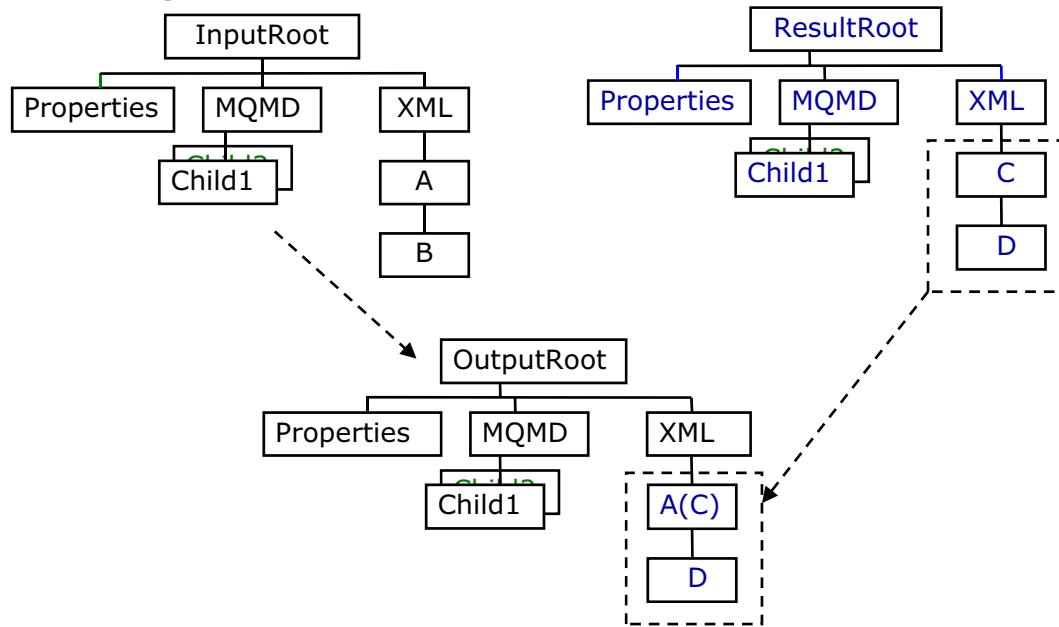
WM666 / ZM6661.0

### Notes:

The output message tree from an MQGet node is constructed by combining the input tree with the result tree from the MQGET call.

You can set the properties of the MQGet node to control the way that messages are received. For example, you can indicate to process a message under transaction control. Or, you can request to convert the data on receipt of every input message when the result tree is being created.

## MQGet example



### MQGet node properties

```

outputDataLocation="OutputRoot.XML.A"
resultDataLocation="ResultRoot.XML.C"

```

© Copyright IBM Corporation 2015

Figure 4-15. MQGet example

WM666 / ZM6661.0

### Notes:

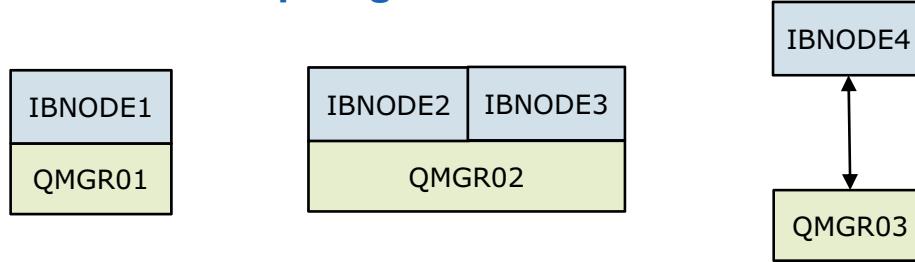
This figure shows a sample MQGet node.

The MQGet node properties and the two Root trees that are going into the MQGet node determine the resulting tree. The first Root tree is the InputRoot from the previous node in the flow. The second Root tree is the ResultRoot from the MQGET of the new message.

The OutputRoot tree that is propagated to the next node would be as shown in the figure with element C from ResultRoot replacing the InputRoot element A and its siblings.

The figure also shows the MQGet node properties.

## Flexible IBM MQ topologies



- Flexible topology options for IBM MQ access for simplicity, scalability, availability, and migration
  - Single integration node can be connected to a single queue manager
  - Multiple integration nodes can be connected to a single queue manager
  - Queue manager can be local or remote from integration node
- If IBM MQ is installed, IBM Integration Bus detects IBM MQ and configures it as required
- IBM MQ policies identify run time resources

**Note:** On z/OS, only local connections to queue managers are supported

© Copyright IBM Corporation 2015

Figure 4-16. Flexible IBM MQ topologies

WM666 / ZM6661.0

### Notes:

On distributed systems, IBM Integration Bus supports local and remote queue managers.

On distributed systems, support for IBM MQ is extended. You can configure local or client connections to IBM MQ so that the integration nodes can get messages from or put messages to queues on any local or remote queue manager. On z/OS, you can have IBM MQ message flow nodes connect to different local queue managers, not just the queue manager that is specified on the integration node.

If the IBM MQ queue manager is running on the same computer as the integration node, you can specify a local connection to the queue manager. Alternatively, if the queue manager is hosted on a separate computer from IBM Integration Bus, you can configure a client connection from the integration node so that it can access the messages on the remote queue manager.

You can also create message flows that contain multiple MQInput and MQOutput nodes, each of which can access different queue managers as specified in the node. Using this feature, you can adapt the message flows to your existing IBM MQ topologies.

## IBM Integration Bus features that require IBM MQ

- Some IBM Integration Bus features and implementations require IBM MQ server on the same system as the integration node, and that you specify a queue manager on the integration node:
  - z/OS implementations
  - Record and replay
  - Global transaction management
  - Event-driven processing nodes that are used for aggregation and timeout flows, message collections, and message sequences
  - IBM MQ File Transfer Edition nodes
  - IBM Sterling Connect: Direct nodes
  - IBM Business Process Manager Advanced nodes that use IBM MQ bindings
  - SAP nodes with transactional processing
  - Integration nodes with HTTP listeners
  - HTTP proxy servlet
  - High availability configurations
  - Using accounting and statistics by using the web user interface and IBM MQ

© Copyright IBM Corporation 2015

Figure 4-17. IBM Integration Bus features that require IBM MQ

WM666 / ZM6661.0

### Notes:

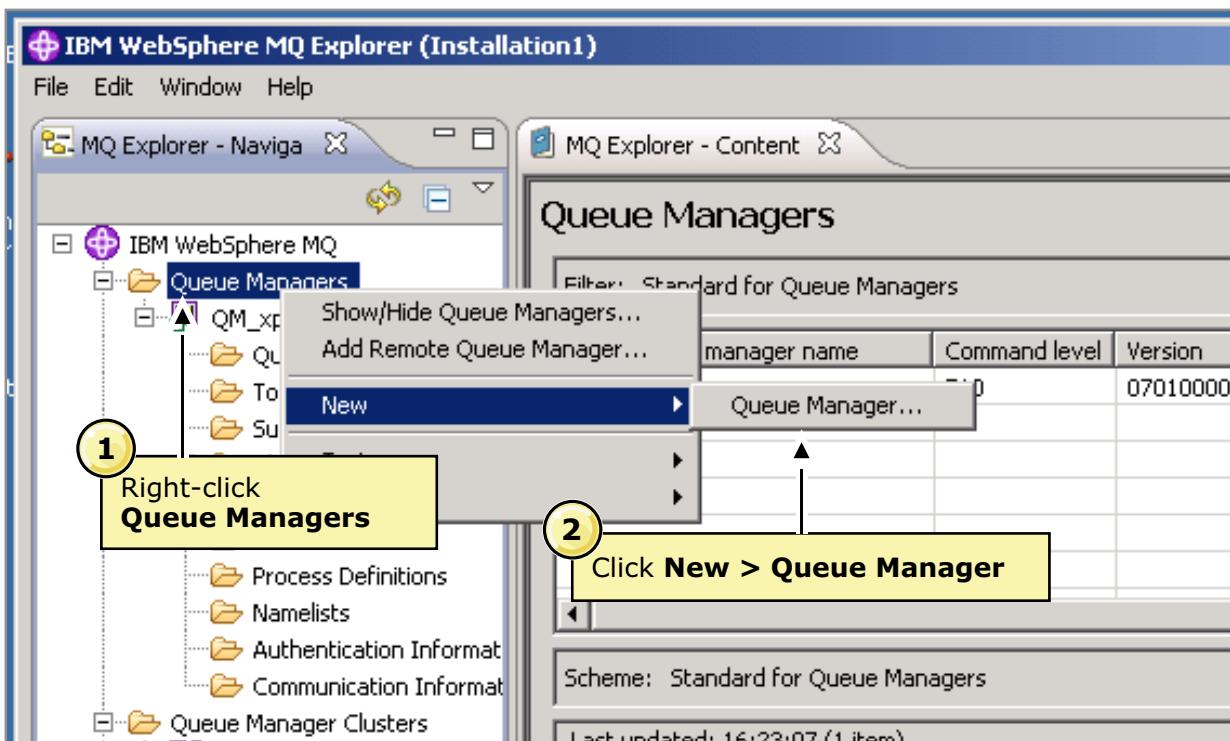
In IBM Integration Bus V10, IBM MQ is no longer a prerequisite for using IBM Integration Bus on distributed operating systems.

Some IBM Integration Bus features require access to an IBM MQ queue manager. Also, some IBM Integration Bus features require access to system queues on a queue manager that is specified on the integration node, for the storage and retrieval of state information.

The integration node listener requires access to IBM MQ Server, so you must install it if you want to use an integration node listener to manage HTTP messages in your HTTP or SOAP flows.

However, if you use HTTP nodes or SOAP nodes with the integration server embedded listener, they do not require access to IBM MQ.

## Creating a local queue manager with IBM MQ Explorer (1 of 5)



© Copyright IBM Corporation 2015

Figure 4-18. Creating a local queue manager with IBM MQ Explorer (1 of 5)

WM666 / ZM6661.0

### Notes:

A queue manager can be created from a command or by using IBM MQ Explorer.

IBM MQ Explorer automatically shows all local queue managers regardless of the method that is used to create them.

The figure shows the first two steps for adding a queue manager to the server by using the IBM MQ Explorer.

1. Right-click **Queue Managers** in the **MQ Explorer - Navigator** view.
2. Click **New > Queue Manager**.



## Creating a local queue manager with IBM MQ Explorer (2 of 5)

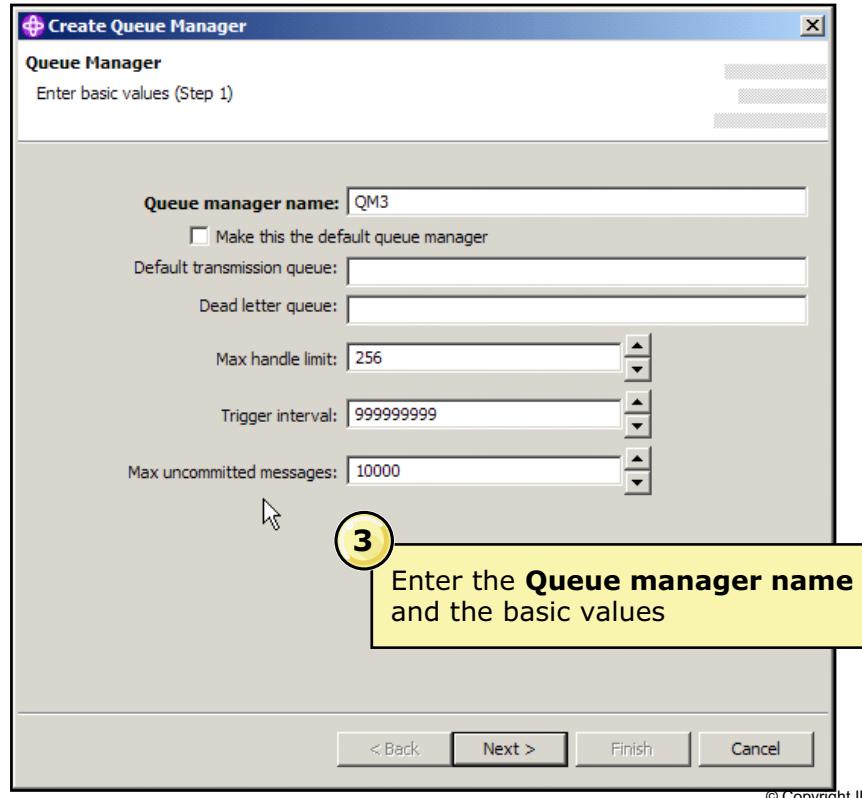


Figure 4-19. Creating a local queue manager with IBM MQ Explorer (2 of 5)

WM666 / ZM6661.0

### Notes:

As shown in the figure, the next step for creating a queue manager in IBM MQ Explorer is to specify a queue manager name and other basic configuration. Basic configuration includes specifying a default transmission queue, dead-letter queue, and trigger interval.



## Creating a local queue manager with IBM MQ Explorer (3 of 5)

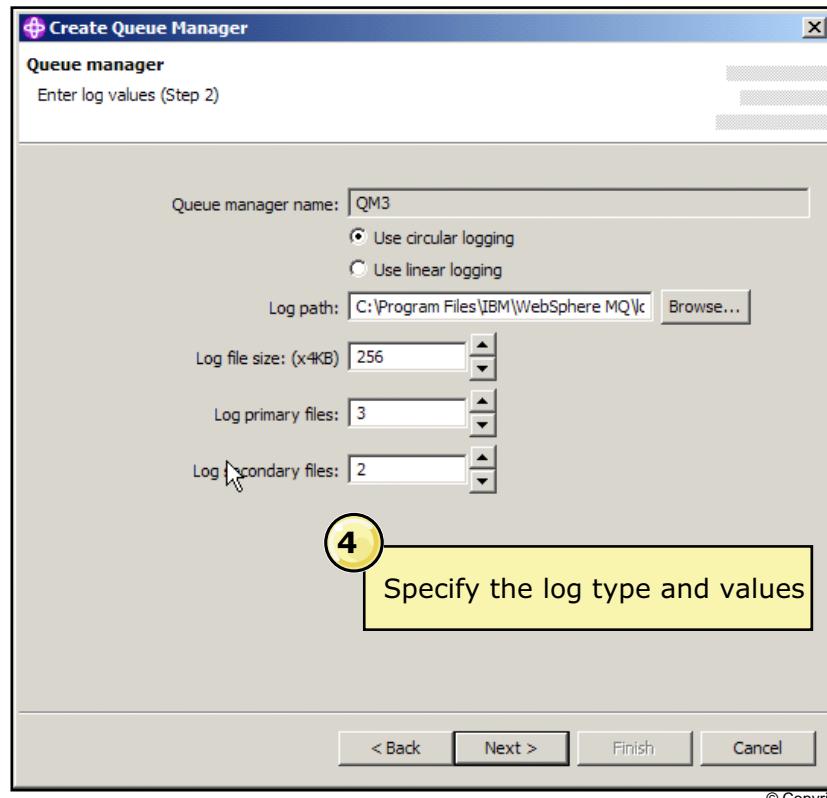


Figure 4-20. Creating a local queue manager with IBM MQ Explorer (3 of 5)

WM666 / ZM6661.0

### Notes:

The next step for creating a queue manager is to specify the queue manager log values.

The log data is held in a series of files called log files. The log file size is specified in units of 4 KB pages.

For more information about the IBM MQ log files, see the IBM Knowledge Center for IBM MQ.

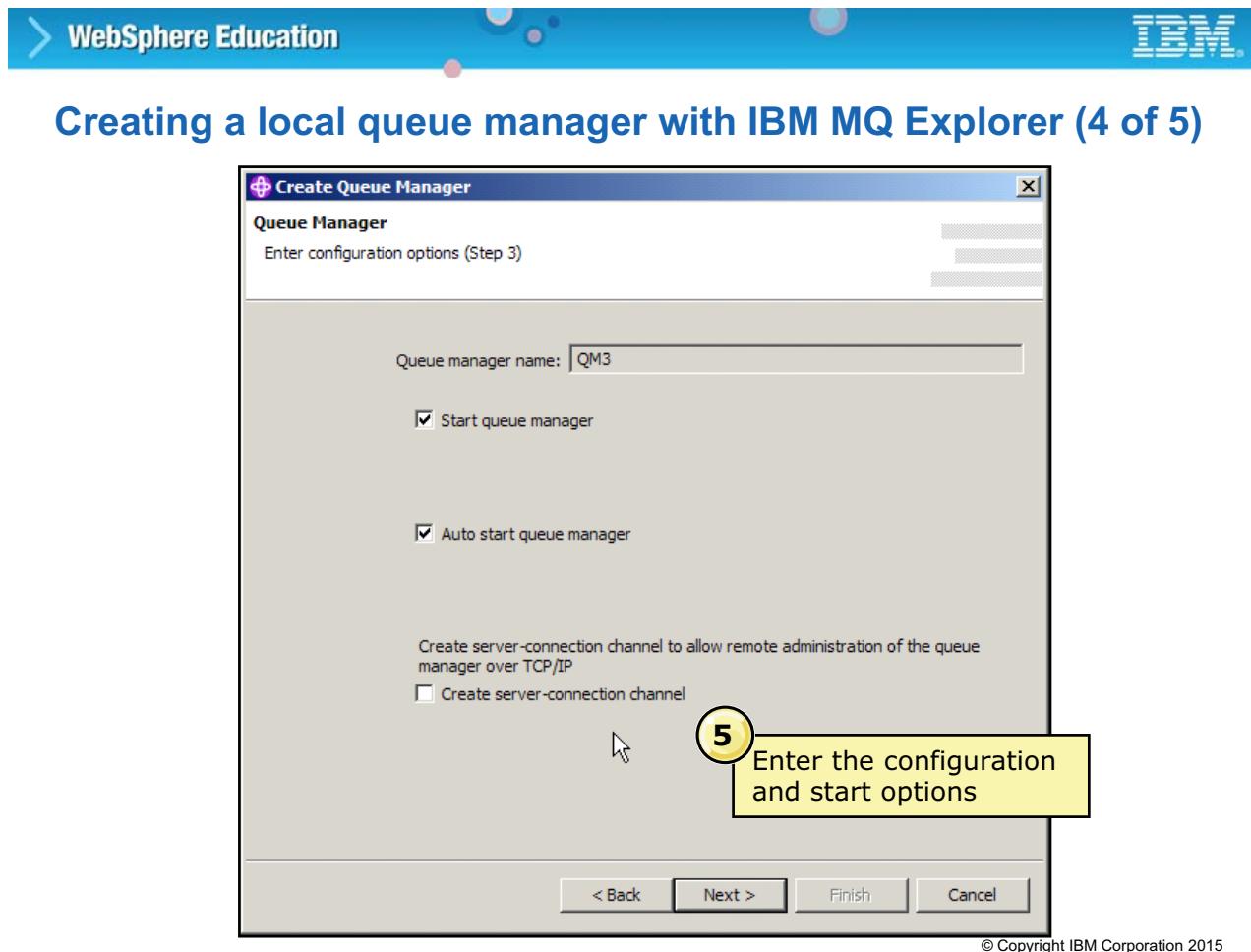


Figure 4-21. Creating a local queue manager with IBM MQ Explorer (4 of 5)

WM666 / ZM6661.0

## Notes:

The next step for creating a queue manager is to enter the queue manager configuration options.

- Select **Start queue manager** if you want the queue manager to start immediately after it is created.
- Select **Auto start queue manager** if you want the queue manager to automatically start after a system restart.
- Select **Create server-connection channel** to allow another administrator on another computer to manage this queue manager on this computer. Enable remote administration for single-point administration.

## Creating a local queue manager with IBM MQ Explorer (5 of 5)

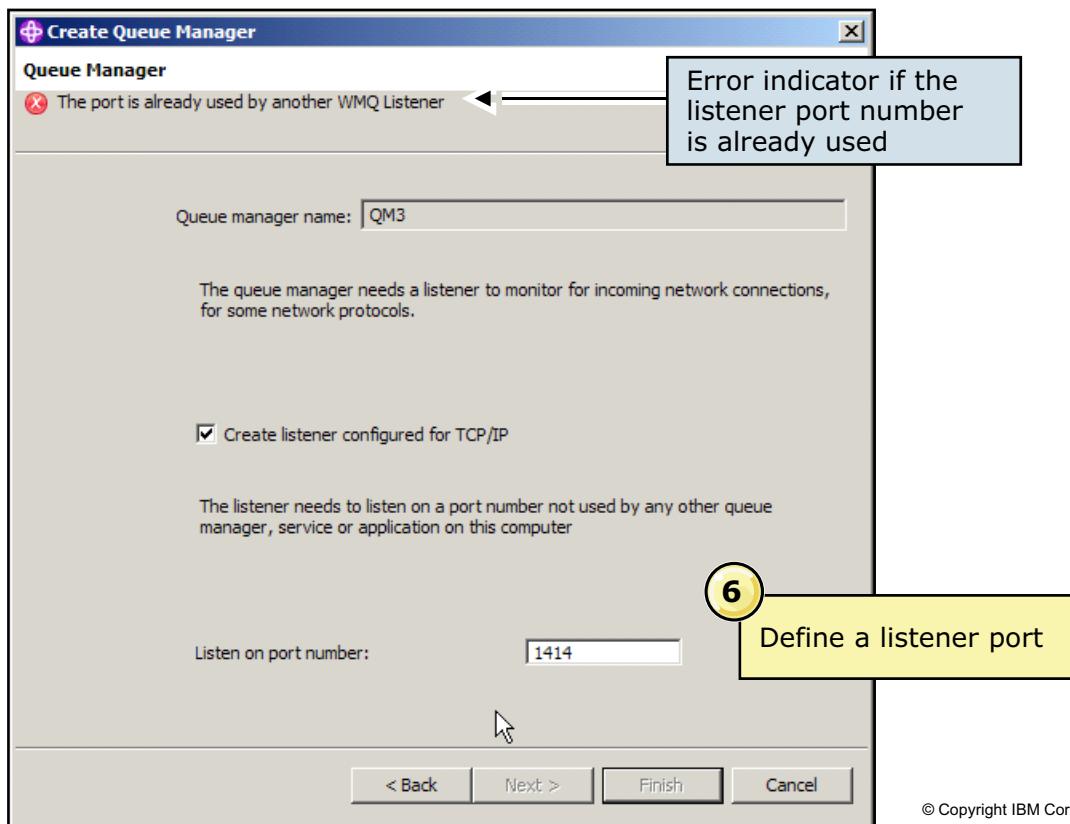


Figure 4-22. Creating a local queue manager with IBM MQ Explorer (5 of 5)

WM666 / ZM6661.0

### Notes:

The final step for creating a queue manager is to create a listener for TCP/IP and identify the listener port value.

A unique port number is required for each listener on the same host. If IBM MQ detects that another listener is using TCP/IP port number, an error message is displayed at the top of the page, as shown in the figure.

The screenshot shows the MQ Explorer interface with the title "MQ Explorer - Content" and the queue manager name "Queue Manager QM1". The interface is divided into three main sections:

- Connection QuickView:** Shows connection status information such as status (Connected), type (Local), and refresh interval (15).
- Status QuickView:** Shows the status of the queue manager, including command server status (Running), channel initiator status (Running), and installation name (Installation1).
- Properties QuickView:** Shows properties of the queue manager, including name (QM1), description (Windows), and command level (750).

Each section is highlighted with a gray box and labeled "Connection QuickView", "Status QuickView", and "Properties QuickView" respectively.

© Copyright IBM Corporation 2015

Figure 4-23. Queue manager content view

WM666 / ZM6661.0

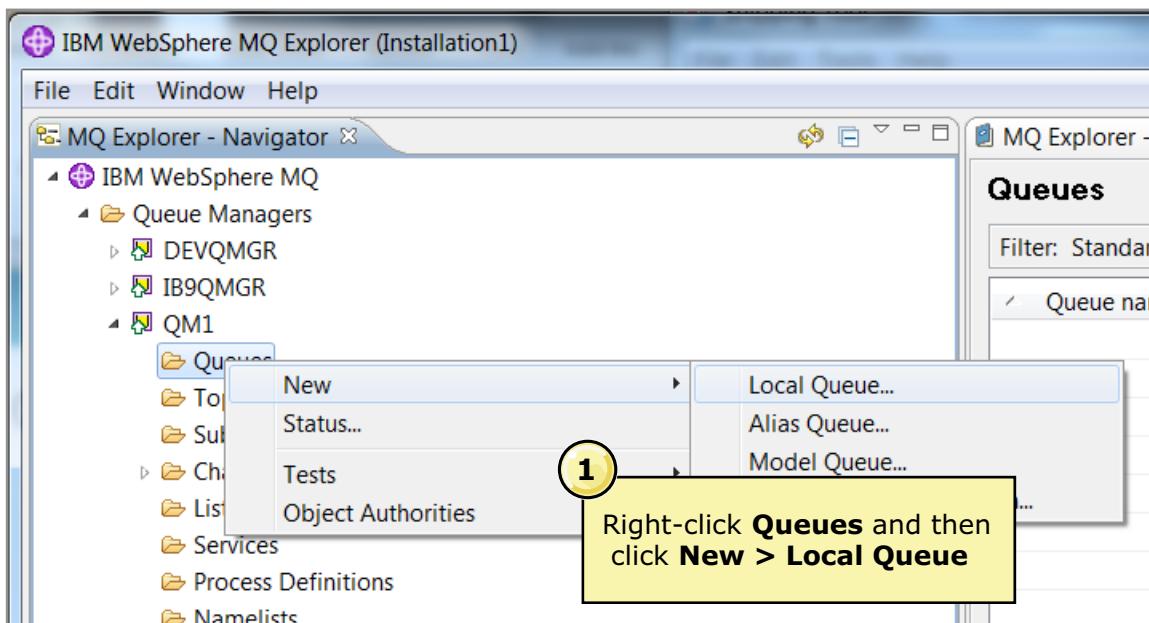
## Notes:

The Queue Manager content view is divided into three QuickView panes.

- The **Connection QuickView** shows the queue manager connection status information, such as connection status, connection type, and connection name.
- The **Status QuickView** shows the status of the queue manager. It includes the command server status, channel initiator status, and the installation name.
- The **Properties QuickView** shows some of the configurable properties of the queue manager such as the operating system, command level (version of IBM MQ), and the default transmission queue.



## Creating a local queue with IBM MQ Explorer (1 of 3)



© Copyright IBM Corporation 2015

Figure 4-24. Creating a local queue with IBM MQ Explorer (1 of 3)

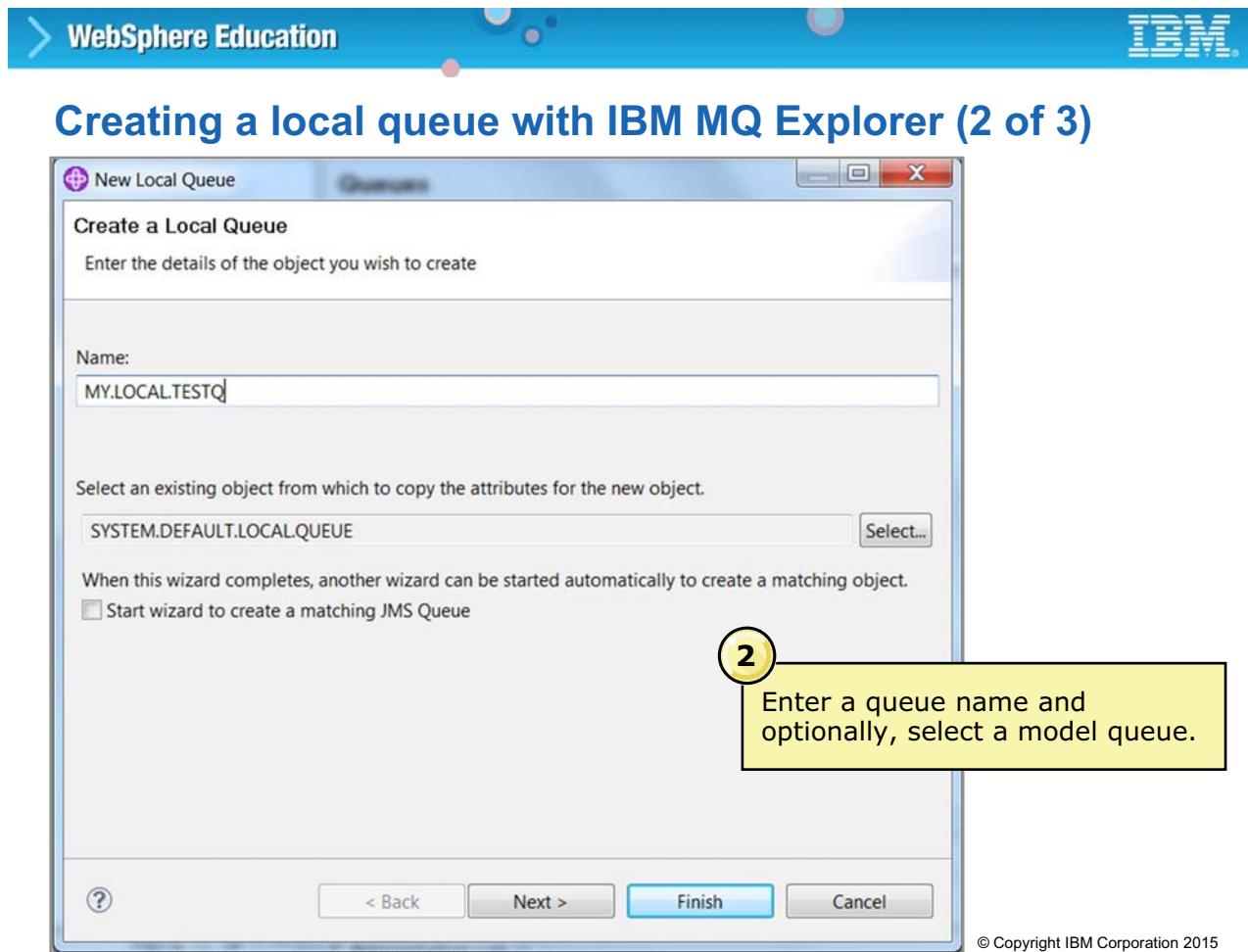
WM666 / ZM6661.0

### Notes:

It is possible to define queues for any local queue manager that is connected to IBM MQ Explorer.

To create a local queue:

1. Right-click the **Queues** folder under the queue manager in the **MQ Explorer - Navigator** view and then click **New > Local Queue**.



© Copyright IBM Corporation 2015

Figure 4-25. Creating a local queue with IBM MQ Explorer (2 of 3)

WM666 / ZM6661.0

### Notes:

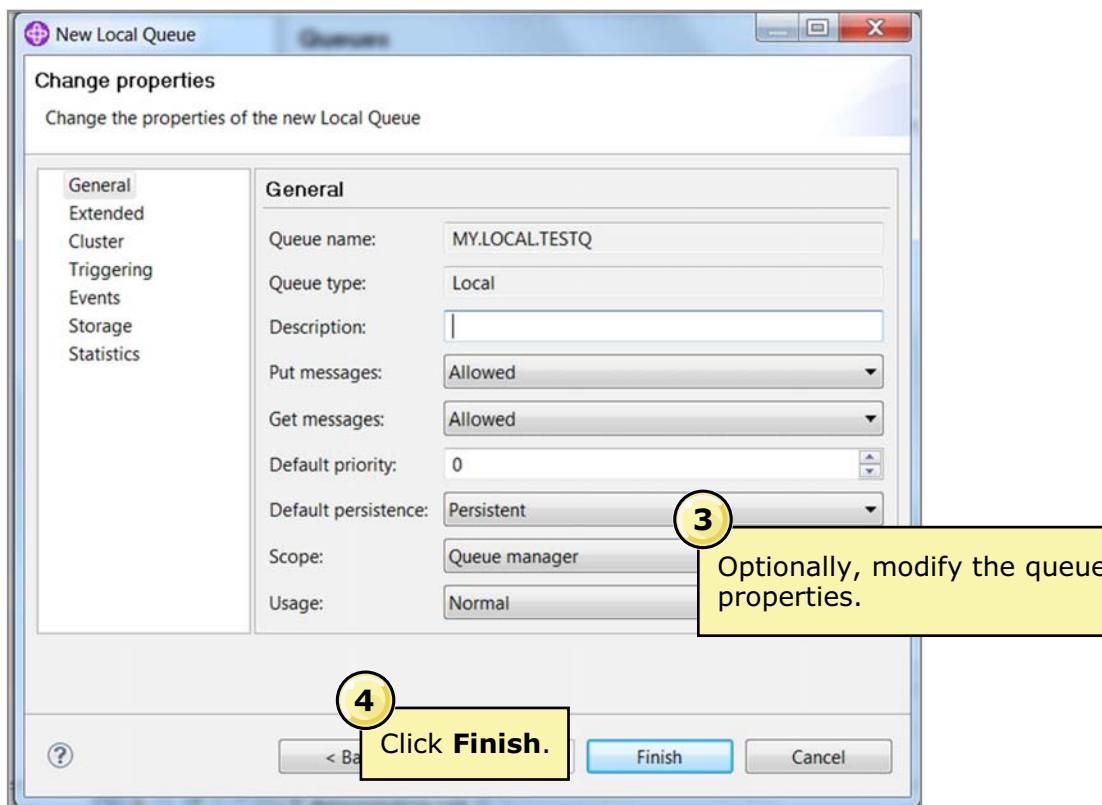
The next step for creating a queue is to enter a queue name.

By default, IBM MQ uses a default SYSTEM queue as the model for the new queue. You can specify a different queue to use as the model for this queue if you want.

There is also an option to automatically start another wizard to create a matching JMS queue.



## Creating a local queue with IBM MQ Explorer (3 of 3)



© Copyright IBM Corporation 2015

Figure 4-26. Creating a local queue with IBM MQ Explorer (3 of 3)

WM666 / ZM6661.0

### Notes:

The next step is to modify the queue properties, if necessary. The queue properties are categorized by function:

- General
- Extended
- Cluster
- Triggering
- Events
- Storage
- Statistics



## Creating a default queue manager for an integration node

- Prerequisites
  - IBM MQ server installation
  - User must be `mqm` and `mqbrkrs` operating system groups
  - On Linux and UNIX, ensure that the command environment has been initialized, by running the `mqsinprofile` command.

1. In IBM MQ, create and start the queue manager
2. On Linux and UNIX systems, run the `setmqenv` command to configure the IBM MQ environment where you want the integration node to run
3. Specify the queue manager for the integration node

© Copyright IBM Corporation 2015

Figure 4-27. Creating a default queue manager for an integration node

WM666 / ZM6661.0

### Notes:

Some IBM Integration Bus capabilities require access to IBM MQ components. A subset of these capabilities require a set of system queues to be created on the queue manager that is associated with the integration node. The system queues are used to store information about in-flight messages.

## Specifying a default queue manager for an integration node

- Setting the default queue manager identifies the queue manager to use for:
  - IBM MQ nodes if no queue manager is specified explicitly on the node
  - Message flow nodes that require a queue manager to be specified on the integration node
  - Global transaction management
  - Publish/subscribe messages from the integration node

```
mqsicreatebroker IntNodeName -q QmgrName
```

- Creates an integration node with a default queue manager
- Does not define and start the queue manager

```
mqsicchangebroker IntNodeName -q QmgrName
```

- Changes integration node definition to add a default queue manager

© Copyright IBM Corporation 2015

Figure 4-28. Specifying a default queue manager for an integration node

WM666 / ZM6661.0

### Notes:

As mentioned previously, some operations require that the integration node is connected to a queue manager.

When you use the **mqsiccreatebroker** command on distributed systems with the **-q** parameter, the named local queue manager is associated with the integration node. This queue manager is used by default for IBM MQ processing in the message flow if no queue manager is specified on the message flow node.

The queue manager that is specified on the integration node is also required for message flow nodes that use system queues to store state information about the messages. Message flow nodes that store state information include event-driven processing nodes that are used for aggregation and timeout flows, message collections, and message sequences.

You can use the **mqsicchangebroker** command on distributed systems to add or change the default queue manager. You must stop the integration node before you enter the **mqsicchangebroker** command, and restart the integration node for the changes to take effect.



## IBM Integration Bus system queues

- Some IBM Integration Bus capabilities require access to IBM MQ queue manager and default system queues on the queue manager
  - Flow control with the TimeoutControl and TimeoutNotification nodes
  - Record and replay
  - Message aggregation with AggregateControl, AggregateRequest, and AggregateReply nodes
  - Advanced data processing with Collector, Sequence, and Resequence nodes
  - Using the HTTP proxy servlet with the integration node
  - Using an integration node listener instead of embedded listeners in the integration node
  - Using accounting and statistics by using the web user interface and IBM MQ

© Copyright IBM Corporation 2015

Figure 4-29. IBM Integration Bus system queues

WM666 / ZM6661.0

### Notes:

Some IBM Integration Bus features require access to system queues on a queue manager that is specified on the integration node, for the storage and retrieval of state information.



## Creating the IBM Integration Bus system queues

1. Verify that you have an IBM MQ server and a queue manager to use with the integration node
2. Ensure that you are a member of the “mqm” and “mqbrkrs” operating system groups
3. From the command environment (on Linux and UNIX) or the IBM Integration Console (on Windows), go to the following directory:  
Windows: *iib\_install\_dir\server\sample\wmq*  
Linux, UNIX: *iib\_install\_dir/server/sample/wmq*
4. Run the following command:  
Windows: *iib\_createqueues.cmd QmgrName*  
Linux, UNIX: *./iib\_createqueues.sh QmgrName*  
*QmgrName* is the name of the IBM MQ queue manager where you want to create the queues

© Copyright IBM Corporation 2015

Figure 4-30. Creating the IBM Integration Bus system queues

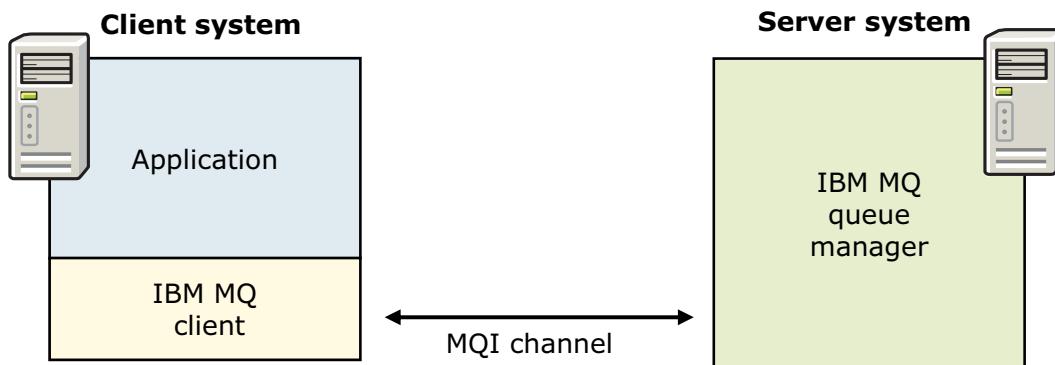
WM666 / ZM6661.0

### Notes:

This figure lists the steps for creating the system queues that IBM Integration Bus needs for some functions.



## IBM MQ client



- IBM MQ component on a system without a queue manager
  - Communicates with the queue manager by using an *MQI channel*
  - Allows an application that runs on the client system to connect to a queue manager that is running on another system

© Copyright IBM Corporation 2015

Figure 4-31. IBM MQ client

WM666 / ZM6661.0

### Notes:

An IBM MQ client is a component of IBM MQ that allows an application that is running on one system to send MQI calls to a queue manager that is running on another system.

The client connection receives the input parameters of an MQI call from the application and sends them over a communications connection to the server connection on the same system as the queue manager. The server connection then sends the MQI call to the queue manager on behalf of the application. After the queue manager completes the MQI call, the server connection sends the output parameters of the callback to the client connection, which then passes them onto the application.

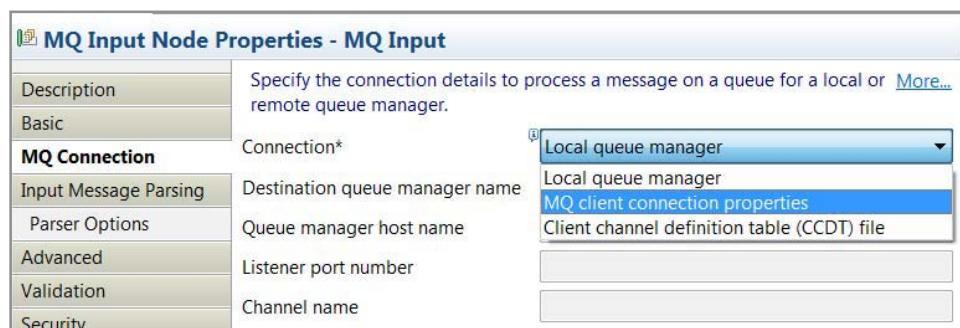
The combination of a client connection, a server connection, and a communications connection between them (for example, an SNA LU6.2 conversation or a TCP connection) is called an MQI channel.

The full range of MQI calls and options are available to a client application.

## Client connections to IBM MQ

- If IBM MQ is not running on the same server as the integration node, configure a client connection
- Define a client connection on the MQ nodes by using one of the following options:
  - Select **MQ client connection properties** and specify the connection details of the queue manager explicitly
  - Select **Client channel definition table (CCDT)** and specify the location of the CCDT file by running the `mqsicchangeproperties` command with the integration node registry object property `mqCCDT`

Example: `mqsicchangeproperties IBNODE -o BrokerRegistry -n mqCCDT  
-v "C:\Program Files\IBM\MQ\Qmgrs\QM1@\ipcc\AMQCLCHL.TAB"`



© Copyright IBM Corporation 2015

Figure 4-32. Client connections to IBM MQ

WM666 / ZM6661.0

### Notes:

## MQEndpoint policy

- Use an MQEndpoint policy to control the values of IBM MQ node connection properties at run time
- Overrides any equivalent properties that were specified for the **MQ Connection** properties in the message flow definition
- Validated at run time
- If you set the **ccdt** property in an MQEndpoint policy document, you must also run the **mqsischangeproperties** command to specify the IBM MQ client CCDT file path:

```
mqsischangeproperties IntNodeName -o BrokerRegistry  
-n mqCCDT -v CCDT Path
```

- Connect to a secured local or remote queue manager, by setting a user name and password in the policy
- Choose whether to use the SSL protocol when a client connection is made to a remote queue manager

© Copyright IBM Corporation 2015

Figure 4-33. MQEndpoint policy

WM666 / ZM6661.0

### Notes:

## MQEndpoint policy document

- Generated by default from the connection details
- Automatically attached when the IBM MQ service is applied to an IBM MQ message flow node

```
<policy type="MQEndpoint">
  <policyProperties>
    <mqConnectionDetailsPolicy>
      <connection>CLIENT</connection>
      <destinationQueueManagerName>QMGR1</destinationQueueManagerName>
      <queueManagerHostname>localhost</queueManagerHostname>
      <listenerPortNumber>1414</listenerPortNumber>
      <channelName>SYSTEM.DEF.SVRCONN</channelName>
      <securityIdentity>SecId</securityIdentity>
      <useSSL>true</useSSL>
      <SSLPeerName>CN=IIB10*</SSLPeerName>
      <SSLCipherSpec>TLS_RSA_WITH_AES_128_CBC_SHA</SSLCipherSpec>
    </mqConnectionDetailsPolicy>
  </policyProperties>
</policy>
```

© Copyright IBM Corporation 2015

Figure 4-34. MQEndpoint policy document

WM666 / ZM6661.0

### Notes:

## Creating an MQEndpoint policy

- Use the IBM Integration Toolkit to generate a policy document from an existing message flow node
  - Operational property values can be edited as required
  - Generated policy document can be saved in the Integration Registry
- Use the IBM Integration web user interface to create, retrieve, update, and delete policy documents that are stored in the Integration Registry
- Use the following commands to create, retrieve, update, and delete policy documents:
  - To create a policy, use the `mqsicreatepolicy` command
  - To retrieve details of a policy, use the `mqsi reportpolicy` command
  - To update a policy, use the `mqsicangepolicy` command
  - To delete a policy, use the `mqsideletepolicy` command

© Copyright IBM Corporation 2015

Figure 4-35. Creating an MQEndpoint policy

WM666 / ZM6661.0

### Notes:



## Generating an MQ Policy in the IBM Integration Toolkit (1 of 2)

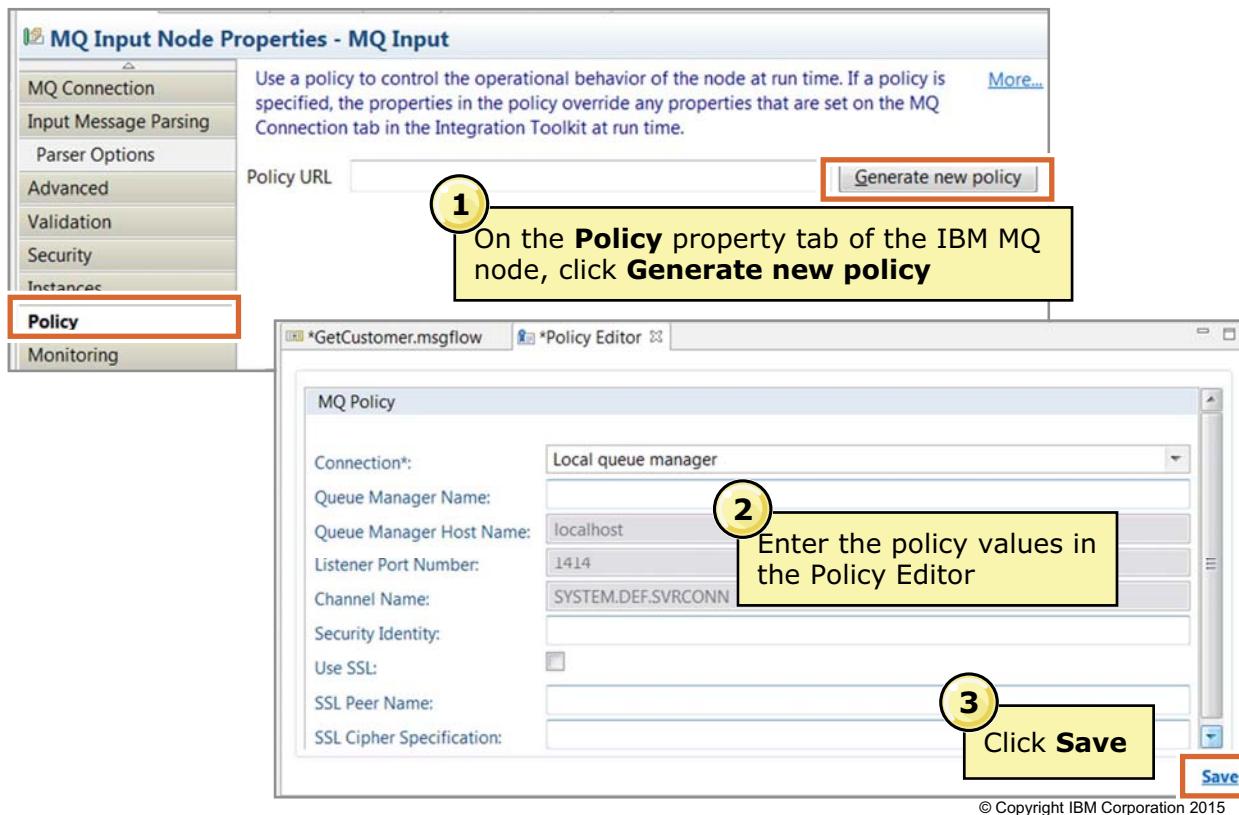


Figure 4-36. Generating an MQ Policy in the IBM Integration Toolkit (1 of 2)

WM666 / ZM6661.0

### Notes:

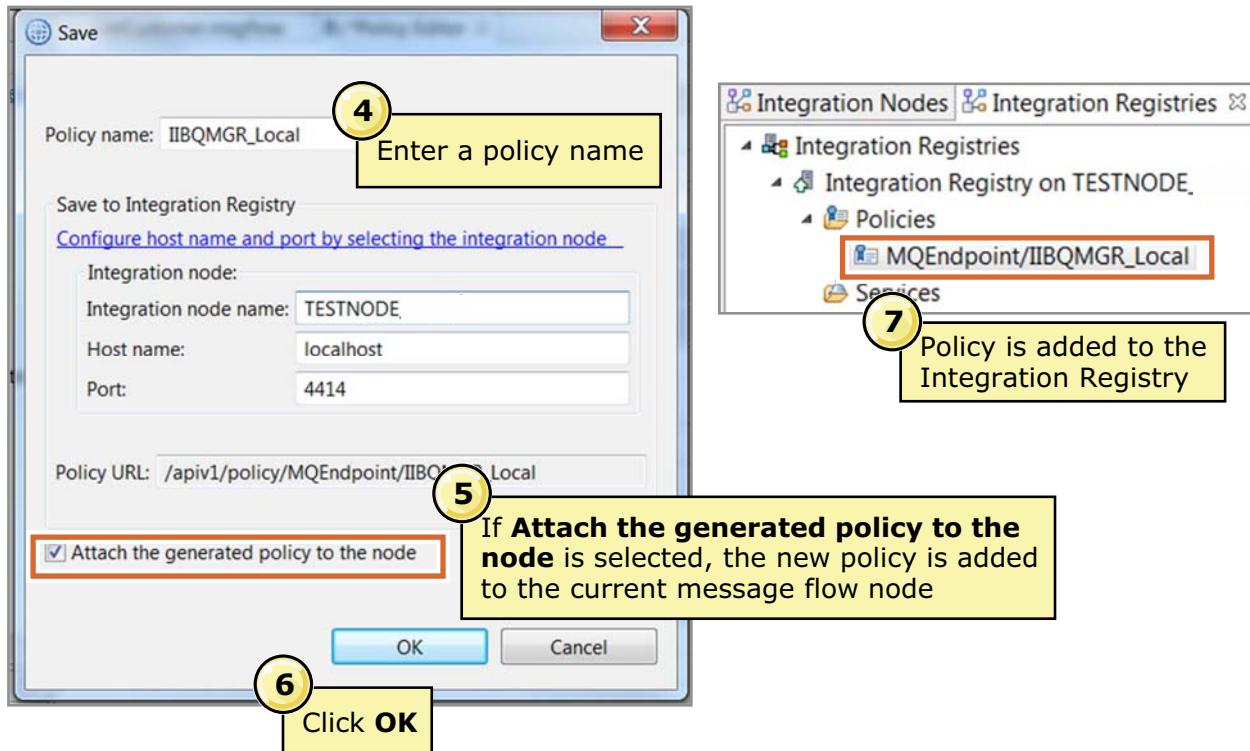
You can find the URL of a saved policy by viewing the policy in the IBM Integration web user interface.

In the IBM Integration web user navigation tree, expand **Operational Policy** followed by the policy type, and select the policy name that you want to view. The format of a policy URL is `/apiv1/policy/policy type/policy name`. For example,

`/apiv1/policy/MQTTSubscribe/Policy1`



## Generating an MQ Policy in the IBM Integration Toolkit (2 of 2)



© Copyright IBM Corporation 2015

Figure 4-37. Generating an MQ Policy in the IBM Integration Toolkit (2 of 2)

WM666 / ZM6661.0

### Notes:

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.

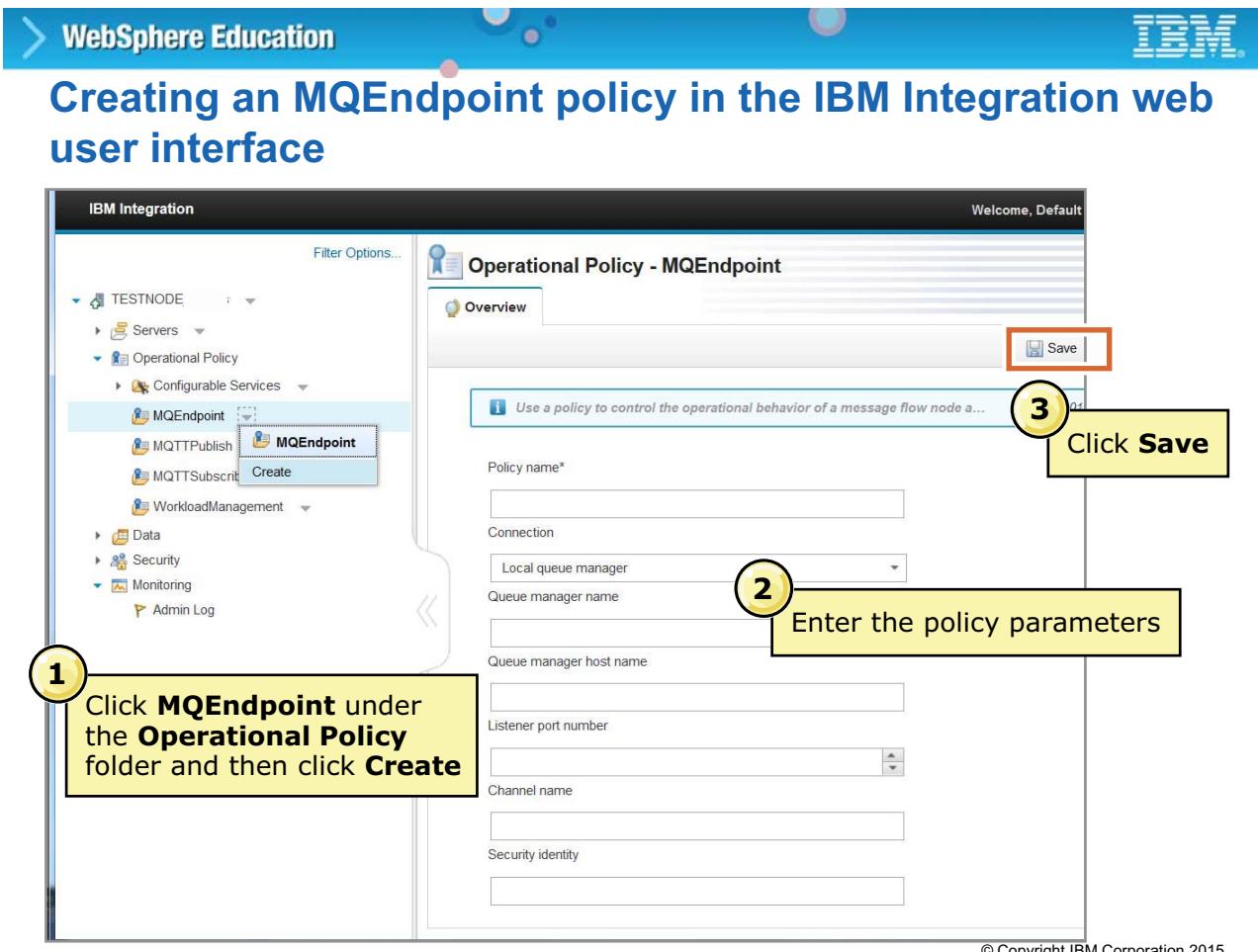


Figure 4-38. Creating an MQEndpoint policy in the IBM Integration web user interface

WM666 / ZM6661.0

**Notes:**



## Integration Registry

- Contains service definitions and operational policies
  - Facilitates collaboration and reuse
  - Hosted in an integration node
- Populated by:
  - Publishing discovered IBM MQ services from IBM Integration Toolkit
  - Publishing MQEndpoint, MQTTPublish, MQTTSubscribe, and workload management policies
- Can be accessed in the IBM Integration Toolkit and the IBM Integration web user interface

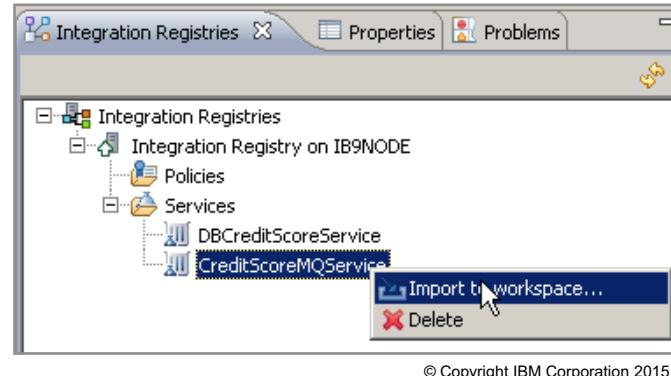


Figure 4-39. Integration Registry

WM666 / ZM6661.0

### Notes:

The Integration Registry contains IBM Integration Bus service definitions and operational policies.



## Testing message flows by using IBM MQ Explorer

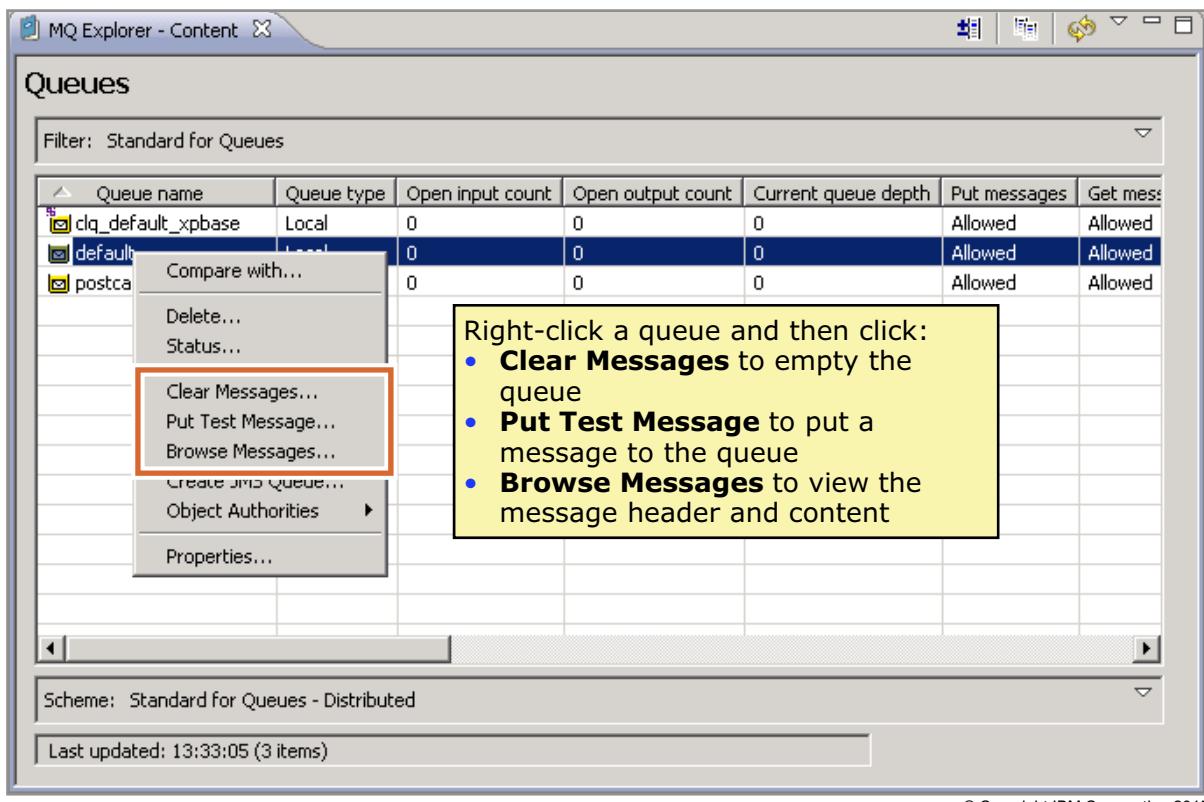


Figure 4-40. Testing message flows by using IBM MQ Explorer

WM666 / ZM6661.0

### Notes:

You can test message flows that contain MQInput or MQGet nodes by using IBM MQ Explorer to put test messages on input queues. You can also use the IBM MQ Explorer to verify that messages were put to output queues.



## Testing message flows with RFHUtil (SupportPac IH03)

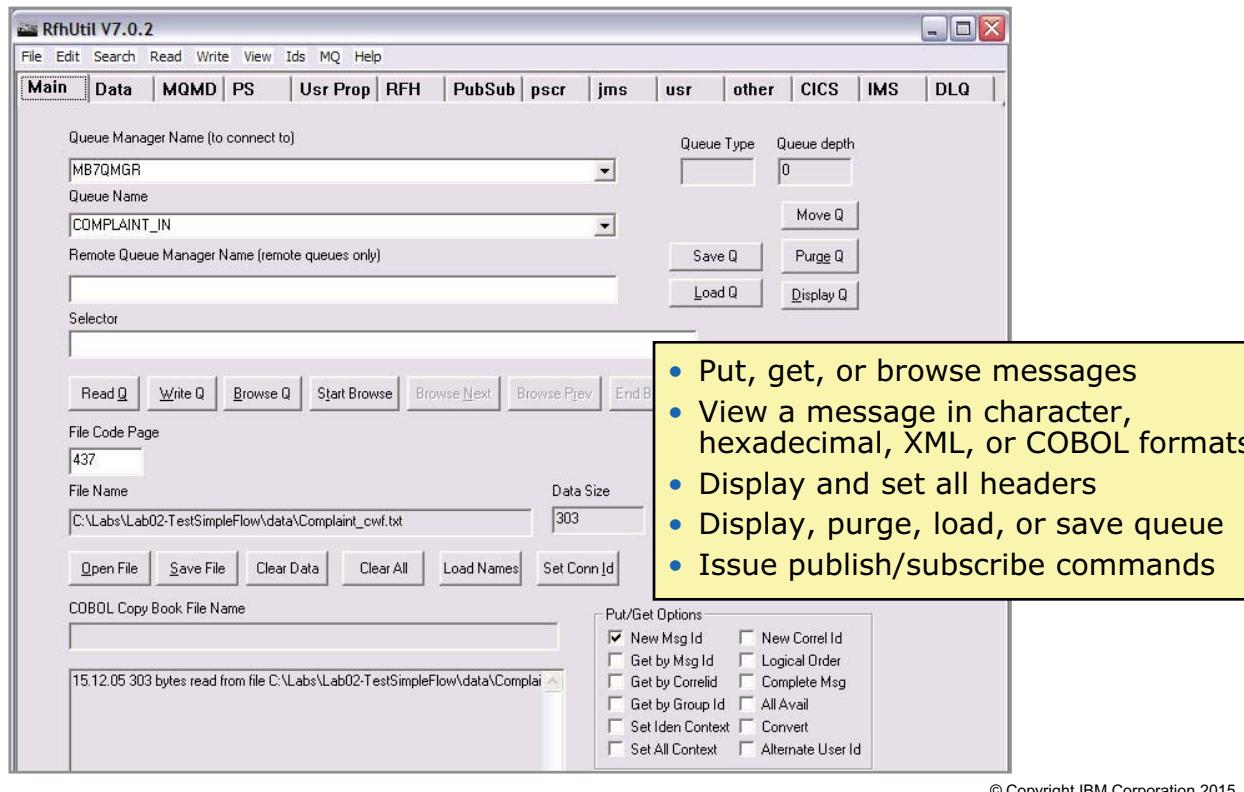


Figure 4-41. Testing message flows with RFHUtil (SupportPac IH03)

WM666 / ZM6661.0

### Notes:

The RFHUtil utility program is a WebSphere Message Broker SupportPac. It reads data from files, queues or both. It also writes data to files, queues or both, and displays data in various formats.

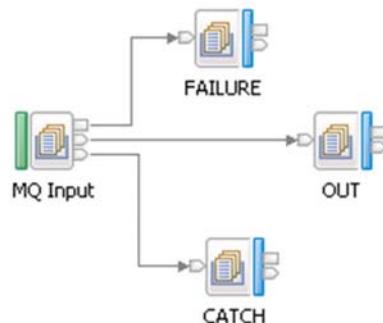
You can use this program to put messages to input queues and verify that messages were put on output queues.

The user data portion of the message can be displayed in various formats, but it cannot be changed. Another program must be used to create or change the user data.

The utility program can add rules and formatting headers (RFH) to messages or files. It writes and formats these headers when they are found in messages or files that it reads. The headers can include publish/subscribe commands.

You can download RFHUtil from the IBM SupportPac website at: <http://www.ibm.com/support>.

## Message flow result dependencies



- Wired Failure and Catch terminals
- **Transaction mode** property on an Input node
  - **Automatic**: The incoming persistence property determines the sync point
  - **Yes**: All messages are received under the sync point and are handled as persistent (default)
  - **No**: All messages are not received under the sync point and are handled as non-persistent
- IBM MQ configuration
  - Back-out queue name (BOQNAME) and back-out threshold (BOTHRESH) properties of the input queue
  - Dead-letter queue (DLQ) property of the queue manager

© Copyright IBM Corporation 2015

Figure 4-42. Message flow result dependencies

WM666 / ZM6661.0

### Notes:

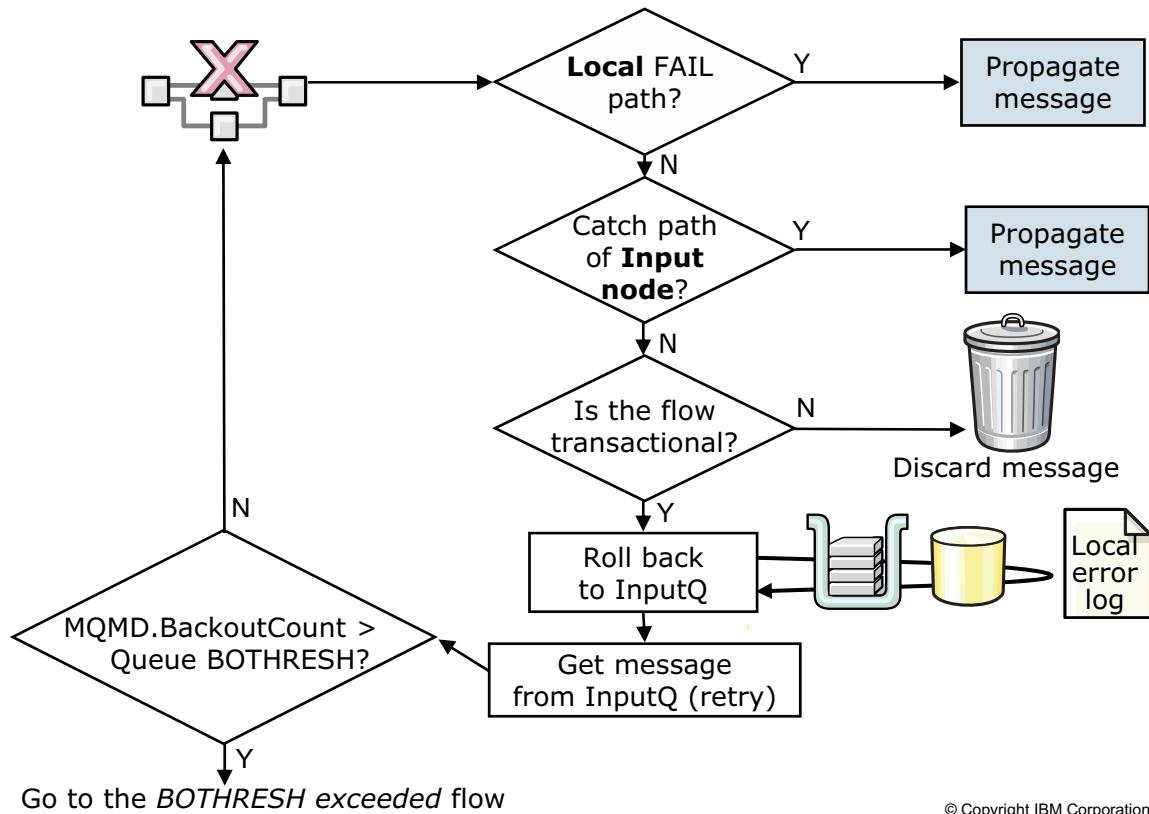
If a message flow completes successfully, messages almost always end up on the output as designed. However, depending on the terminals that are connected and if there is a problem, the message can go to another destination.

Other factors can affect the message path if a runtime error occurs. For example, the MQInput node **Transaction mode** property that controls whether the incoming message is received under sync point.

- If **Transaction mode** is set to **Automatic**, the incoming message is received under sync point if it is marked as a persistent message by the application; otherwise, it is not received under sync point. If a failure occurs, the message is discarded if it is marked as non-persistent.
- If **Transaction mode** is set to **Yes**, the incoming message is received under sync point. If there is a failure, a rollback can occur regardless of message persistence.
- If **Transaction mode** is set to **Yes No**, the incoming message is not received under sync point. If a failure occurs, the message is discarded regardless of message persistence.

IBM MQ also has two other queues where messages might appear: the queue manager dead-letter queue and a backout queue.

## Message flow error behavior with IBM MQ



© Copyright IBM Corporation 2015

Figure 4-43. Message flow error behavior with IBM MQ

WM666 / ZM6661.0

### Notes:

The figure shows the events that occur when an error occurs in a processing node and the flow starts with an MQInput node.

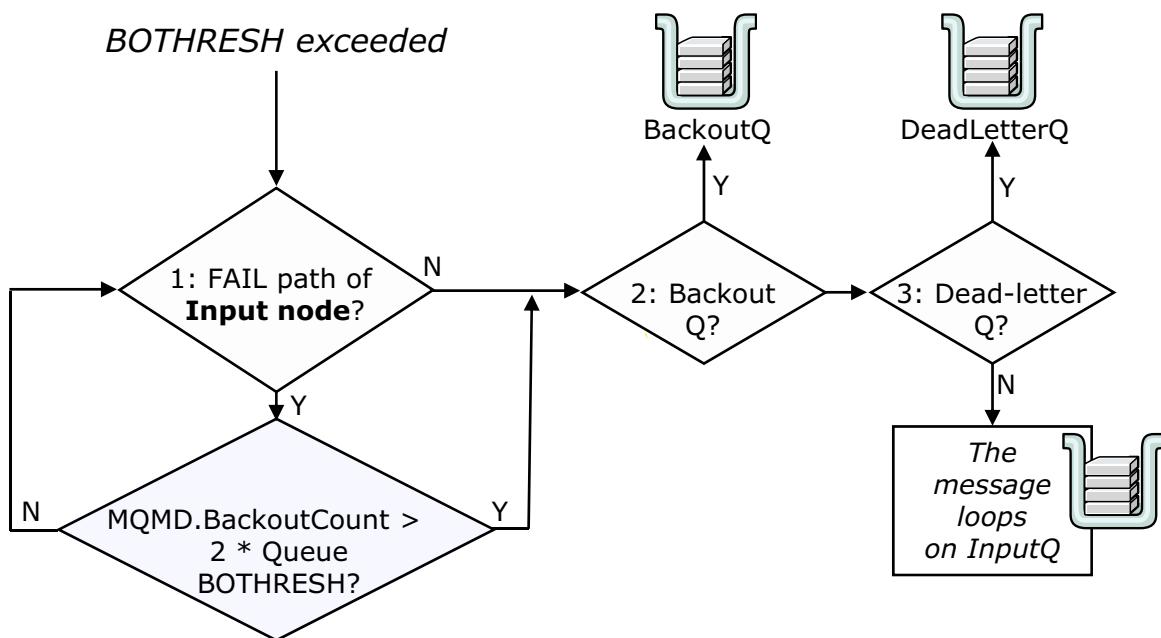
The levels of error handling are like filters. If the **Failure** terminal of the node where the exception occurred is connected, the message is propagated out the **Failure** terminal. If the Failure terminal is not connected, the next step is to check the **Catch** terminal on the MQInput node.

If a node is wired to the **Catch** terminal of the MQInput node, the message is propagated to that node. If the **Catch** terminal of the MQInput node is not connected, then the next step is to check the **Transaction mode** setting on the MQInput node.

If the message is not under sync point (not transactional), it is discarded. If the message is under sync point the message is rolled back to the input queue, and the message flow restarts. The second attempt might not work and ends with another backout.

The backout threshold (BOTHRESH) property on the input queue limits the number of times the message flow tries to process the message. The next page shows the behavior that occurs when the input queue backout threshold is exceeded.

## Message flow error behavior: BOTHRESH exceeded



© Copyright IBM Corporation 2015

Figure 4-44. Message flow error behavior: BOTHRESH exceeded

WM666 / ZM6661.0

### Notes:

If the backout threshold is reached, the message destination depends on whether the Failure terminal the input node is wired. The message destination also depends on the existence of a backout queue (BOQ) for the input queue, and existence of a queue manager dead-letter queue. The queues are checked in the order that is shown in the flow chart.

If a failure occurs beyond the FAILURE terminal of the input node, further retry attempts occur until the backout count in the MQMD is twice the backout threshold that is set for the input queue. When this limit is reached, the node attempts to put the message to either the backout queue or dead-letter queue. If the backout queue or the dead-letter queue do not exist, or if any of the queues fail, the message ends looping on the input queue and prevents any new messages from being processed. In these cases, the error condition must be corrected manually.

## Unit summary

Having completed this unit, you should be able to:

- Describe the IBM MQ connection options
- Examine the properties of the IBM MQ nodes
- Predict the location of the message if a runtime error is encountered during message flow processing
- Attach an MQEndpoint policy to one or more IBM MQ nodes in a message flow to control connection details at run time

© Copyright IBM Corporation 2015

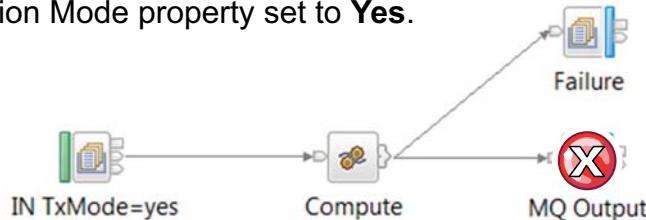
Figure 4-45. Unit summary

WM666 / ZM6661.0

### Notes:

## Checkpoint questions

- In this flow, an MQInput node is wired to a Compute node. The Compute node **Failure** terminal is wired to an MQOutput node that is named Failure. The message fails on the node that is labeled **MQ Output**. The MQInput node has the Transaction Mode property set to **Yes**.



Where would you find the message if the dead-letter queue of queue manager is DLQ and a Backout Queue is configured?

- On the queue that is identified on Failure node
- On the dead-letter queue
- On the Backout queue
- The message is discarded

© Copyright IBM Corporation 2015

Figure 4-46. Checkpoint questions

WM666 / ZM6661.0

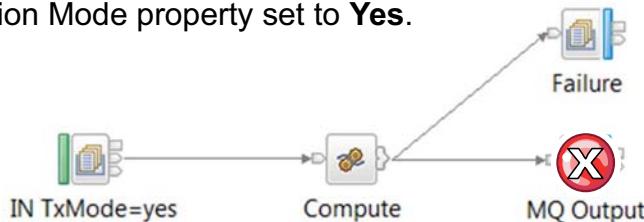
### Notes:

Write your answers here:

1.

## Checkpoint answers

- In this flow, an MQInput node is wired to a Compute node. The Compute node **Failure** terminal is wired to an MQOutput node that is named Failure. The message fails on the node that is labeled **MQ Output**. The MQInput node has the Transaction Mode property set to **Yes**.



Where would you find the message if the dead-letter queue of queue manager is DLQ and a Backout Queue is configured?

- On the queue that is identified on Failure node
- On the dead-letter queue
- On the Backout queue
- The message is discarded

**Answer:** c. The Failure terminal of the MQ Output node is not wired so the message rolls back to the input queue. After reaching the backout threshold, the message is put the Backout queue.

© Copyright IBM Corporation 2015

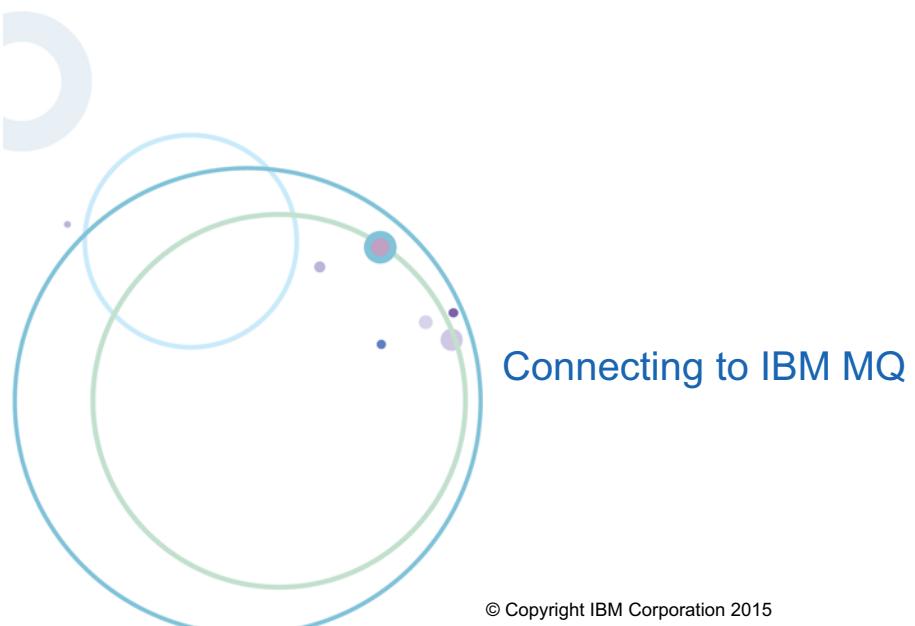
Figure 4-47. Checkpoint answers

WM666 / ZM6661.0

### Notes:



## Exercise 3



© Copyright IBM Corporation 2015

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

10.1

Figure 4-48. Exercise 3

WM666 / ZM6661.0

### Notes:



## Exercise objectives

After completing this exercise, you should be able to:

- Create an integration node that uses a default IBM MQ queue manager
- Share a default IBM MQ queue manager with multiple integration nodes
- Create a message flow that gets a message with an MQInput node and puts a message with an MQOutput node
- Edit a BAR file
- Manually deploy a BAR file
- Verify that the integration nodes that share a queue manager also share the workload

© Copyright IBM Corporation 2015

Figure 4-49. Exercise objectives

WM666 / ZM6661.0

### Notes:

See the *Student Exercises Guide* for detailed instructions.

# Unit 5. Controlling the flow of messages

## What this unit is about

In this unit, you learn about message processing nodes that are used to control the flow of a message.

## What you should be able to do

After completing this unit, you should be able to:

- Describe logical messages and the message assembly, and explain how they are used in IBM Integration Bus application programming
- Use the Filter and Route message processing nodes to examine the contents of a message and alter its flow
- Use the RouteToLabel and Label nodes to dynamically change the routing of messages
- Use the FlowOrder node to control the flow path order in which a message is processed through a message flow
- Create reusable subflows

## How you will check your progress

- Checkpoint questions
- Lab exercise

## References

IBM Knowledge Center

Appendix A: ESQL examples

## Unit objectives

After completing this unit, you should be able to:

- Describe logical messages and the message assembly, and explain how they are used in IBM Integration Bus application programming
- Use the Filter and Route message processing nodes to examine the contents of a message and alter its flow
- Use the RouteToLabel and Label nodes to dynamically change the routing of messages
- Use the FlowOrder node to control the flow path order in which a message is processed through a message flow
- Create reusable subflows

© Copyright IBM Corporation 2015

Figure 5-1. Unit objectives

WM666 / ZM6661.0

### Notes:

## Message parsing overview

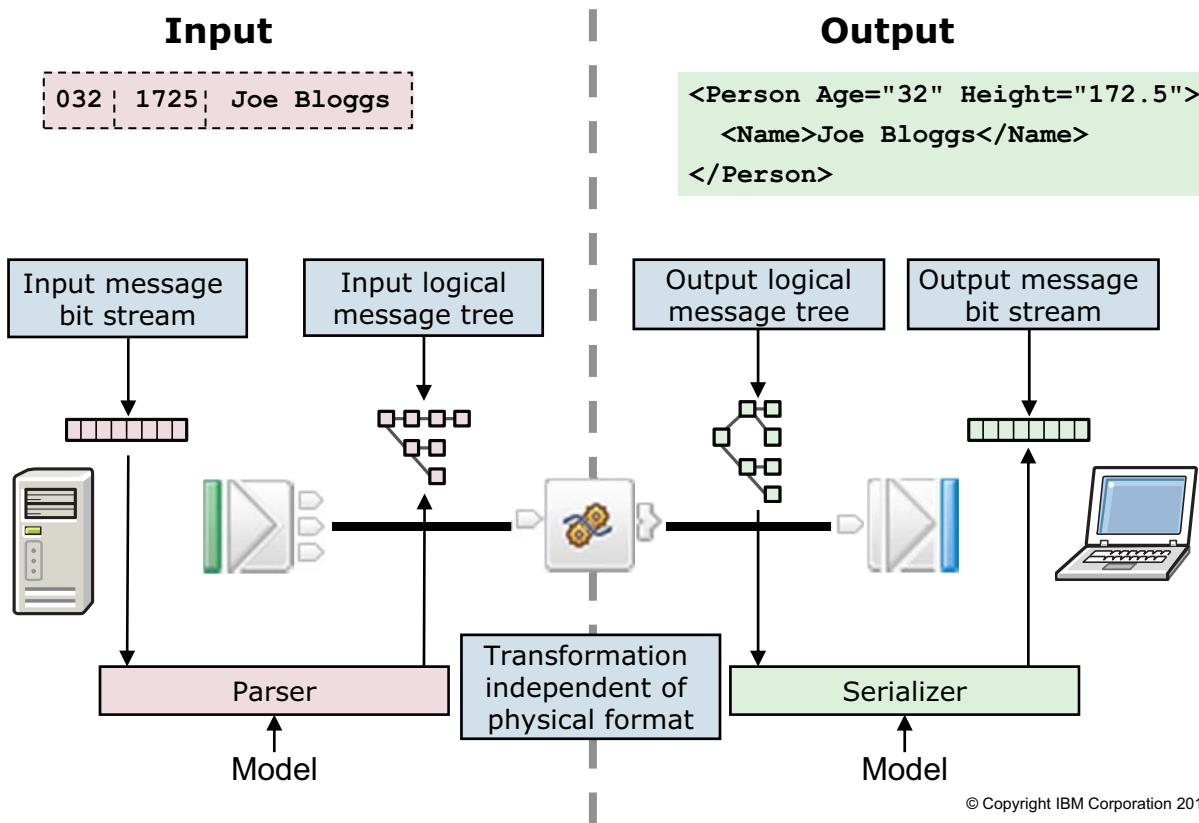


Figure 5-2. Message parsing overview

WM666 / ZM6661.0

### Notes:

An IBM Integration Bus message flow typically receives messages in a defined format, transforms them, and outputs them in a different format.

The example in the figure shows a COBOL data structure that is transformed into an XML document. In the example, the COBOL input message contains an age, height, and name field that arrives in the form of an input bit stream. Before the message flow can process it, it is converted into an input logical message tree for transformation by the message flow. A logical message tree is an integration node data structure that reflects the logical structure of the message that the processing nodes use. The structure is independent of the transformation logic.

In the example, logic is applied in a Compute node to transform the message and create an output logical message tree. The data is converted into an output bit stream that represents the message as an XML document.

The component of the integration node responsible for converting a bit stream into a logical message tree is a *parser*.

## Logical message model

- A consistent, convenient way to represent message content inside IBM Integration Bus
- Removes and isolates the physical details of the message
- Organized as a message assembly that contains four trees:
  - Message tree contains representation of the message properties and content
  - Environment tree for storing information while the message passes through the message flow
  - Local environment tree that the input node creates to store variables that can be referred to and updated by message processing nodes that occur later in the message flow
  - Exception list tree contains message flow exception information
- Element values are stored in Unicode to facilitate code conversions
- Elements in the tree are addressed by using XPath or the dotted name notation

Example: TRANSACTION.CUSTOMER.FIRST\_NAME

© Copyright IBM Corporation 2015

Figure 5-3. Logical message model

WM666 / ZM6661.0

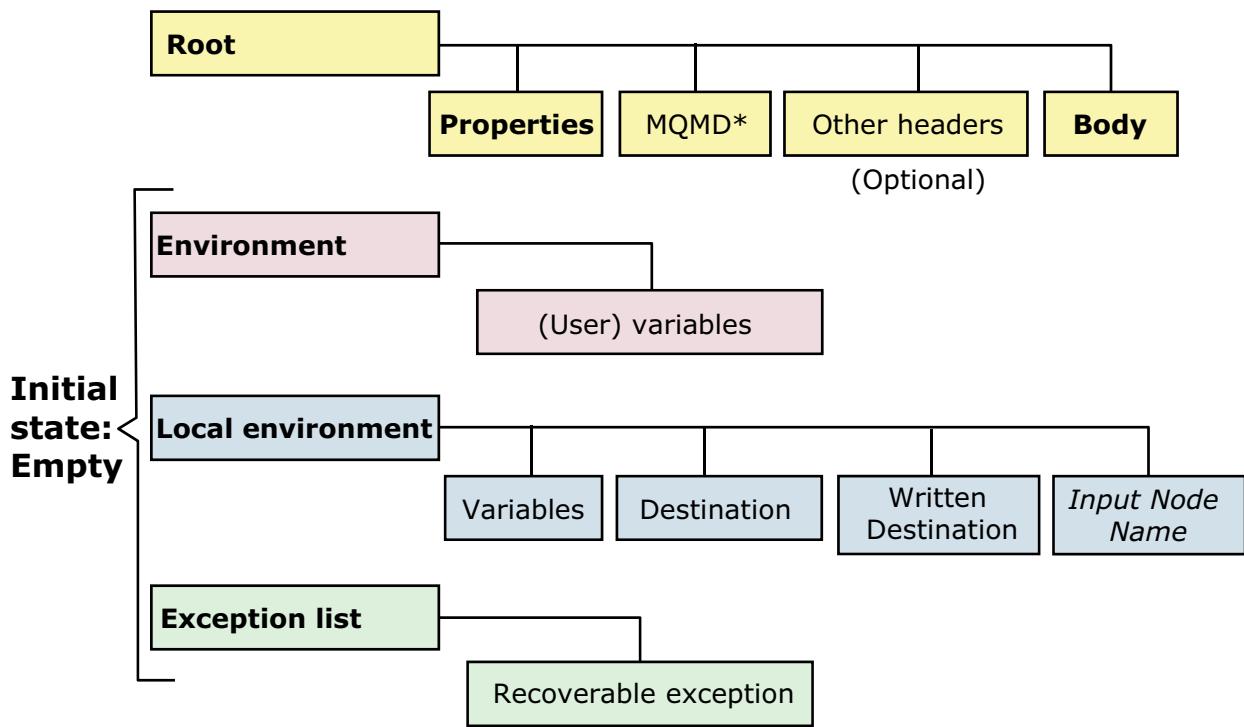
### Notes:

The logical message model is a consistent, convenient way to represent various message formats. Using a logical message model makes the message easier to understand and manipulate.

The physical details are removed and only the important business elements remain. The resulting information is stored in a message tree that is organized in top-down fashion. It represents both the data and any parent-child relationships that the data elements have.

## IBM Integration Bus message assembly

### Parent layers



© Copyright IBM Corporation 2015

Figure 5-4. IBM Integration Bus message assembly

WM666 / ZM6661.0

### Notes:

In a previous unit, you were introduced to the concepts of the logical message tree and message parsing. This unit begins with a more in-depth look at logical and physical messages.

When an input node receives a message, a message assembly of four different message trees is created. While the parsed message is written into the Root tree, the other three trees are initially empty. The trees can be populated:

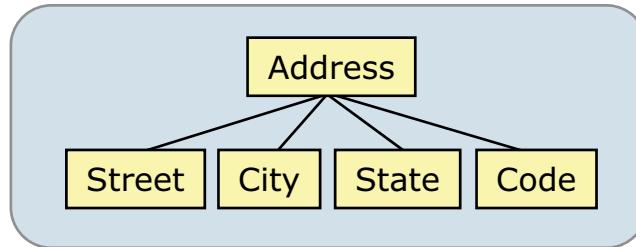
- By certain processing nodes (the LocalEnvironment)
- By the integration node when an exception occurs (ExceptionList)
- By ESQL or Java statements in JavaCompute, Compute, Filter, or Database nodes (all trees)

Root and certain parts of LocalEnvironment can also be modified by using a Mapping node.

## Example data representations

```
<Address>
  <Street></Street>
  <City></City>
  <State></State>
  <Code></Code>
</Address>
```

```
class Address
{
    public String street;
    public String city;
    public String state;
    public int code;
}
```



```
01 ADDRESS.
  02 STREET PIC X(40).
  02 CITY   PIC X(40).
  02 STATE  PIC X(20).
  02 CODE   PIC 9(5).
```

```
struct Address
{
    char street[40];
    char city[40];
    char state[20];
    int code;
}
```

© Copyright IBM Corporation 2015

Figure 5-5. Example data representations

WM666 / ZM6661.0

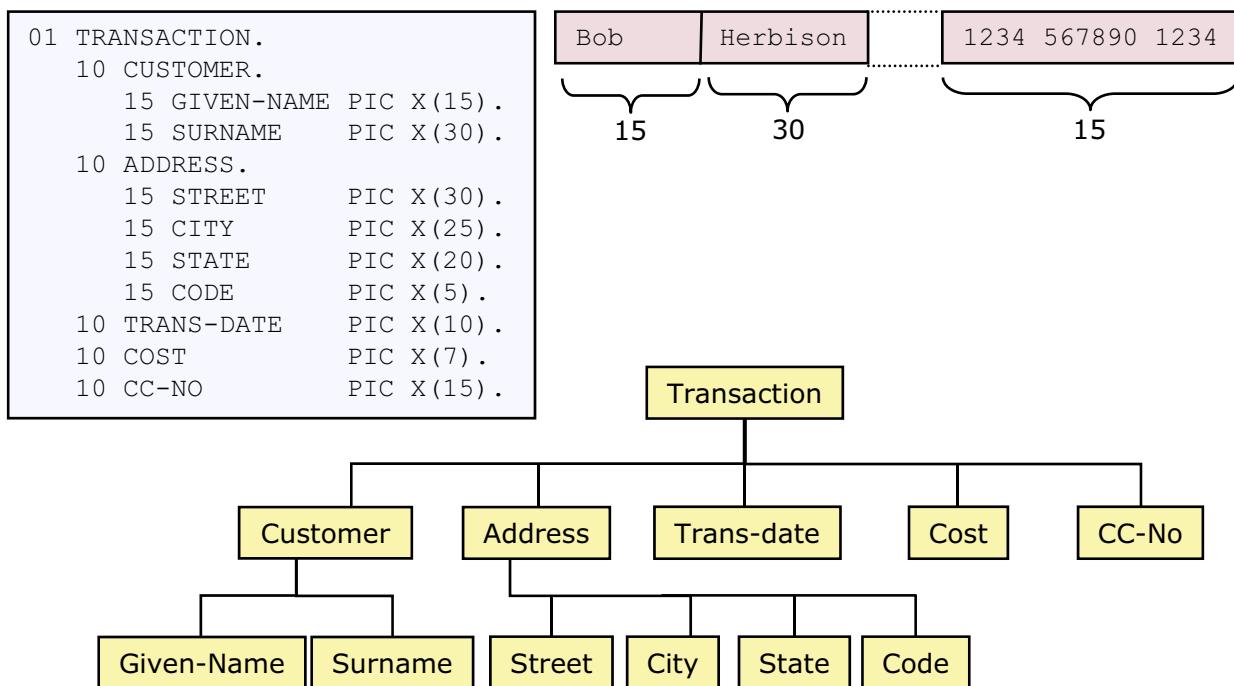
### Notes:

This figure shows several data structure representations of the same logical business data entity, an address. As shown in the figure, there is a wide array of possibilities, but they all represent the same piece of business data.

A simple tree structure provides a common abstracted logical view of any of these data structures. This simple tree, which is organized from top to bottom and left to right, can represent any of these structures. In fact, any data that must be processed in a message flow can be represented in a tree structure.

This example is simple; the tree has only two levels and not much structure to it in terms of parent and child relationships. However, a much more elaborate tree can be built.

## Logical message model example



© Copyright IBM Corporation 2015

Figure 5-6. Logical message model example

WM666 / ZM6661.0

### Notes:

In the example in the figure, a COBOL Copybook represents a fixed-format message. To the right of the Copybook, is a physical instantiation of that message.

Information from both of these entities is used to build the message tree that is shown below them. Each leaf node in the tree contains both a label and a value. For example, the `Surname` node contains a value of `Herbison`. The data types that are supported in a leaf node come from a limited and simple set such as string, integer, and float.

You do not need to be concerned with the physical details of the data representations. For example, if it is an integer, you do not specify in the tree that it is a fullword or doubleword integer, or that it is Big-endian or Little-endian encoded. Those aspects of the physical representation of the data do not have anything to do with the business content of the data.

Element values in the tree are stored in Unicode. Unicode facilitates and simplifies code conversions in IBM Integration Bus. Elements are not stored by using EBCDIC or ASCII, but rather by using this “physical neutral” means instead. The physical details of the data are removed allowing the developer to concentrate on the business data, not its physical representation.



## ResetContentDescriptor node

- Request a different parser to parse the message
- Associates the new parser information with the input message bit stream
- Does not change the message content
- Does not convert the message from one format to another
- The node is in the **Construction** drawer of the Message Flow editor palette

© Copyright IBM Corporation 2015

Figure 5-7. ResetContentDescriptor node

WM666 / ZM6661.0

### Notes:

The ResetContentDescriptor node associates a new parser with the input message bit stream.

For an example, the format of an incoming message might be unknown when it enters a message flow, so the BLOB parser is selected on the input node. Later on in the message flow, it might be necessary to refer to some message content. You can use the ResetContentDescriptor node to set the domain to DFDL or XMLNSC so that a subsequent node in the message flow can reference the message content.

## Message routing nodes

- **Filter** node routes a message according to message content by evaluating an ESQL filter expression in an ESQL module
- **Route** node directs the messages down various paths by evaluating XPath filter expressions
- **Label** node processes a message a **RouteToLabel** node propagates to dynamically determine the message route
- **FlowOrder** node forces a prescribed order of processing of the messages that this node propagates

© Copyright IBM Corporation 2015

Figure 5-8. Message routing nodes

WM666 / ZM6661.0

### Notes:

The Filter, Route, Label, and RouteToLabel nodes are used to route a message to a specific path in the message flow. Each of these nodes is described in more detail next.

When you connect message flow nodes together, the integration node determines the order in which they are processed. The problem is that if you connected more than one node or sequence of nodes to a single output terminal, you cannot predict whether one sequence is processed before another for a message. If the order of processing is important in your message flow, use the FlowOrder node to force a prescribed order of processing of the messages that this node propagates.

## Filter node



- Evaluates ESQL expressions in an ESQL module
  - SQL WHERE clause
- Routes a message through the appropriate output terminal
  - TRUE
  - FALSE
  - UNKNOWN (the test field is not present)
  - FAILURE (a runtime error occurs during node processing)
- Incoming message is not modified
- Message is lost if the terminal is unwired (True, False, and Unknown)
- Can access a relational database

© Copyright IBM Corporation 2015

Figure 5-9. Filter node

WM666 / ZM6661.0

### Notes:

The Filter node can be used to test fields in an incoming message.

The Filter node has one input terminal and four output terminals. The original incoming message is routed to one of the four output terminals.

The Filter node works by running ESQL that you write. The ESQL must return a Boolean value of “True”, “False”, or “Unknown”. The message is routed as follows:

- If the return value evaluates to “True”, the message is propagated out the **True** terminal.
- If the return value evaluates to “False”, the message is propagated out the **False** terminal.
- If the return value evaluates to an unknown value (a value that is not a valid Boolean), the message is propagated out the **Unknown** terminal. If you code RETURN without an expression (RETURN;) or with a null expression, the node propagates the message to the **Unknown** terminal.
- If an error occurs while the node is running, the message is propagated out the **Failure** terminal, if it is wired; otherwise, default error handling takes place.

Although you can write ESQL that changes the incoming message, any changes you make are discarded when then the message is propagated.

Remember that the message is always propagated through one of the output terminals, even if those terminals are unwired. As you learned earlier, messages can be lost if the message is propagated through an unwired terminal. The message that is lost does not apply to the **Failure** terminal, which is handled the same as the **Failure** terminal on any other message flow node.

The Filter node can access a relational database table. The values that are retrieved from the database can be part of the ESQL code, which can be used to set the Boolean return value. Database access is covered in a subsequent unit. When you are accessing a database and an error occurs, the message is propagated through the **Unknown** terminal.

For more sophisticated branching and routing capabilities, use the Route node (or RouteToLabel and Label nodes).

## ESQL

- Extends the constructs of the SQL language to define the behavior of nodes in a message flow
  - Test, calculate, and manipulate fields in a logical message
  - Reference message header fields
  - Change field properties
- Based on standard SQL 3
- See “ESQL reference” topic in the IBM Knowledge Center for IBM Integration Bus

### Examples:

```
RETURN Body.Msg.Name = 'IBM';
SET OutputRoot.XMLNSC.Msg.Name = 'IBM';
SET OutputRoot.XMLNSC.Msg.Name = InputBody."First.Name";
```

© Copyright IBM Corporation 2015

Figure 5-10. ESQL

WM666 / ZM6661.0

### Notes:

ESQL is a way of writing “code” for testing and manipulating fields in messages within IBM Integration Bus. It is based on standard SQL 3 and is extended with facilities to work with logical message trees.

In the example in the figure, the first line of ESQL tests the **Msg.Name** field in the message body for the constant value “IBM.” Such a statement would be used in a Filter node, where message content cannot be changed.

The second and third lines are samples of Compute node ESQL. Since message data can be changed in a Compute node, the field name but distinguish between **InputRoot** and **OutputRoot**. The second line sets an output field that is called **Name** in an XML message to “IBM.” The third line sets the **Name** value to whatever value is in the “**First.Name**” field in the incoming message.

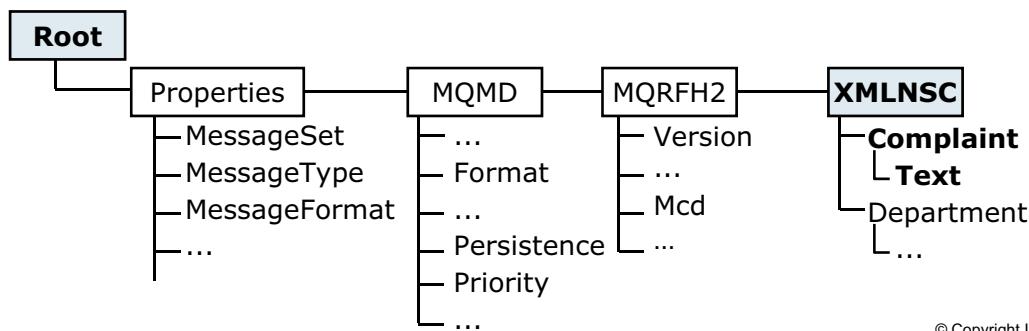
The input field name is “**First.Name**” in the third example. The dot (.) is part of the name and does not denote field hierarchy, so you must enclose the field name with double quotation marks.

Including a period as part of a field name is not good programming practice, but might occur when an existing message structure is imported into Integration Bus.

## Accessing message trees with ESQL correlation names

- Pointer into a message tree
  - Root, InputRoot, OutputRoot
  - Body (=Root.\*[<]), InputBody (=InputRoot.\*[<])
  - LocalEnvironment, Environment, ExceptionList
  - Database
  - Other methods
- Used in ESQL code, patterns in a Trace node, data location parameters in an MQGet node, and others

Example: Root.XMLNSC.Complaint.Text



© Copyright IBM Corporation 2015

Figure 5-11. Accessing message trees with ESQL correlation names

WM666 / ZM6661.0

### Notes:

In ESQL code and in the Trace node, correlation names such as **Body**, **Database**, and **Root** are used as pointers into the message tree (and into the database, as in normal SQL joins).

**Body** is a shortcut for “the last child of **Root**.” The body is where the original incoming message is stored.

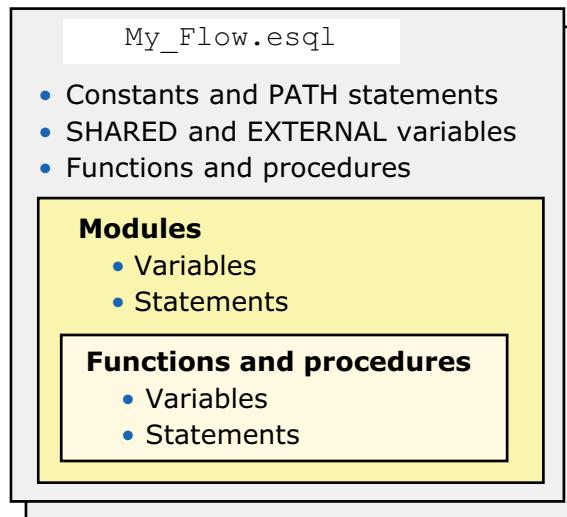


#### Information

Appendix A of this Student Guide contains more examples of ESQL statements.

## ESQL module

- Must begin with the **CREATE node\_type MODULE** statement and end with an **END MODULE** statement
- Name of file must match name that is specified in the **Filter expression** node property
- ESQL file must have a suffix of **.esql**
- Within the ESQL file, the value of the corresponding property of the message flow node determines the name of the module



© Copyright IBM Corporation 2015

Figure 5-12. ESQL module

WM666 / ZM6661.0

### Notes:

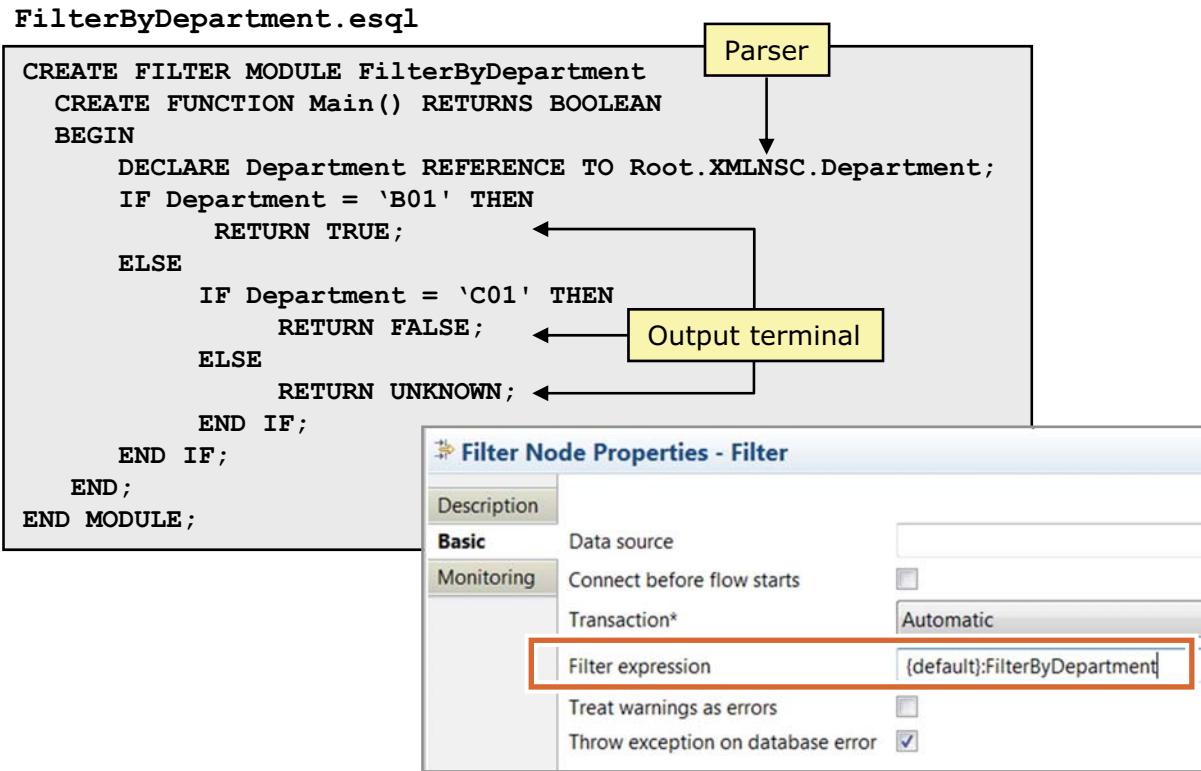
In an IBM Integration Bus application, the ESQL code is stored in a separate file. A name identifies each ESQL module that is defined in an ESQL file in the same integration node schema as the node that references it. There must be one module of the correct type for each corresponding node. For example, the Filter node uses a FILTER module.

The name of the ESQL file must also match the name that is referenced in the node.

If you want to reuse ESQL constants, functions, or procedures, you must declare them at the integration node schema level. You can then reference the ESQL objects from any resource within that integration node schema, in either the same or in another project. If you want to use this technique, you must either fully qualify the resource or include a PATH statement that sets the qualifier. The PATH statement must be coded in the same ESQL file, but not within any module.

```
PATH MySchema2;
CALL MySchema2.ProcX(p);
```

## Sample ESQL for Filter node



© Copyright IBM Corporation 2015

Figure 5-13. Sample ESQL for Filter node

WM666 / ZM6661.0

### Notes:

In this example, the Filter expression is `FilterByDepartment`. It matches the ESQL file name and the module name in the ESQL code.

The code in this example filters the incoming message by the value of the `Department` field. It routes the message to either the **True**, **False**, or **Unknown** terminal based on that value.

The `DECLARE` statement defines a local variable that is named `Department` points to a value in the logical message tree. In the logical message path `Root.XMLNSC.Department`, `XMLNSC` identifies the parser to use to parse the message.



## ESQL editor

- Statement syntax validation
  - Missing punctuation, incorrectly used functions and keywords, and more
  - Project can be deployed with warnings, but not errors
- Semantic validation
  - Unresolved identifiers
  - Message or database reference mismatch
  - Deprecated keywords
- Can set validation preferences
- Content assist
  - To start, press Ctrl+Space or click **File > Edit**
  - Right-click in the editor to access more editor options

© Copyright IBM Corporation 2015

Figure 5-14. ESQL editor

WM666 / ZM6661.0

### Notes:

The ESQL editor can help you get started. It provides statement syntax and semantic validation.



## ESQL editor options

- **Undo** to undo a change that you made to the ESQL file
- **Revert File** to undo all the changes since the last time file was saved
- **Comment** to change a line of ESQL code into a comment
- **Uncomment** to change a comment into a line of ESQL code
- **Format** to formats all selected lines of code
- **Organize Schema Paths** adds a broker schema that contains procedures or functions that the ESQL file calls to the PATH statement
- **Add Schema Path** adds a schema to the PATH statement when the code includes a call to a procedure or function that is in a different broker schema

© Copyright IBM Corporation 2015

Figure 5-15. ESQL editor options

WM666 / ZM6661.0

### Notes:

The figure lists some of the ESQL editor options. Right-click in the ESQL editor view to access the editor options.

Click **Organize Schema Paths** to automatically add a broker schema that contains procedures or function that is called by the ESQL to the PATH statement. This function scans the ESQL file for instances of procedures or function that is in schemas that are not already fully qualified in the file.

Click **Add Schema Path** when you code a call to a procedure or function that is in a different broker schema to paths. This schema is added to the PATH statement. Ensure that the cursor is on the name of the procedure you are calling.



## ESQL content assist and validation

- Create a module for a node: flowName\_nodeName
- Associate a node with an existing module

CREATE COMPUTE MODULE Module1

```

CREATE FUNCTION Main () RETURNS BOOLEAN
BEGIN
    DECLARE number CHARACTER;
    SET myNumber = InputRoot.MRM.items.item[1].partNum;
    SET OutputRoot.MRM.items[1].ItemNo = number;
    -- CALL eed.schema2.setFirstItemNumber(InputRoot.MRM.
    RETURN TRUE;
END;

```

Tasks (Filter matched 1 of 32 items)

C.	Description	Resource	In Folder	Locati...
!	Identifier myNumber cannot be resolved.	File1.e...	EsqlEditorDemo/e...	line 8

© Copyright IBM Corporation 2015

Figure 5-16. ESQL content assist and validation

WM666 / ZM6661.0

### Notes:

IBM Integration Bus contains ESQL content assist to help you create syntactically valid ESQL statements. The figure shows an example of the ESQL for a Compute node.

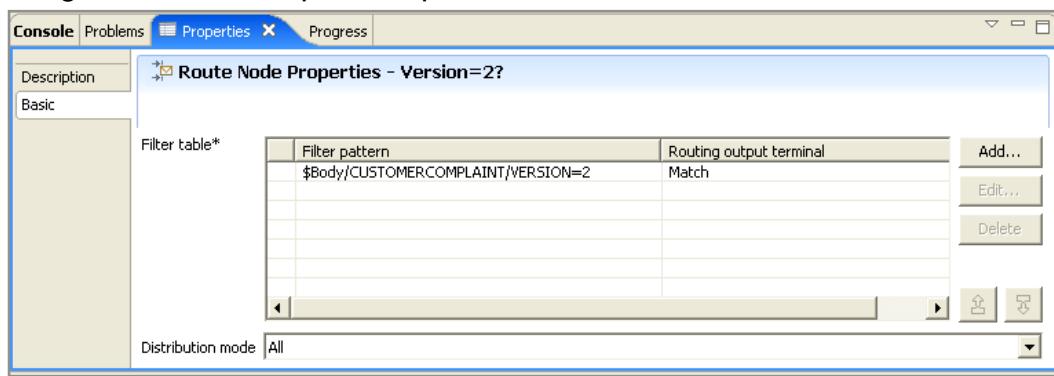
An ESQL module can be deployed if it contains warnings, but not if it contains errors. The error decorator in the content assist margin indicates that a warning (yellow triangle) or an error (red exclamation point) was detected on the marked line. Review the **Problems** tab for details about the problems the decorator represents.



## Route node



- XPath expressions direct messages to output terminals
  - Message is routed to **Match** when the filter evaluates to “true”
  - Message is routed to **Default** if none of the filter expressions are true
  - Message is routed to **Failure** if a runtime error occurs (if wired)
- Can add more terminals
- Node propagates messages to the terminals in the order in which they appear in the **Filter table**
- **Distribution mode (First or All)** determines the routing behavior when an inbound message matches multiple filter patterns



© Copyright IBM Corporation 2015

Figure 5-17. Route node

WM666 / ZM6661.0

### Notes:

The Route node directs messages through various paths that are based on filter criteria that you configure in the node properties. The figure shows an example of a Route node filter.

The Route node has one input terminal and by default, three output terminals: **Match**, **Default**, and **Failure**. You can create extra output terminals to meet your routing needs.

At design time, you populate a filter table with XPath expressions and corresponding output terminal names in the **Properties** view. At run time, the table is scanned, and if an XPath expression evaluates to true, the logical message is routed to the named terminal.

If you set the **Distribution mode** property to **First** at design time, the Route node ends after encountering the first match in the table. If **Distribution mode** is set to **All**, the remainder of the table is scanned for more matches.

All XPath expressions in the filter pattern must start with \$Root, \$Body, \$Properties, \$LocalEnvironment, \$DestinationList, \$ExceptionList, or \$Environment. Expressions are evaluated in the order in which they are displayed in the table. To improve performance, specify the expressions that are satisfied most frequently at the top of the filter table. Typically, you specify a unique terminal name for each XPath expression.

Like the Filter node, the Route node does not alter the incoming message. The message that is propagated from the node (on whichever terminals) is the same message that was received on the input terminal.

## Accessing message trees with XPath

- Select an element from an XML tree by using a declarative expression
  - No need for explicit navigation: /document/chapter/title
  - Returns a nodeset of syntax elements
- Can select multiple elements  
**Example:** /library/books/book returns all book elements in the /library/book path
- Thirteen XPath axes allow a tree to be traversed in different ways
  - Self, parent, child, ancestor, descendant, attribute, namespace, and more
  - Abbreviated forms are typically used . . . // @ \*
- XPath expressions support selection predicates for search criteria  
**Example:** book [author='Steven Bawking']
- XPath functions for trivial expression evaluations  
**Example:** count (/sum/library/books/book)
- Broad range of retrievals; from simple to SQL SELECT-style expressions

© Copyright IBM Corporation 2015

Figure 5-18. Accessing message trees with XPath

WM666 / ZM6661.0

### Notes:

An XPath expression can be used to search through an XML document, and extract information from the nodes (any part of the document, such as an element or attribute) in it. Although XPath is designed for XML documents, it can be applied to any tree structure. Within the integration node, it is used to select elements within the IBM Integration Bus logical message model regardless of the format of the bit stream. Several nodes use XPath to reference and manipulate elements in the message tree.

Integrated XPath support for scalar and complex properties gives you the ability to start the XPath builder by using Content Assist for the scalar property or the column in the complex property.

There is a set of preinstalled set of XPath variables that are commonly used in the Application Development perspective, such as \$Body and \$Environment. Double-clicking an element in the message tree, builds an XPath expression up to the element.

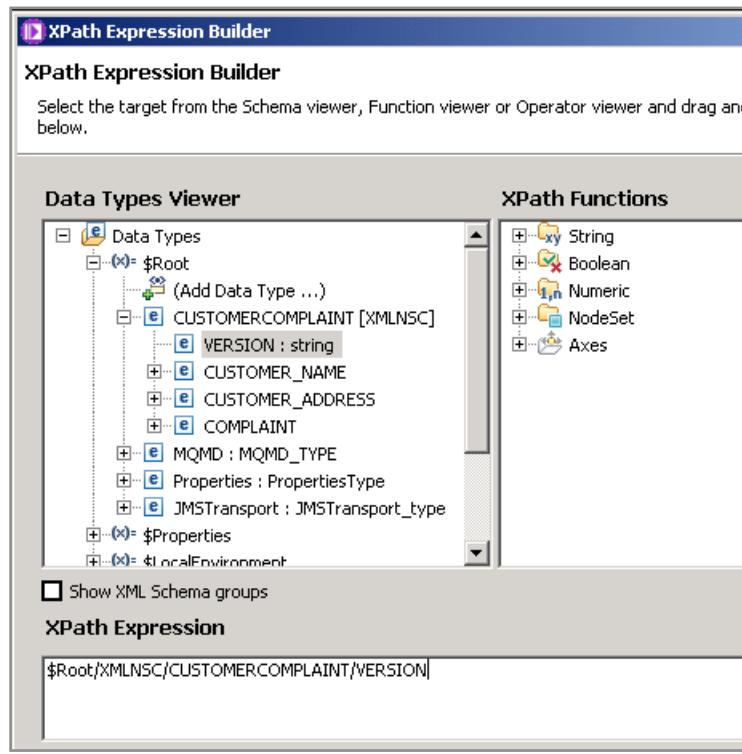
You can find a full XPath reference at <http://www.w3.org/TR/xpath/>

To complete a tutorial on XPath, see <http://www.w3schools.com>xpath/>



## XPath Expression Builder

- Start the XPath Expression Builder from most property fields that support, or expect, XPath expressions
  - Data Types Viewer** shows the different schema types, elements, and attributes that you can use within the XPath
  - XPath Functions** pane shows four main top-level categories, which are: String, Boolean, Numeric, Nodeset
  - Operators** pane shows a list of all of the available operators that you can use within the XPath expression



© Copyright IBM Corporation 2015

Figure 5-19. XPath Expression Builder

WM666 / ZM6661.0

### Notes:

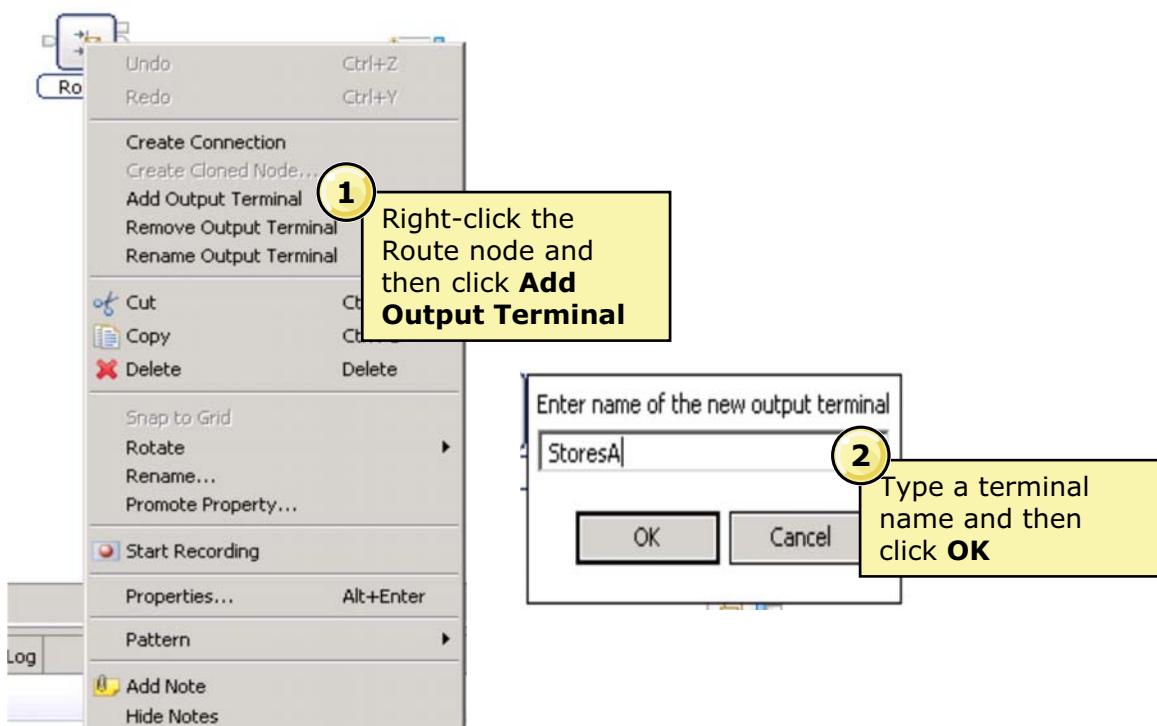
The XPath Expression Builder helps you to construct message processing expressions in either XPath or ESQL.

You can start the XPath Expression Builder from most property fields that support, or expect, XPath expressions as a value that can be entered into the field.

There are three main views when functions are supported. Whether a view is displayed, and what is displayed in it, depends on what type of property editor you used to start the dialog box. It also depends on its tailored settings. The operators that are supported can change as can the list of applicable variables.



## Adding terminals to the Route node



© Copyright IBM Corporation 2015

Figure 5-20. Adding terminals to the Route node

WM666 / ZM6661.0

### Notes:

You can add custom terminals to the Route node.

## RouteToLabel and Label nodes



- Transfers control within a message flow that is based on the content of the LocalEnvironment tree
- RouteToLabel node
  - Interrogates LocalEnvironment.DestinationData about the message to determine the identifier of the Label node to which the message must be routed
  - Can use a Compute, JavaCompute, or Mapping node to set the LocalEnvironment.DestinationData
- Label node
  - Provides a target for a routing decision; it does not alter the message that it receives
  - Can be used with a SOAPExtract node or as the target of an ESQL PROPAGATE statement

© Copyright IBM Corporation 2015

Figure 5-21. RouteToLabel and Label nodes

WM666 / ZM6661.0

### Notes:

You can use the RouteToLabel and Label nodes to dynamically change the routing of messages within a message flow.

RouteToLabel queries the LocalEnvironment.DestinationData tree in the message assembly (the exact location in the tree depends on your application code). The value that it retrieves from the appropriate field is the **Label** property of a corresponding Label node.

The RouteToLabel node sends the message tree to the Label node. The Label node is not a processing node; it acts only as an entry point into a branch of the message flow. Often this branch is implemented by using a subflow.

You set the value of the DestinationData with the use of a compute-type node. If the value of the intended destination field is not set when the message reaches the RouteToLabel node, a runtime error occurs.

You can also use a Label node without a RouteToLabel node. You can send a message flow to a Label node by using the PROPAGATE statement in a Compute node.



## FlowOrder node

- Two output terminals control the order in which subsequent nodes process the message
    - **First** terminal
    - **Second** terminal
  - Message that is propagated out both terminals is identical
1. Input message is passed to node connected to **First** terminal
  2. All processing that is defined by all subsequent nodes in this sequence is completed before control returns to the FlowOrder node
  3. Input message is propagated to the node connected to the **Second** terminal

© Copyright IBM Corporation 2015

Figure 5-22. FlowOrder node

WM666 / ZM6661.0

### Notes:

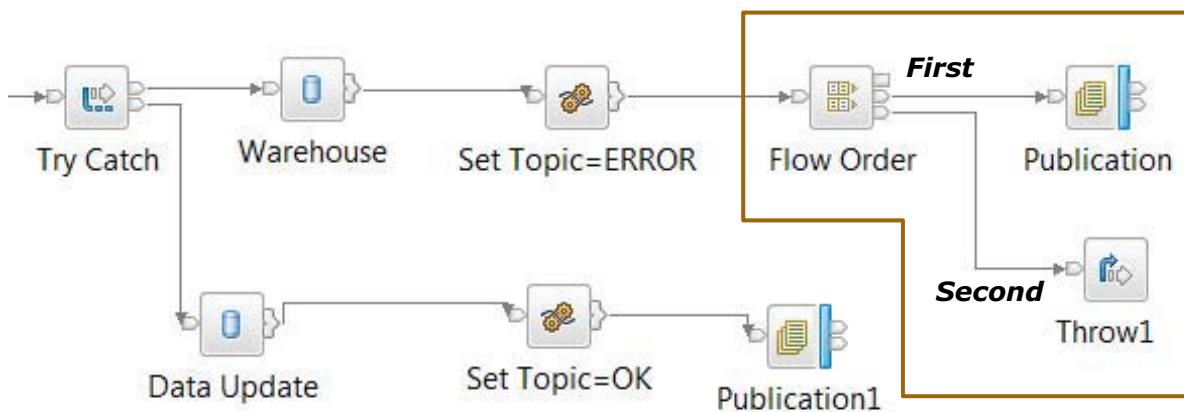
If the order of processing is important in your message flow, use the FlowOrder node to force a prescribed order of processing of the messages that this node propagates.

The FlowOrder node has two output terminals that you can connect to control the order in which subsequent nodes process the message. When you connect a node or sequence of nodes to the First terminal, the input message is passed to the next node. All processing by all subsequent nodes in this sequence is completed before control returns to the FlowOrder node. The input message is then propagated to the next node in the sequence of nodes that are connected to the Second terminal.

The message that is passed to both sequences of nodes is identical. It is always the message that the FlowOrder node receives as input.

The FlowOrder node provides no other processing on the input message; it is used only for imposing order on subsequent processing.

## FlowOrder node example



1. Input message is passed to Publication node that is connected to **First** terminal
2. After processing on Publication node is complete, control returns to the FlowOrder node
3. Input message is propagated to Throw1 node that is connected to the **Second** terminal

© Copyright IBM Corporation 2015

Figure 5-23. FlowOrder node example

WM666 / ZM6661.0

### Notes:

This figure shows an example of how the FlowOrder node can be used to control flow order processing.

In the example, a Compute node creates an error topic. The FlowOrder node first passes the message to the Publication node, which is connected to the First terminal. When the Publication node completes, the FlowOrder node sends the same input message to the Throw node, which is connected to the Second terminal.

## Subflows

- Message flows with Input or Output nodes (or both)
  - Subflow that is defined in a `.subflow` file
  - Subflow that is defined in a `.msgflow` file
- Provide a common sequence of actions to be used by several message flows, applications, or integration services
- Used like normal processing nodes (a compound node)
  - Drag (embed) into the main flow
  - Wire terminals
- Provide reuse of program code
- Support multiple nesting levels
- Node properties can be promoted and then set from a higher message flow level

© Copyright IBM Corporation 2015

Figure 5-24. Subflows

WM666 / ZM6661.0

### Notes:

A subflow is a message flow that can be started from another flow. It is most often used for a sequence of message processing steps that must be done on a frequent basis within a message flow or across multiple flows. A subflow is similar in function to a programming macro: after you write it, you can embed it in other message flows. You can also use a subflow to simplify the main flow, by breaking out a portion of the processing logic into its own subflow, and then calling that subflow from the main flow.

You construct a subflow the same way that you do a main flow, by adding, configuring, and wiring message flow nodes. A subflow differs from a main flow in that it must begin with an **In** node, and end with an **Out** node. These nodes are used to pass control from and return control to the main flow.

You can develop subflows in two ways:

- You create a subflow in its own `.subflow` file.
- You can create a subflow that is part of a `.msgflow` (or main flow) file.

Both of these methods for subflow development are described later in this unit.

## Message flow nodes that are used with subflows

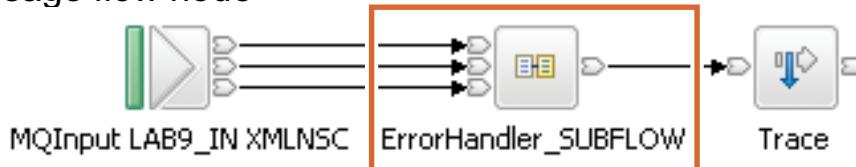
- Every subflow must begin with at least one input node



- Every subflow must end with at least one output node



- When a subflow is referenced in a main flow, it is shown as a single message flow node



© Copyright IBM Corporation 2015

Figure 5-25. Message flow nodes that are used with subflows

WM666 / ZM6661.0

### Notes:

When you create a subflow, it is represented as a single node that you add to the canvas at whatever point you want to start it.

Every Input node that is defined in the subflow represents an **In** terminal on the Subflow node. Every Output node that is defined in the subflow represents an **Out** terminal on the Subflow node. You can use these terminals to connect the subflow node to other nodes in the parent flow.

The figure indicates that you must have at least one Input node and one Output node per subflow. Depending on your processing requirements, you can have more than one of each. Multiple input nodes correspond to multiple entry points into, and multiple return points from the subflow.

- If you want to use a subflow as the first node in your message flow, you need to add at least one Input node, such as the MQInput node, and one generic Output node.
- If you want to use a subflow in the middle of a message flow, you need to add at least one generic Input node. You need to add a generic Output node if you want to connect more nodes in the message flow after you add the subflow.
- If you want to use a subflow as the last node in a message flow, you need to add at least one generic Input node, and at least one Output node, such as the MQOutput node.

When you include the subflow in a “main” message flow, this subflow icon shows a terminal for each Input node that you include in the subflow. The name of the terminal (which you see when you hover the mouse cursor over it) matches the name of that instance of the Input node. Give your Input nodes meaningful names that you can recognize easily when you use their corresponding terminal on the subflow node in your message flow.

## Subflow that is defined in a .subflow file

- Can be deployed in any of the following ways:
  - To the same integration server, but separately from any of the message flows that use this subflow
  - As part of an application or integration service
- You cannot use the following nodes in this type of subflow:
  - Nodes that represent subflows that are defined in `.msgflow` files
  - User-defined nodes that are created from subflows that are defined in `.msgflow` files
- If you create a BAR file that contains both ESQL code and a subflow that is defined in a `.subflow` file, you cannot include the ESQL code directly in compiled message flow files



When you develop new integration solutions in IBM Integration Bus, create subflows as `.subflow` files.

© Copyright IBM Corporation 2015

Figure 5-26. Subflow that is defined in a `.subflow` file

WM666 / ZM6661.0

### Notes:

A subflow that is defined in a `.subflow` file can be deployed in any of the following ways:

- The subflow is deployed separately from any of the message flows that use this subflow. The subflow and the message flows that include this subflow must be deployed in the same integration server.
- The subflow can be deployed directly to an integration server or as part of a library. If you update this type of subflow and redeploy it, all message flows that use this subflow, and are not part of an application or the integration service, are automatically updated. You do not need to redeploy these message flows. If you update a subflow in a shared library and redeploy it, all message flows in an application or integration service that use the subflow are updated automatically.
- The subflow is deployed as part of an application or integration service.

You cannot use the following nodes in this type of subflow:

- Nodes that represent subflows that are defined in `.msgflow` files
- User-defined nodes that are created from subflows that are defined in `.msgflow` files

If you create a BAR file that contains both ESQL code and a subflow that is defined in a .subflow file, you cannot include the ESQL code directly in compiled message flow files.

## Subflow that is defined in a .msgflow file

- Embedded inside a parent message flow that uses it when the message flow is placed in a BAR file
- Must be deployed to an integration server with the message flow in which it is used
- If subflow is updated, you must redeploy all message flows that use the subflow for the changes to be recognized
- Can be packaged as a user-defined node



When possible, convert any existing subflows that were created as .msgflow files into .subflow files

© Copyright IBM Corporation 2015

Figure 5-27. Subflow that is defined in a .msgflow file

WM666 / ZM6661.0

### Notes:

A subflow that is defined in a .msgflow file is embedded inside any parent message flows that use it when the message flow is placed in a BAR file.

This type of subflow can be deployed only to an integration server with the message flow in which it is used. If you update this type of subflow, you must redeploy all message flows that use the subflow for the changes to be used.

This type of subflow can be packaged as a user-defined node.



#### Note

WebSphere Message Broker releases earlier than version 8 only supported subflows that are created as a .msgflow file. If you plan to develop new integration solutions in IBM Integration Bus, you should create subflows that are created as a .subflow file. You should convert your subflows that are created as .msgflow files into subflows that are created as .subflow files.

## Subflows in shared libraries

- Can contain the standard subflow input and output nodes
- Can also contain any input node
  - Subflows get instantiated when used within a message flow only
  - Subflows do not get their own thread pool, and are unable to run outside of an application
- Must be placed in a broker schema
- Can be placed in multiple broker schemas
- Can be dropped into a message flow or subflow in an application in the IBM Integration Toolkit

© Copyright IBM Corporation 2015

Figure 5-28. Subflows in shared libraries

WM666 / ZM6661.0

### Notes:

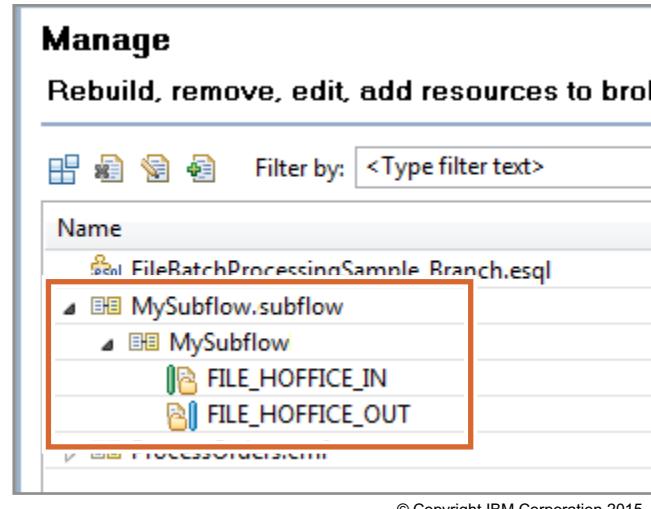
Subflows in shared libraries cannot be in the default broker schema.

Subflows for a broker schema must not be in two or more shared libraries that a single application or shared library references.

WebSphere Education

## Subflow properties in the BAR file

- Can configure overrides for subflows
- Use common property values or modify values for each subflow instance
- Configurable any time the subflow is deployed:
  - When deploying main message flows and applications
  - When deploying just the subflow
- Compatible with existing tools:
  - BAR file editor
  - `mqsiapplybaroverride` command
  - API applications



© Copyright IBM Corporation 2015

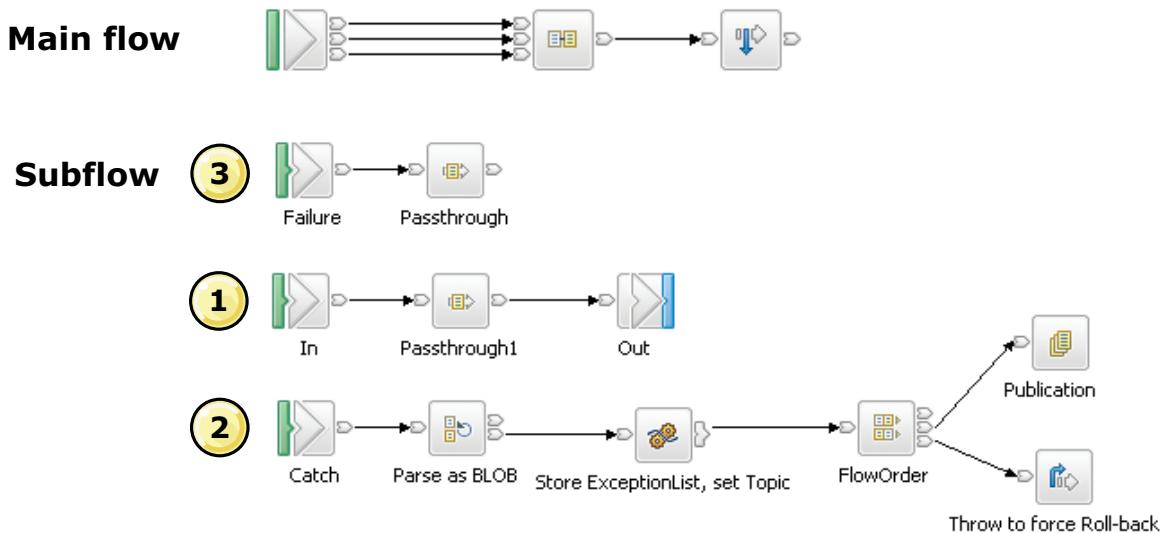
Figure 5-29. Subflow properties in the BAR file

WM666 / ZM6661.0

### Notes:

A subflow can be deployed independently of the message flows that use the subflow. When you deploy a subflow independently, it is similar to a message flow and has its own set of properties. It can be added to the BAR file and modified by using any of the standard BAR file tools such as the BAR file editor and the `mqsiapplybaroverride` command.

## Example: Generic error handler subflow



1. The original message is published
2. Support persons or monitoring applications subscribe to relevant topics
3. Manual error resolution is sometimes necessary

© Copyright IBM Corporation 2015

Figure 5-30. Example: Generic error handler subflow

WM666 / ZM6661.0

### Notes:

This figure shows how you can use a subflow to collect information about run time errors, with the original message.

This example detects any exception, and in a subflow, includes the content of the ExceptionList in the message and publishes the extended message to a special topic. The offending message is removed from input queue and discarded.

Support personnel or monitoring applications can subscribe to the error topic, receive error messages, inspect its ExceptionList, take appropriate actions to resume the error, delete the ExceptionList, and reprocess the message.

This solution is a generic error handler; it works for all message domains. It also handles parsing errors. The only way to achieve this result is to treat the message body as BLOB. So, you reparse the message as BLOB in a ResetContentDescriptor node before doing the other manipulations in the error handler subflow.

## Unit summary

Having completed this unit, you should be able to:

- Describe logical messages and the message assembly, and explain how they are used in IBM Integration Bus application programming
- Use the Filter and Route message processing nodes to examine the contents of a message and alter its flow
- Use the RouteToLabel and Label nodes to dynamically change the routing of messages
- Use the FlowOrder node to control the flow path order in which a message is processed through a message flow
- Create reusable subflows

© Copyright IBM Corporation 2015

Figure 5-31. Unit summary

WM666 / ZM6661.0

### Notes:

## Checkpoint questions

1. True or False: A Subflow must be created in the same project as the main flow that calls it.
2. True or False: A Route node uses a table of ESQL statements to determine which terminals to propagate a message through.

© Copyright IBM Corporation 2015

Figure 5-32. Checkpoint questions

WM666 / ZM6661.0

### Notes:

Write your answers here:

1.

2.

## Checkpoint answers

1. True or False: A Subflow must be created in the same project as the main flow that calls it.

**Answer: False.** A subflow can be created in its own subflow file and deployed separately from the main flows that call it.

2. True or False: A Route node uses a table of ESQL statements to determine which terminals to propagate a message through.

**Answer: False.** The Route node table is composed of XPath statements.

© Copyright IBM Corporation 2015

Figure 5-33. Checkpoint answers

WM666 / ZM6661.0

### Notes:

## Exercise 4



Adding flow control to a message flow application

© Copyright IBM Corporation 2015

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

10.1

Figure 5-34. Exercise 4

WM666 / ZM6661.0

### Notes:



## Exercise objectives

After completing this exercise, you should be able to:

- Use the Route node to control message processing
- Use the XPath Expression Builder to define a filter pattern
- Create custom output terminals on the Route node
- Connect a Failure terminal to an output node to capture exceptions
- Test the message flow by importing messages into the IBM Integration Toolkit Flow exerciser

© Copyright IBM Corporation 2015

Figure 5-35. Exercise objectives

WM666 / ZM6661.0

### Notes:

See the *Student Exercise Guide* for exercise instructions.

# Unit 6. Modeling the data

## What this unit is about

In this unit, you learn about the message modeling options that IBM Integration Bus provides. The unit concentrates on using DFDL to model the data.

## What you should be able to do

After completing this unit, you should be able to:

- Explain the concepts of message models and how they are used to help message transformation
- List the parsers that are available for use within IBM Integration Bus
- Create and modify a DFDL model
- Use importers to create data models
- Choose the appropriate message validation options
- Organize and administer message models
- Reference message models in message flows

## How you will check your progress

- Checkpoint questions
- Lab exercise

## References

IBM Knowledge Center

## Unit objectives

After completing this unit, you should be able to:

- Explain the concepts of message models and how they are used to help message transformation
- List the parsers that are available for use within IBM Integration Bus
- Create and modify a DFDL model
- Use importers to create data models
- Choose the appropriate message validation options
- Organize and administer message models
- Reference message models in message flows

© Copyright IBM Corporation 2015

Figure 6-1. Unit objectives

WM666 / ZM6661.0

### Notes:



## Message modeling

- Most messages require a model at run time to parse the message bit stream from the physical data and serialize to create the physical bit stream from the model
- Modeling accelerates the development of transformations at design time in the IBM Integration Toolkit
  - Provides a source and a target for graphical mappings
  - Assists when editing transformation programs
- Validation can be used for checking message correctness when parsing
- Provides version control when storing artifacts in a central shared repository
- Instant documentation for programmers, analysts, and integration specialists

© Copyright IBM Corporation 2015

Figure 6-2. Message modeling

WM666 / ZM6661.0

### Notes:

Most messages are not self-defining. For example, a C or COBOL program creates a message that is just a stream of binary data. Without a model to interpret it, the data is meaningless. XML is the opposite. It is verbose and self-descriptive and an XML parser can parse any XML document without using a model.

You need a model to ensure that the messages are validated correctly. An XML parser can parse any XML document. However, the parser can verify that the message content is correct only if it has a model to guide it.

A model can improve the development of transformations. For example, graphical mapping from a source message to a target message is not possible without a model. If you are transforming XML documents, you need a model even if the XML parser chooses not to use a model at run time.

Models provide a good way of tracking different versions of your messages. A COBOL programmer would typically create a version of a COBOL Copybook each time a change is made by using a configuration control system. The same principle applies to message models.

## Message model object properties

- Logical model is a format-independent description, similar to default values
- Physical model
  - Controls the parsing and the writing of the object
  - One set of physical properties for the physical format
  - For the MRM message set, one set of physical properties for each format (XML, Text, and Binary)
- Documentation does not affect processing

© Copyright IBM Corporation 2015

Figure 6-3. Message model object properties

WM666 / ZM6661.0

### Notes:

In IBM Integration Bus, logical properties and the physical properties describe the data.

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

The physical properties of an object describe how the object is written. These properties control the parsing and writing of the object. There is one set of physical properties for each physical format in your message model.



## IBM Integration Bus message modeling

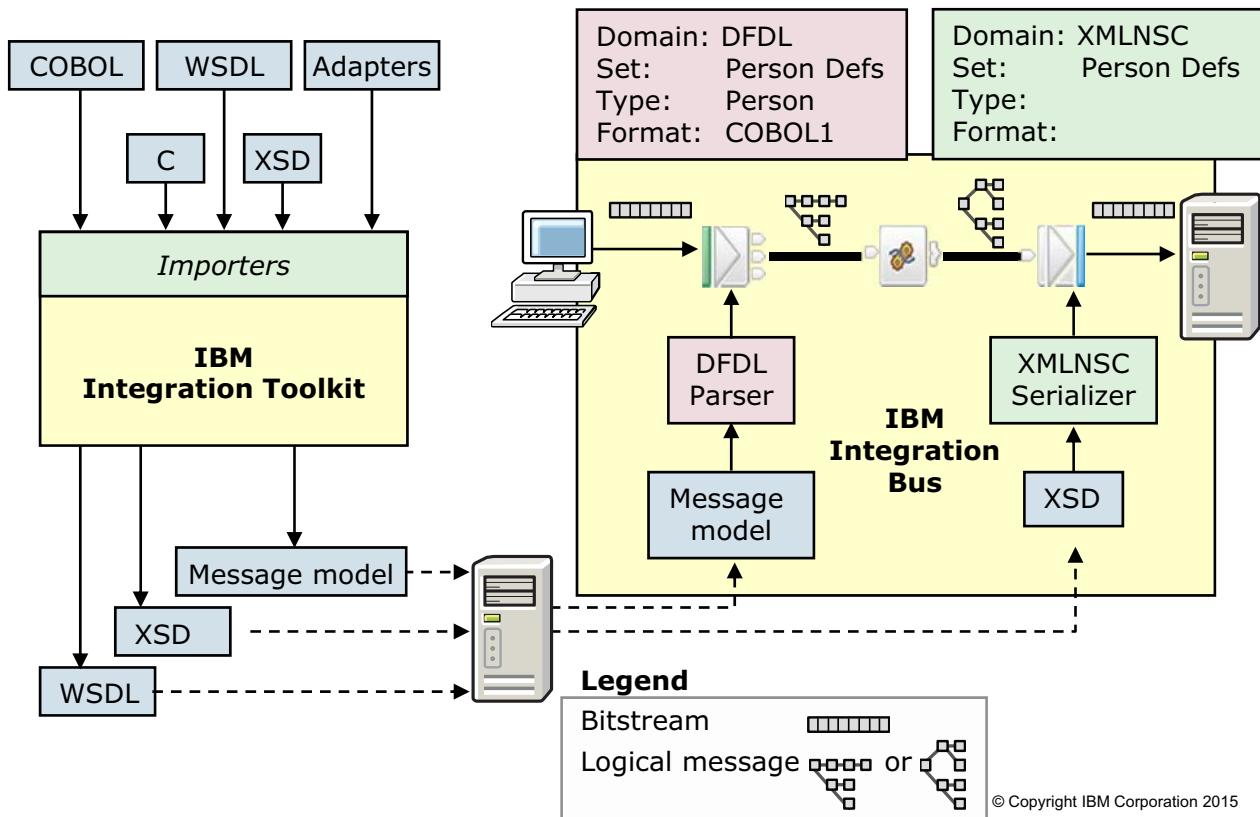


Figure 6-4. IBM Integration Bus message modeling

WM666 / ZM6661.0

### Notes:

IBM Integration Bus uses a message model to model a message format. The message models that IBM Integration Bus uses are all based on World Wide Web Consortium (W3C) XML schema 1.0 specification.

IBM Integration Bus uses *message models* that are based on XML schema to describe the structure of all kinds of message format, including message formats that are not XML.

A *message set* is a logical grouping of your messages and the objects that comprise them. A message set is the original container for message models. In IBM Integration Bus, message model schema files that are contained in applications and libraries are the preferred way to model messages for most data formats. If you must model data formats for use in the MRM or IDOC domains, you must first enable message set development in the IBM Integration Toolkit.

At run time, Integration Bus uses a message model and a parser to create the logical model from the bit stream. At the end of the flow, Integration Bus serializes the data to create the bit stream that the target application requires.

## Parser types

Parsers understand the format of messages

- Programmatic parsers
  - Knowledge of the message format is encoded in a *program*
  - Each message format needs a new parser program
- Descriptive parsers
  - Knowledge of the message format is encoded in a *model*
  - A general-purpose parser program uses the model when parsing
  - Model can be used as-is or generated into code
- IBM Integration Bus uses both approaches
  - XMLNSC, JSON, and BLOB are programmatic parsers
  - DFDL, MRM, and XMLNSC with an XML schema are model-driven parsers

© Copyright IBM Corporation 2015

Figure 6-5. Parser types

WM666 / ZM6661.0

### Notes:

A physical message is nothing more than a string of bits. To use the data, parsers are required to understand how to traverse the message tree from the root and what characteristics the fields have. Essentially, the parser breaks the tree into its elements. If a message can be parsed, its fields no longer present the issues such as data conversion that traditionally causes problems with cross-platform applications. A parser must understand the format of the messages that it is parsing and serializing.

There are two basic approaches to writing a parser:

1. The **Programmatic** approach, in which a specific program is written to parse each different message format. If there are changes, the parser program must be changed, recompiled, relinked, and redeployed.
2. The **Descriptive** approach, in which a single general-purpose model-driven program is written to parse all formats. A model represents each message format.

IBM Integration Bus uses both programmatic and descriptive parsers, depending on the message format.

## Model definition files

- Describe the logical structure of the message
- Describe the physical formats of the message bit stream during transmission
- Can be imported from WSDL, an XML schema, DTD, COBOL, or C (or hand-edited)
- Contains a message model with messages, elements, types, and so on
- Can link DFDL schema files
- Based on XML schema standards

© Copyright IBM Corporation 2015

Figure 6-6. Model definition files

WM666 / ZM6661.0

### Notes:

You typically import application message formats to create and populate message models. You can then edit the logical structure of your messages in a message definition editor.

For example, in the MRM message definition editor, you can create and edit binary, XML, and tag-delimited string physical formats that describe the precise appearance of your message bit stream during transmission.

You can also create an empty message definition file and create your messages by using only the editor.

This unit focuses DFDL message models and the DFDL schema editor that the Integration Toolkit provides.



## Message definition importers

- COBOL, C, XML DTD, XML schema, WSDL, and SCA
- Logical model
  - Importers create types, elements, and attributes
  - Message creation is optional
- Physical information
  - Importers populate selected wire formats
  - Create more wire formats, such as XML, if necessary
  - For C or COBOL, you must specify the target environment and the compiler
  - Other wire formats are filled with default values and can lead to warnings for a custom wire format and tagged-delimited
- Use the Message Definition wizard or the `mqsicreatemsgdefs` command

© Copyright IBM Corporation 2015

Figure 6-7. Message definition importers

WM666 / ZM6661.0

### Notes:

One way to create the message model is to import existing application message formats. You can import the following message formats into a message model: XML DTD files, XML schema files, C header files, COBOL Copybooks, and WSDL files.

When you import one of these formats, a message model is created, consisting of the elements, attributes, groups, and types that are needed to represent the message format. You choose the name of the message model file; if it exists, the content is deleted and re-created as part of the import operation. The new message model that is created consists of both logical and physical information. Appropriate physical formats are created if they do not exist in the message model at the time of the import.

An imported message model can be modified and customized in the IBM Integration Toolkit.

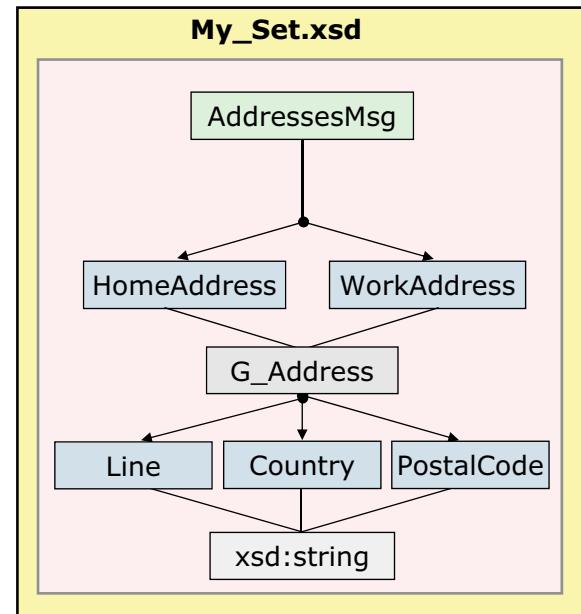
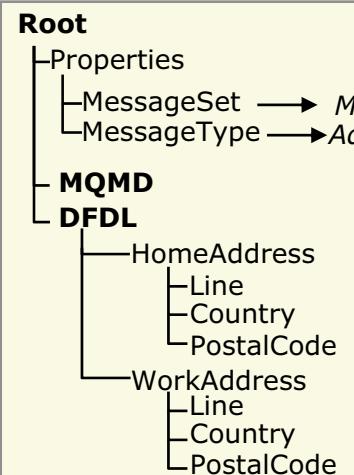
You import a message by using either the Message Definition wizard or the `mqsicreatemsgdefs` command. The command creates message definition files (\*.mxsd), according to a set of import options that are specified in an option file.

## Specification, model, logical message tree

```

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

```



© Copyright IBM Corporation 2015

Figure 6-8. Specification, model, logical message tree

WM666 / ZM6661.0

### 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 depicted as a level in the logical message tree. For example, WorkAddress is a complex type that contains Line, Country, and PostalCode.

## Custom wire format (binary or CWF)

- Typical legacy 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
  - Dynamic length or repeat counter that is determined by preceding INTEGER field
- Fixed field order

© Copyright IBM Corporation 2015

Figure 6-9. Custom wire format (binary or CWF)

WM666 / ZM6661.0

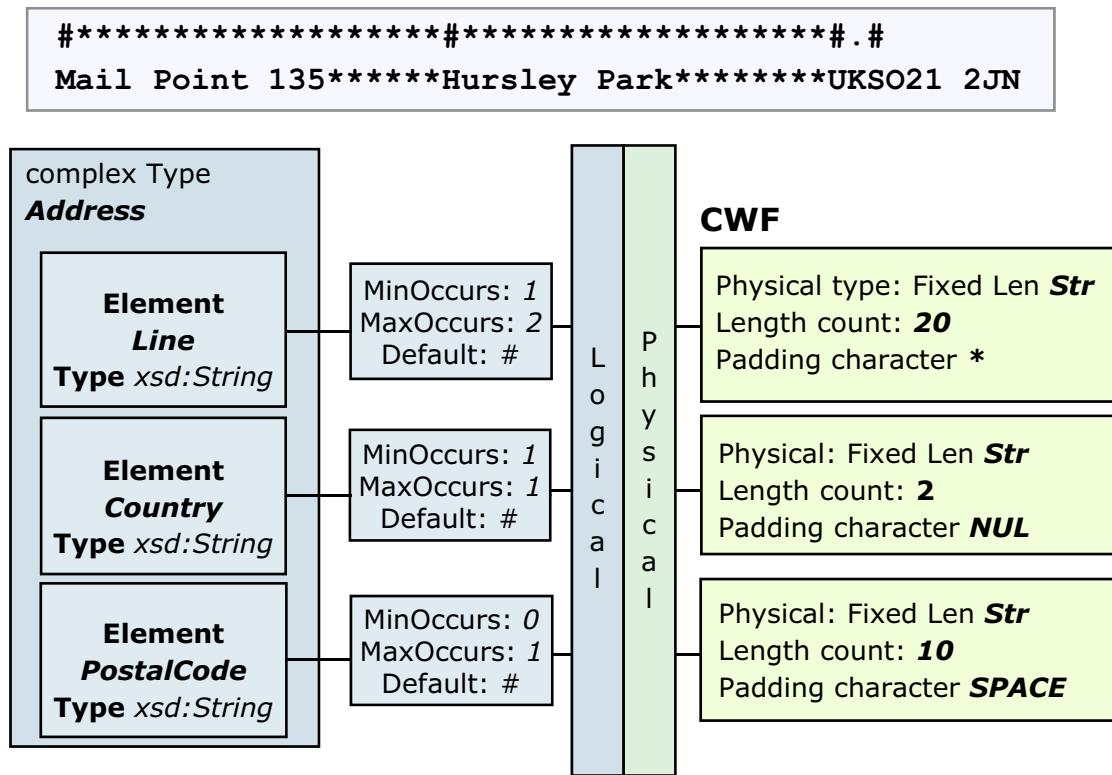
### Notes:

Within a binary or CWF data, 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 of the elements (defined in the logical properties)
- The length of the elements (specified in bytes, characters, or character units)
- The *cardinality* (the number of occurrences) of each element
- 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)
- Element characteristics that are based on the logical type of the data

While the other physical representations, such as XML, support self-defining elements (that is, elements that do not have a definition in the logical model), the parsing of a CWF message does not.

## Logical message and CWF format



© Copyright IBM Corporation 2015

Figure 6-10. Logical message and CWF format

WM666 / ZM6661.0

### Notes:

This figure shows the logical model and the physical properties of CWF data.

The first step for modeling the data is to define each element and its type.

After you define the element and type components, the next step is to specify number of occurrences of each element. You must define whether an element is mandatory (*MinOccurs=1*), optional (*MinOccurs=0*) or repeating (*MaxOccurs>1*).

The CWF format 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 bit stream) when used within the context of another complex type.

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

## Tagged and delimited string format (text or TDS)

- Typical industry standard formats, including SWIFT, EDI, ACORD, AL3, and TLOG
- 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] or {1,3}

© Copyright IBM Corporation 2015

Figure 6-11. Tagged and delimited string format (text or TDS)

WM666 / ZM6661.0

### Notes:

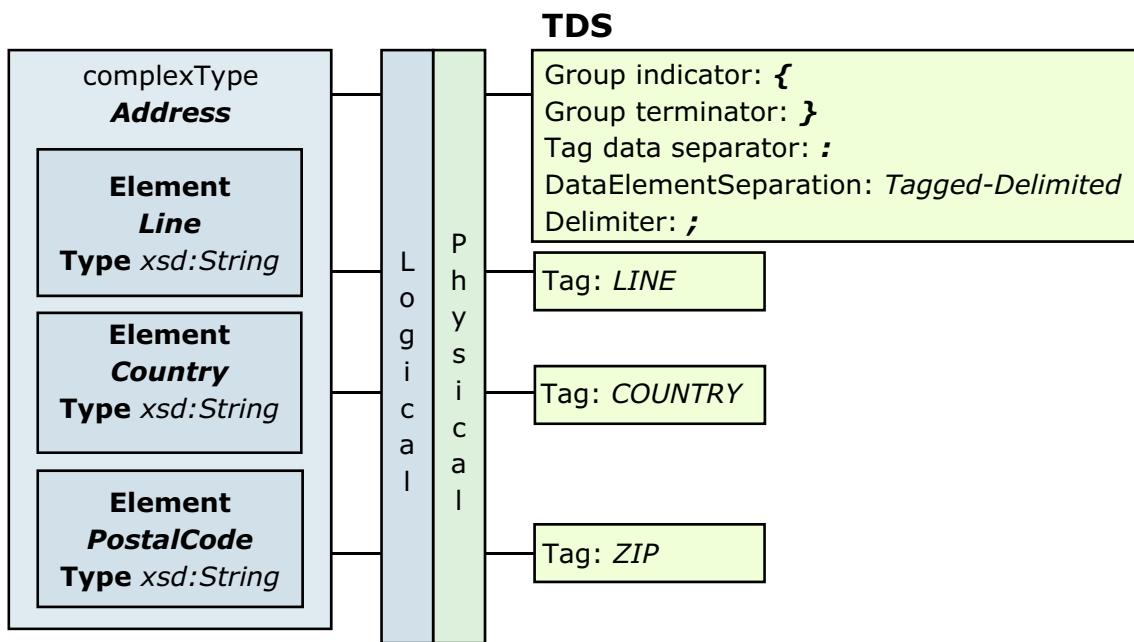
The tagged-delimited string physical format is designed to model messages that consist of text strings, but it can also handle binary data. Tagged-delimited string allows a high degree of flexibility when defining message formats, and it is not restricted to model-specific industry standards. You can 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. The tagged-delimited string format associates 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.

## Logical message and TDS format

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



© Copyright IBM Corporation 2015

Figure 6-12. Logical message and TDS format

WM666 / ZM6661.0

### 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, there is now another physical layer for tagged-delimited string 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; for example, a tagged-delimited layer and a variable-delimited layer.

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



## Data Format Description Language (DFDL)

- An open standard from the Open Grid Forum (OGF)
- Uses XML technology and concepts
  - W3C XML Schema subset and type system describe the logical format of the data
  - Annotations within the XSD describe the physical representation of the data
  - XPath notation references fields within the data
- Describes any data format
  - Textual and binary
  - Commercial record-oriented
  - Scientific and numeric
  - Modern and legacy data
  - Industry standards
- Supports round-tripping
  - Read and write data in the described format from the same description

© Copyright IBM Corporation 2015

Figure 6-13. Data Format Description Language (DFDL)

WM666 / ZM6661.0

### Notes:

In IBM Integration Bus, you can model data with DFDL. DFDL is an XML-based language that is used to define the structure of formatted data in a way that is independent from the data format itself.

DFDL is a language for describing data formats. A DFDL description allows data to be read from its physical format and to be presented as an instance of an information set (*InfoSet*) or converted to the corresponding XML document. DFDL also allows data to be taken from an instance of an InfoSet and written out to its physical format.

An XML schema is written for the logical model of the data. The schema is augmented with special DFDL annotations. These annotations are used to describe the physical representation of the data. This established approach is being used today in commercial systems.

DFDL uses W3C XML Schema Definition Language (XSDL) 1.0.

IBM Integration Bus supports a DFDL domain. The DFDL domain can be used to parse and write a wide variety of message formats, and is primarily intended for non-XML message formats.

When reading a message, the DFDL parser interprets a bit stream by using grammar that is defined in a DFDL schema file, and generates a corresponding DFDL domain logical message tree in the

integration node. When writing a message, the DFDL serializer generates a DFDL formatted bit stream from a DFDL domain logical message tree.

DFDL maps data from a non-XML representation to an instance of an information set. Think of this representation as a data transformation. However, DFDL is not intended to be a general transformation language and in particular, DFDL does not intend to provide a mechanism to map data to arbitrary XML models.



## DFDL data support (1 of 2)

- Text data types such as strings, numbers, zoned decimals, calendars, and Booleans
- Binary data types such as integers, floats, BCD, packed decimal, calendars, and 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
- 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

© Copyright IBM Corporation 2015

Figure 6-14. DFDL data support (1 of 2)

WM666 / ZM6661.0

### Notes:

By using DFDL, you can model all the types of data that IBM Integration Bus supports. This figure and the next include a complete list of supported data types.

DFDL can model a superset of the data formats that are available in the message modeling components of IBM Integration Bus. However, the traditional modeling components, such as an MRM message set, are still available.

## DFDL data support (2 of 2)

- 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
- User-defined variables in DFDL expressions
- TLOG packed numeric fields
- Delimited binary data
- Fields lengths that come from regular expressions

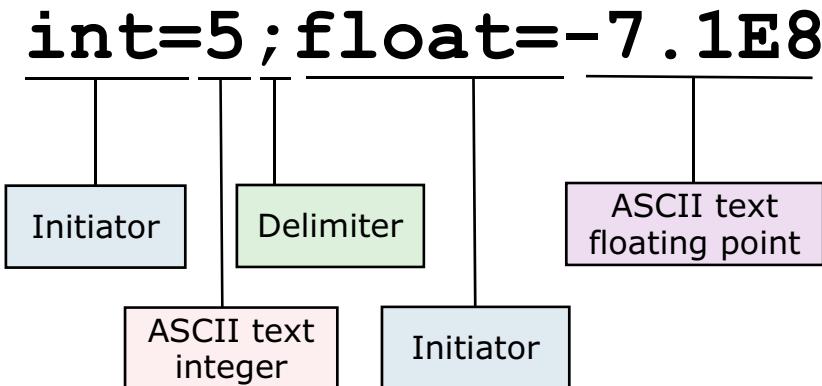
© Copyright IBM Corporation 2015

Figure 6-15. DFDL data support (2 of 2)

WM666 / ZM6661.0

### Notes:

## Example: Delimited text data



Separators, initiators (tags), and terminators are all examples of DFDL *delimiters*

© Copyright IBM Corporation 2015

Figure 6-16. Example: Delimited text data

WM666 / ZM6661.0

### Notes:

To parse data into DFDL format, you use a notation that describes the data itself. You define Initiators, terminators, and delimiters in a DFDL schema so that the parser knows how to interpret the data.

The easiest way to understand the DFDL message model is to start by modeling a simple example.

The figure shows example data and the DFDL descriptors that describe the data.

## Example: DFDL schema

```

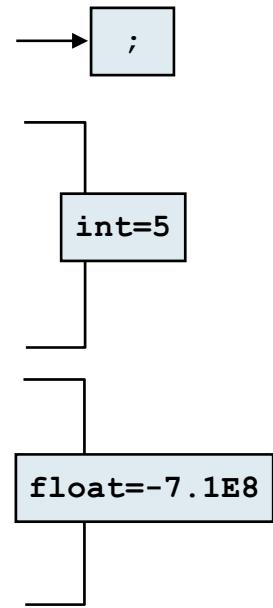
<xs:complexType name="myNumbers">
  <xs:sequence>
    <xs:annotation>
      <xs:appinfo source="http://www.ogf.org/dfdl/v1.0">
        <dfdl:sequence separator=";" encoding="ascii"/>
      </xs:appinfo>
    </xs:annotation>
    <xs:element name="myInt" type="xs:int">
      <xs:annotation>
        <xs:appinfo source="http://www.ogf.org/dfdl/v1.0">
          <dfdl:element representation="text"
            textNumberRep="standard" encoding="ascii"
            lengthKind="delimited" initiator="int=" .../>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="myFloat" type="xs:float">
      <xs:annotation>
        <xs:appinfo source="http://www.ogf.org/dfdl/v1.0">
          <dfdl:element representation="text"
            textNumberRep="standard" encoding="ascii"
            lengthKind="delimited" initiator="float=" .../>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

DFDL annotation

<dfdl:element representation="text"  
textNumberRep="standard" encoding="ascii"  
lengthKind="delimited" initiator="float=" .../>

DFDL properties



© Copyright IBM Corporation 2015

Figure 6-17. Example: DFDL schema

WM666 / ZM6661.0

### Notes:

This DFDL schema describes the format of the data on the previous figure. It describes the separator, delimiter, structure, and physical properties of the data.

## Example: DFDL schema (short form)

```

<xs:complexType name="myNumbers">
  <xs:sequence dfdl:separator=";" dfdl:encoding="ascii" >
    <xs:element name="myInt" type="xs:int"
      dfdl:representation="text"
      dfdl:textNumberRep="standard" dfdl:encoding="ascii"
      dfdl:lengthKind="delimited" dfdl:initiator="int=" ... />
    <xs:element name="myFloat" type="xs:float"
      dfdl:representation="text"
      dfdl:textNumberRep="standard" dfdl:encoding="ascii"
      dfdl:lengthKind="delimited" dfdl:initiator="float=" ... />
  </xs:sequence>
</xs:complexType>

```



© Copyright IBM Corporation 2015

Figure 6-18. Example: DFDL schema (short form)

WM666 / ZM6661.0

### Notes:

This representation is the compact version of the DFDL schema that is shown on the previous figure.



## DFDL support in IBM Integration Bus

- DFDL models (schema files) are stored in IBM Integration Bus projects
- DFDL domain and the model-driven parser are available in all nodes
- IBM Integration Toolkit has built in tools for creating DFDL models
  - DFDL model debugger to test the parsing and serialization of data by using a model
- DFDL models are deployed to the integration server

© Copyright IBM Corporation 2015

Figure 6-19. DFDL support in IBM Integration Bus

WM666 / ZM6661.0

### Notes:

DFDL models can be stored with applications in libraries.

The IBM Integration Toolkit provides several facilities for working with DFDL models, including tools for creating and testing them. You can test a DFDL model without deploying it. However, when it is time to deploy it to the integration node, you can deploy it in a library or include it in the BAR file with the other deployable components.



## When should you use DFDL?

- Use DFDL to model:
  - Binary data from COBOL, C, PL/I, ASM programs
  - Text data with delimiters such as CSV
  - Text industry standards such as SWIFT, HL7, EDIFACT, and X12
  - Binary industry standards such as ISO8583 and TLog
- Do not use DFDL to model:
  - XML data that can use XML parsers and have an XML schema or DTD
  - JSON that can use JSON parsers
  - Serialization formats where the wire format is never available to the consumer and access uses APIs
- Do not use DFDL expressions to implement complex validation rules

© Copyright IBM Corporation 2015

Figure 6-20. When should you use DFDL?

WM666 / ZM6661.0

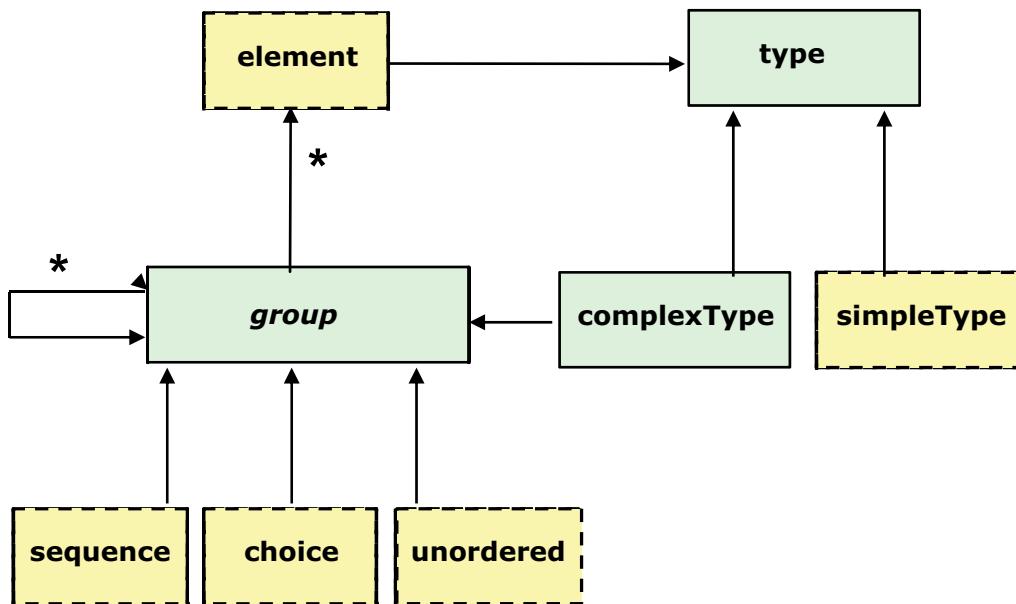
### Notes:

Use DFDL when you need to model and parse text or binary data where either:

- You have a specification of the data format
- You have examples of the data

A Gerber File is a file format is an example of a serialization format. It used by printed circuit board manufacturing machines to lay out electrical connections such as traces, vias, and pads. In addition, the file contains information for drilling, and milling the completed circuit board.

## DFDL object model



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

© Copyright IBM Corporation 2015

Figure 6-21. DFDL object model

WM666 / ZM6661.0

### Notes:

When you model with DFDL, objects define the logical format of the data. This figure shows the elements and relationships.

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 physical representation of the data. These DFDL annotations (properties) are valid only on sequence, choice, element, and simpleType.

A *group* can be a sequence group where the elements are always in a specific order. An example of a group object is header record or trailer record. They are made up of *simpleTypes* (such as fields) but might also contain other *complexType* objects.

A group can also be a choice group. Mixed data that contains purchase order records and acknowledgment records would be an example of a choice group.



## DFDL schema editor

- Use the DFDL schema editor to create, edit, and test DFDL schema files
- DFDL schema editor is started when you open an existing DFDL schema file or when you create a DFDL schema file by using the **New Message Model** wizard
- With the DFDL schema editor, you can:
  - Edit DFDL schema files
  - Populate empty DFDL schema files with DFDL schema objects
  - Parse the DFDL schema files against sample input data
  - Serialize a logical instance document against your DFDL schema file

© Copyright IBM Corporation 2015

Figure 6-22. DFDL schema editor

WM666 / ZM6661.0

### Notes:

Use the DFDL schema editor in the Integration Toolkit to create, edit, and test DFDL schema files.

The DFDL schema editor is started when you open an existing DFDL schema file, or when you create a DFDL schema file by using the wizard.

You use the DFDL schema editor to:

- Edit DFDL schema files. The wizard creates the schema files that are automatically populated with content, which you edit as required.
- Populate empty DFDL schema files with DFDL schema objects by creating the DFDL schema objects and the DFDL properties that are required to represent your message format.
- Test your DFDL parser against sample input data, and view a visual representation of how the DFDL parser uses the DFDL schema annotations to parse your input data.
- Test a logical instance document against your DFDL serializer, and view a visual representation of how the DFDL serializer uses the DFDL schema annotations to serialize the output bit stream.



## Creating a DFDL schema file with the New Message Model wizard

- You can use the **New Message Model** wizard to generate a DFDL schema file by modeling an existing data file
  - Comma-separated value (CSV) data
  - Record-oriented data
  - Custom text or binary data
  - COBOL data structures

© Copyright IBM Corporation 2015

Figure 6-23. Creating a DFDL schema file with the New Message Model wizard

WM666 / ZM6661.0

### Notes:

The IBM Integration Toolkit has a wizard to help you create a DFDL schema file by modeling existing data.

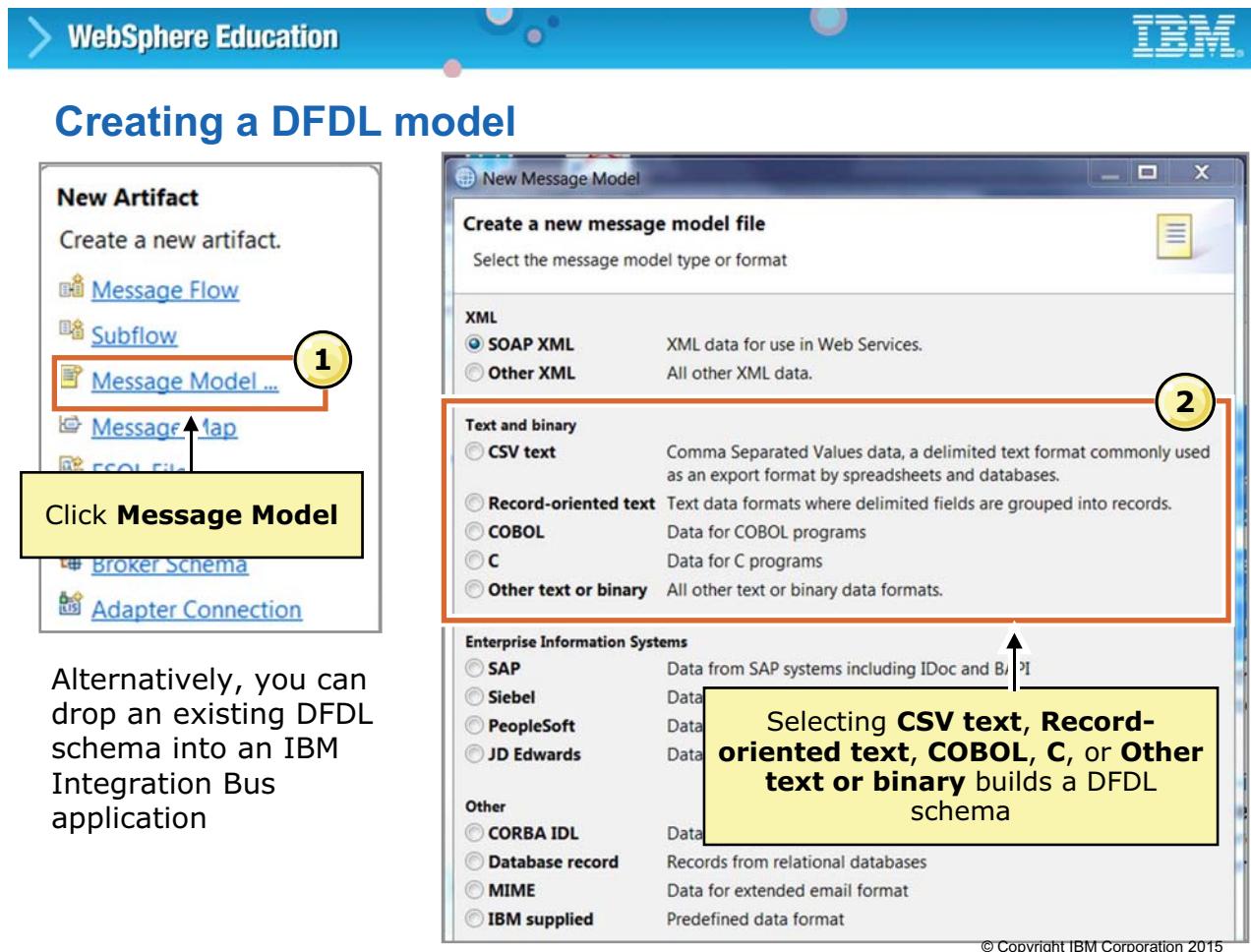


Figure 6-24. Creating a DFDL model

WM666 / ZM6661.0

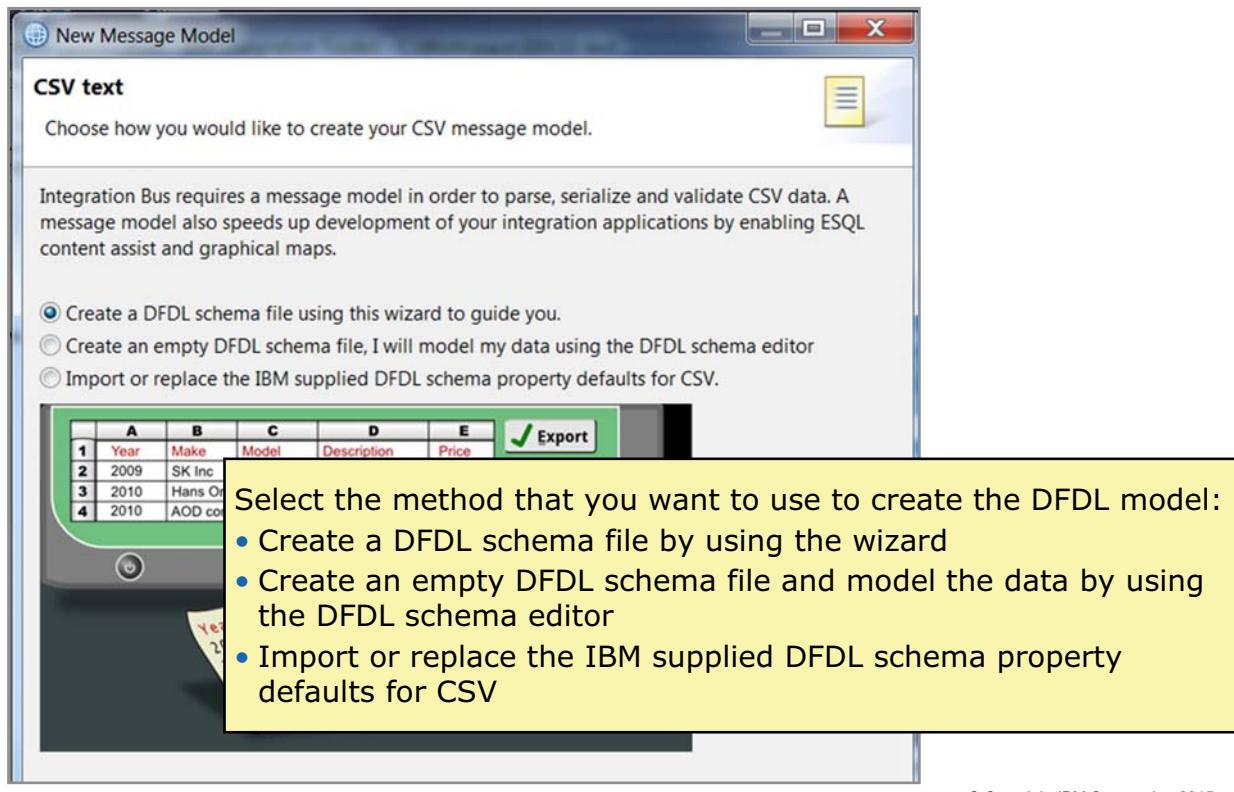
## Notes:

To create a message model, you first define the type of data the model contains.

1. From the toolbar, click **File > New**, and then select **Message Model**.
2. Select the type of data you want to model.



## Wizard options for creating a DFDL model



© Copyright IBM Corporation 2015

Figure 6-25. Wizard options for creating a DFDL model

WM666 / ZM6661.0

### Notes:

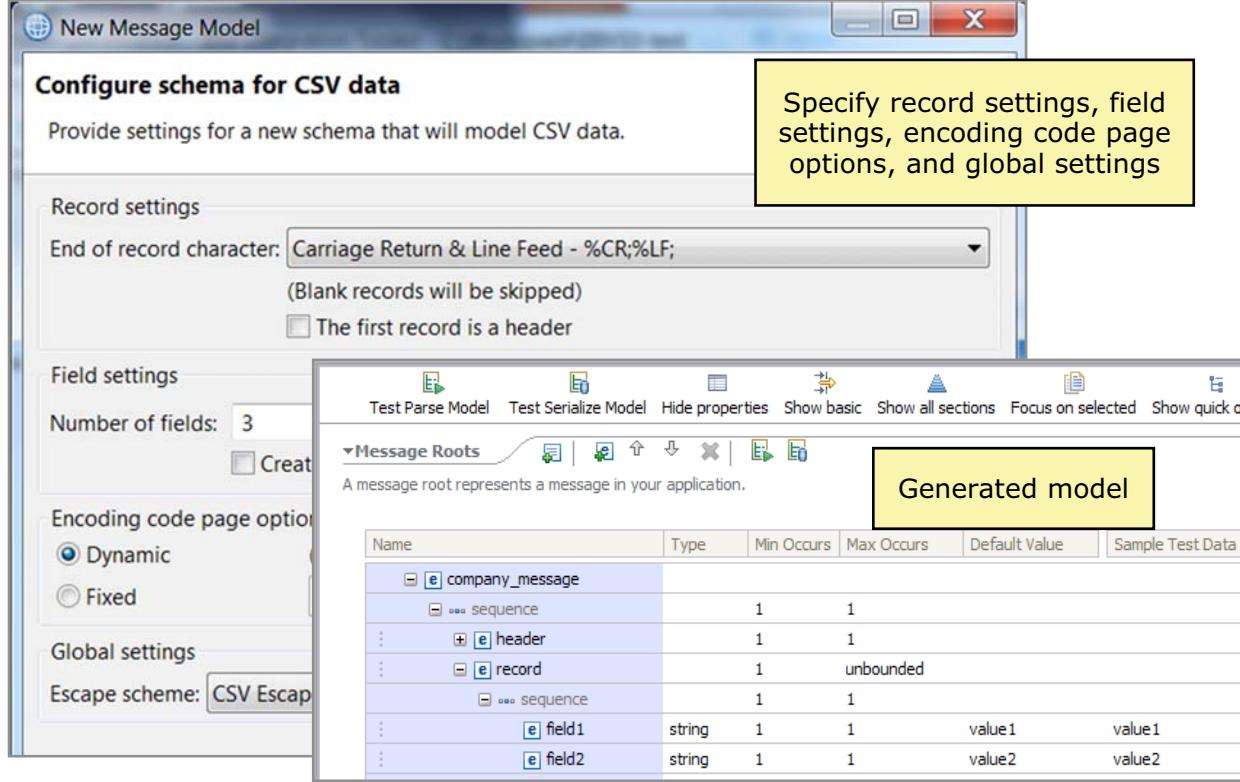
You can choose from three ways to construct the DFDL schema. The DFDL wizard assists you with your choices, and with constructing the schema if you choose to use it.

You can use the New Message Model wizard to create an XML schema file by any of the following methods:

- Create a DFDL schema file by using the wizard to guide you.
- Create an empty DFDL schema file and model the data by using the DFDL schema editor.
- Import or replace the IBM supplied DFDL schema property defaults.

## Creating a DFDL model by using guided authoring



Specify record settings, field settings, encoding code page options, and global settings

Record settings

End of record character: Carriage Return & Line Feed - %CR;%LF;  
 (Blank records will be skipped)  
 The first record is a header

Field settings

Number of fields: 3  
 Create

Encoding code page option

Dynamic  
 Fixed

Global settings

Escape scheme: CSV Escap

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Test Data
company_message					
sequence		1	1		
header		1	1		
record		1	unbounded		
sequence		1	1		
field1	string	1	1	value1	value1
field2	string	1	1	value2	value2

Generated model

© Copyright IBM Corporation 2015

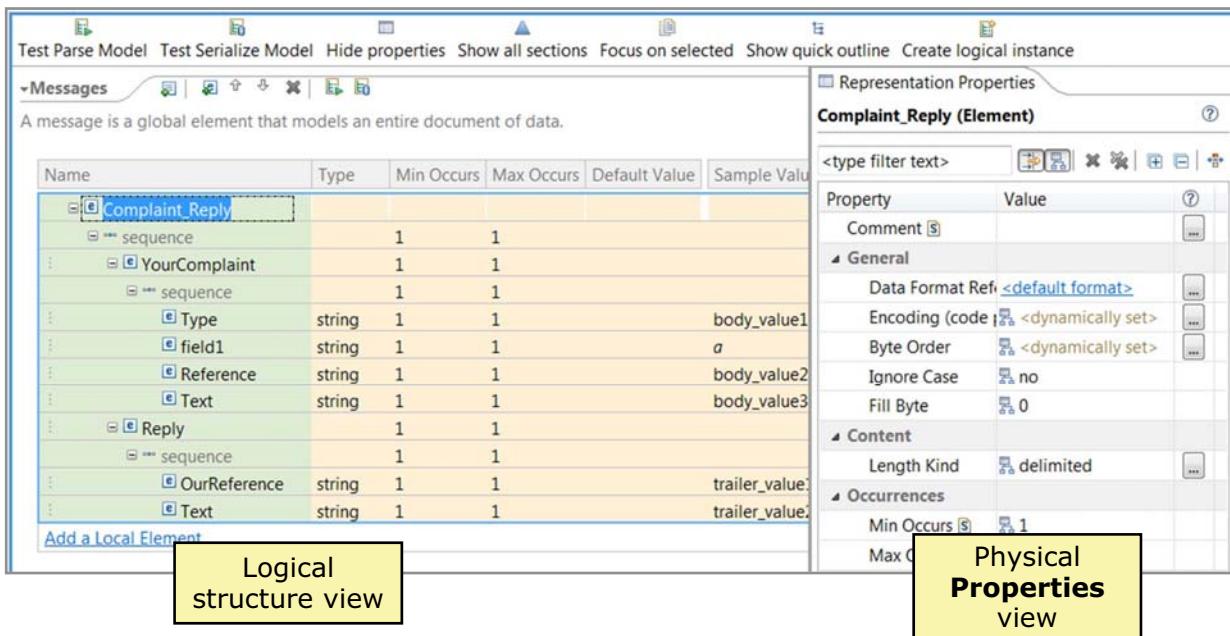
Figure 6-26. Creating a DFDL model by using guided authoring

WM666 / ZM6661.0

### Notes:

One way to create a DFDL schema is by using guided authoring. You specify the characteristics of the data you want to model, such as field delimiters, record terminator characters. The wizard then generates the schema that is based on those specifications.

## Creating a DFDL model by using the editor (1 of 2)



© Copyright IBM Corporation 2015

Figure 6-27. Creating a DFDL model by using the editor (1 of 2)

WM666 / ZM6661.0

### Notes:

The DFDL schema editor uses several Eclipse views, which are organized according to the current perspective.

The DFDL schema editor also contains some menu-driven options to help you construct the schema, including the ability to parse and serialize test data.



## Creating a DFDL model by using the editor (2 of 2)

The screenshot shows the IBM Integration Toolkit interface for creating a DFDL model. The main window is divided into several views:

- DFDL properties view:** Displays the configuration for the element "EmpName". Properties include:
  - Comment: <dynamically set>
  - Byte Order: <dynamically set>
  - Content: string (text, delimited)
  - Text Content: Escape Scheme Reference: recSepFieldsFmt:RecordEscapeSd...
  - Occurrences: Occurs Count Kind: fixed (Min Occurs: 1, Max Occurs: 2)
- Outline view:** Shows the hierarchical structure of the schema, including Schema, Imports (IBMdefined/CobolDataDefinitionFormat.xsd), Message Roots (CompanyTaggedDelimited, Employee), Elements, Types (Complex Types: Employee, Simple Types), Groups, DFDL (Escape Schemes, Formats, Variables), and Variables.
- Problems view:** Displays one error: CTDV1101E : Element declaration :Employee: with occursCountKind="fixed" does not support different minOccurs and maxOccurs values.

© Copyright IBM Corporation 2015

Figure 6-28. Creating a DFDL model by using the editor (2 of 2)

WM666 / ZM6661.0

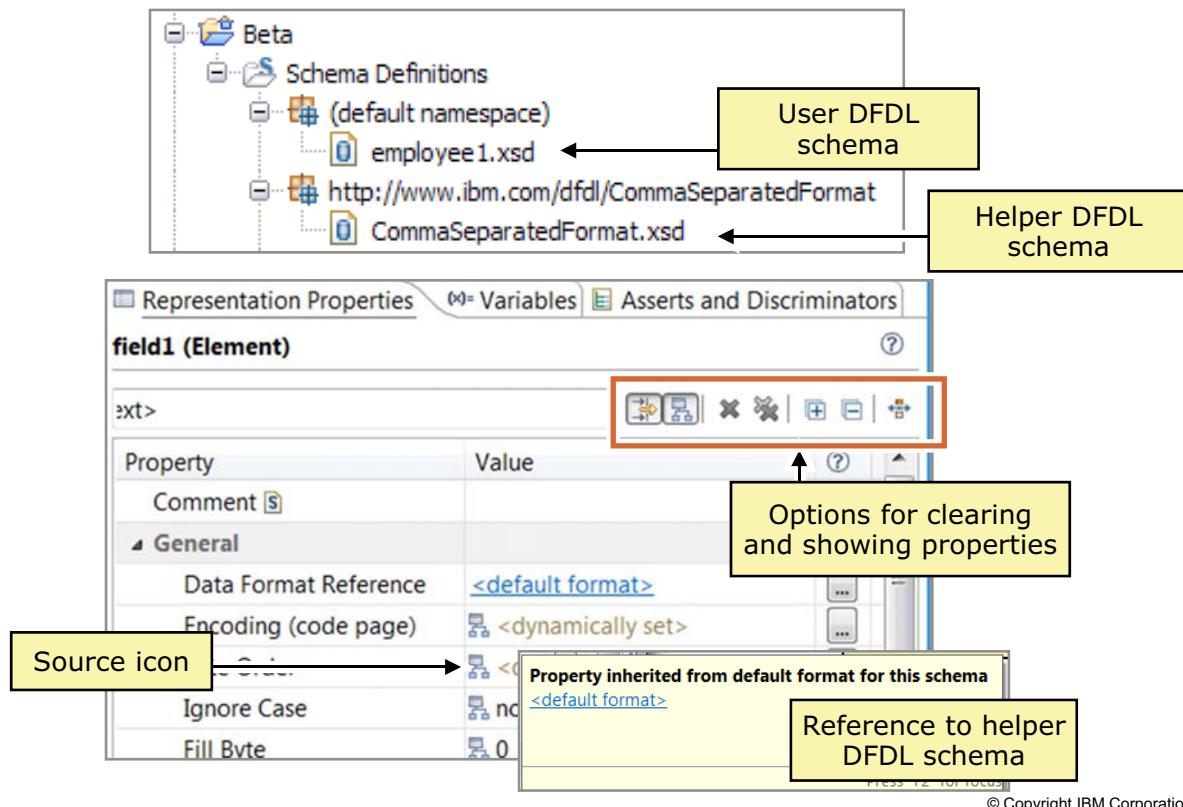
### Notes:

As you construct the DFDL schema, the IBM Integration Toolkit displays the DFDL schema and properties, and displays an outline of the components of the schema.

As with other development tasks, the **Problems** view indicates whether there are any issues with the schema as you construct it. Red markers in the editor highlight the areas where errors are found.



## Created DFDL schemas



© Copyright IBM Corporation 2015

Figure 6-29. Created DFDL schemas

WM666 / ZM6661.0

### Notes:

When you use the New Message Model to create a DFDL schema, it automatically generates a “helper schema” that contains default values for the properties.

After the schema is created, you can review all the details in the IBM Integration Bus Toolkit.

Use the **Representation Properties** tab on the right of the editor to modify properties.

- Use **Filter** to find the property that you want to change.
- To unset a property value, use the icons to the right of the filter bar. Using the Delete key might set the property to an empty string, rather than being unset.
- Click **Show basic** to filter properties. If only basic properties are displayed, click **Show advanced** to display all properties.
- You can select multiple objects in the Editor, but you see properties for only the first item that you selected.



## Helper DFDL schemas

- DFDL properties do not have built-in defaults, so if an object needs a property, a value must be supplied
- **New Message Model** wizard automatically creates a helper DFDL schema with values that are set for most DFDL properties, and adds it as an import into the user DFDL schema
  - Different helper DFDL schema for each type of data such as COBOL or CSV
  - Helper DFDL schemas are created only one time per IBM Integration Bus application
  - Helper DFDL schemas are created as read-only but can be edited by using the DFDL editor, if required
  - Helper DFDL schemas must be deployed with user DFDL schema

© Copyright IBM Corporation 2015

Figure 6-30. Helper DFDL schemas

WM666 / ZM6661.0

### Notes:

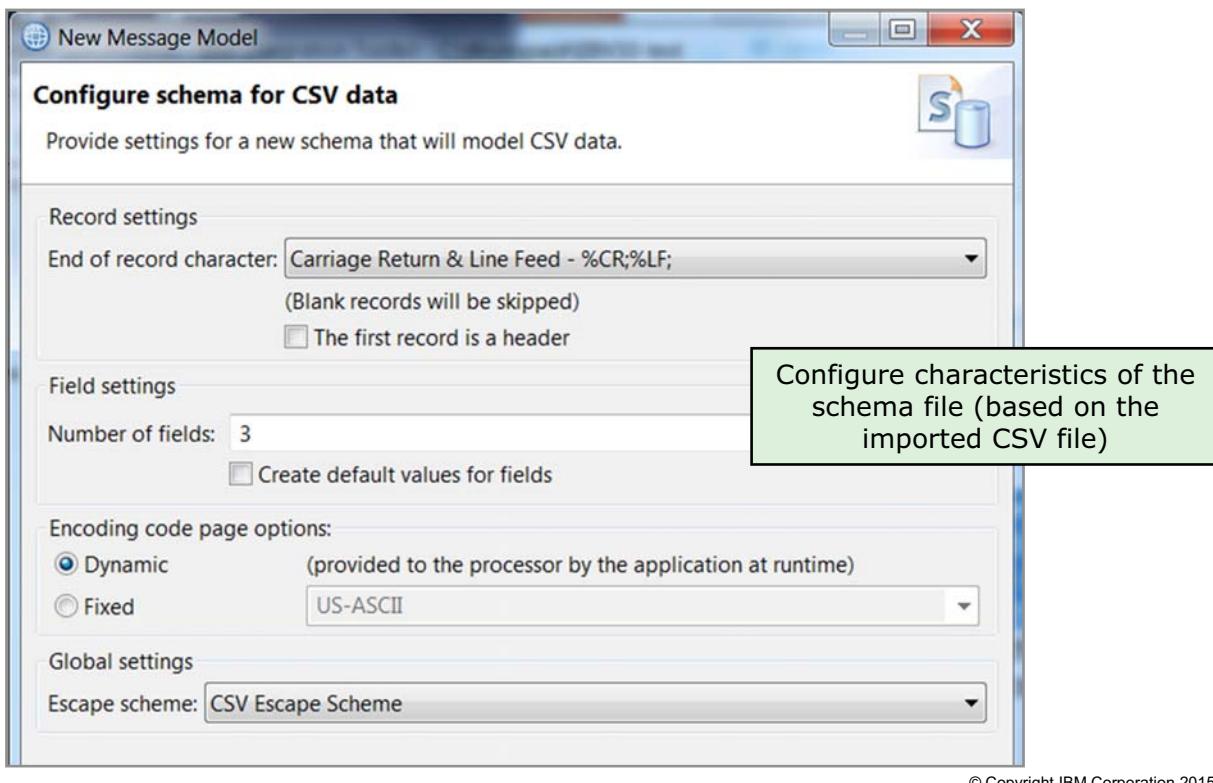
There are no default values for DFDL properties. All properties must be explicitly specified in the schema.

When you use the New Message Model wizard to construct the schema, it automatically generates a “helper” schema. A “helper” schema is an IBM supplied DFDL model that contains **defineFormat** tags. It constructs these schemas based on the type of data that you are modeling. Each type of data (for example, a COBOL Copybook or comma-separated values) results in the creation of a unique helper schema.

At run time, you must deploy both the user-generated schema and the “helper” schema to the integration server.



## New Message Model wizard example



© Copyright IBM Corporation 2015

Figure 6-31. New Message Model wizard example

WM666 / ZM6661.0

### Notes:

The type of data that you are trying to model, determines the New Message Model wizard options. This figure shows the options that you have when you use the New Message Model wizard to define CSV data.

CSV format consists of a number of records, each containing fields that are separated by a comma. Records are ended by operating system-specific new line characters.

Options for creating a DFDL model include:

- The end of record character
- The number of fields in a record
- Handling of code page

The screenshot shows the DFDL schema editor interface. At the top, there are buttons for 'Test Parse Model', 'Test Serialize Model', 'Hide properties', 'Show advanced', 'Show all sections', and 'Focus on'. Below this is a toolbar with icons for creating, deleting, and modifying elements. A message bar at the top says 'Messages' and provides instructions: 'message is a global element that models an entire document of data.' A callout box states 'After the wizard finishes, a generic schema is displayed'. The main area is a table with columns: Name, Type, Min Occurs, Max Occurs, Default Value, and Sample. The table data is as follows:

Name	Type	Min Occurs	Max Occurs	Default Value	Sample
expenseaudit	sequence	1	1		
	record	1	unbounded		
	sequence	1	1		
	field1	string	1	value1	value1
	field2	string	1	value2	value2
	field3	string	1	value3	value3
	field4	string	1	value4	value4

© Copyright IBM Corporation 2015

Figure 6-32. DFDL schema editor (1 of 3)

WM666 / ZM6661.0

### Notes:

The DFDL schema editor generates a generic model that is based on the New Message Model wizard options. In the example, a record element is defined as consisting of four fields.



## DFDL schema editor (2 of 3)

message is a global element that models an entire document of data.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
<b>e expenseaudit</b>					
sequence		1	1		
record		1	unbounded		
sequence		1	1		
last_name	string	1	1	<empty string>	Smith
amount	decimal	1	1	0.00	1.0
expense	string	1	1	<empty string>	travel
audit_reason	string	1	1	<empty string>	invalid amount
<a href="#">Add a Local Element</a>					

Use this toolbar to add, remove, and reorder elements

Add a Local Element

Modify default names, data types, and other schema properties

© Copyright IBM Corporation 2015

Figure 6-33. DFDL schema editor (2 of 3)

WM666 / ZM6661.0

### Notes:

The next step is to rename the fields to more descriptive names and if necessary, change the data type, occurrences, and default values.

The value that is entered in the **Sample Value** column in the main editor view is the lexical representation of the corresponding XML schema type.

Use the toolbar that is located below the main toolbar, to the left of the Editor view to add, remove, and reorder elements.



## DFDL schema editor (3 of 3)

**amount (Element)**

r text>	
Property	Value
Comment	
<b>General</b>	
Encoding (code page)	<dynamically set>
Byte Order	bigEndian
<b>Content</b>	decimal
Representation	text
Length Kind	delimited
Default Value	0.00
<b>Text Content</b>	
+ Text Number Representation	standard
Escape Scheme Reference	<a href="#">csv:CSVEscapeScheme</a>
<b>Occurrences</b>	
Min Occurs	1
Max Occurs	1
<b>Delimiters</b>	

The right side of the schema editor shows the DFDL properties of the selected item

© Copyright IBM Corporation 2015

Figure 6-34. DFDL schema editor (3 of 3)

WM666 / ZM6661.0

### Notes:

The next step is to define the field and record properties.

A Property wizard is available on some properties to help you determine the correct setting.



## DFDL schema editor toolbar actions



- **Test Parse Model:** Runs the parser against a data file with the model
- **Test Serialize Model:** Runs the serializer against logical instance data
- **Hide properties / Show properties:** Hides or reveals the element **Properties** pane
- **Show all sections / hide empty sections:** Hides or reveals schema definition sections that contain no information
- **Focus on selected:** When **Show all sections** is selected, the focus is on the currently selected item
- **Show quick outline:** Displays a summary outline menu of the DFDL schema
- **Create logical instance:** Creates a logical instance for the Test Serialize Model

© Copyright IBM Corporation 2015

Figure 6-35. DFDL schema editor toolbar actions

WM666 / ZM6661.0

### Notes:

A main toolbar spans the DFDL schema editor and the DFDL **Properties** area. Use the options on this toolbar to control what is displayed to test parse and test serialize the model, and to create a logical instance of the message model.



## Testing the DFDL schema

- Parse a data file by using the model as it is defined at the time
- Test the schema definition on real data, and modify it if needed
- IBM DFDL editor uses parser error markers to highlight errors
  - ✖ Static schema error when a schema is saved
  - ✖ Parser error when test parsing data
  - ✖ Serializer error when test serializing a logical instance
- Click **Test Parse Model**
  - Opens the DFDL **Test** perspective in the IBM Integration Toolkit
  - Specify the input file to use for testing; either a resource that is in the application, or one from the file system

© Copyright IBM Corporation 2015

Figure 6-36. Testing the DFDL schema

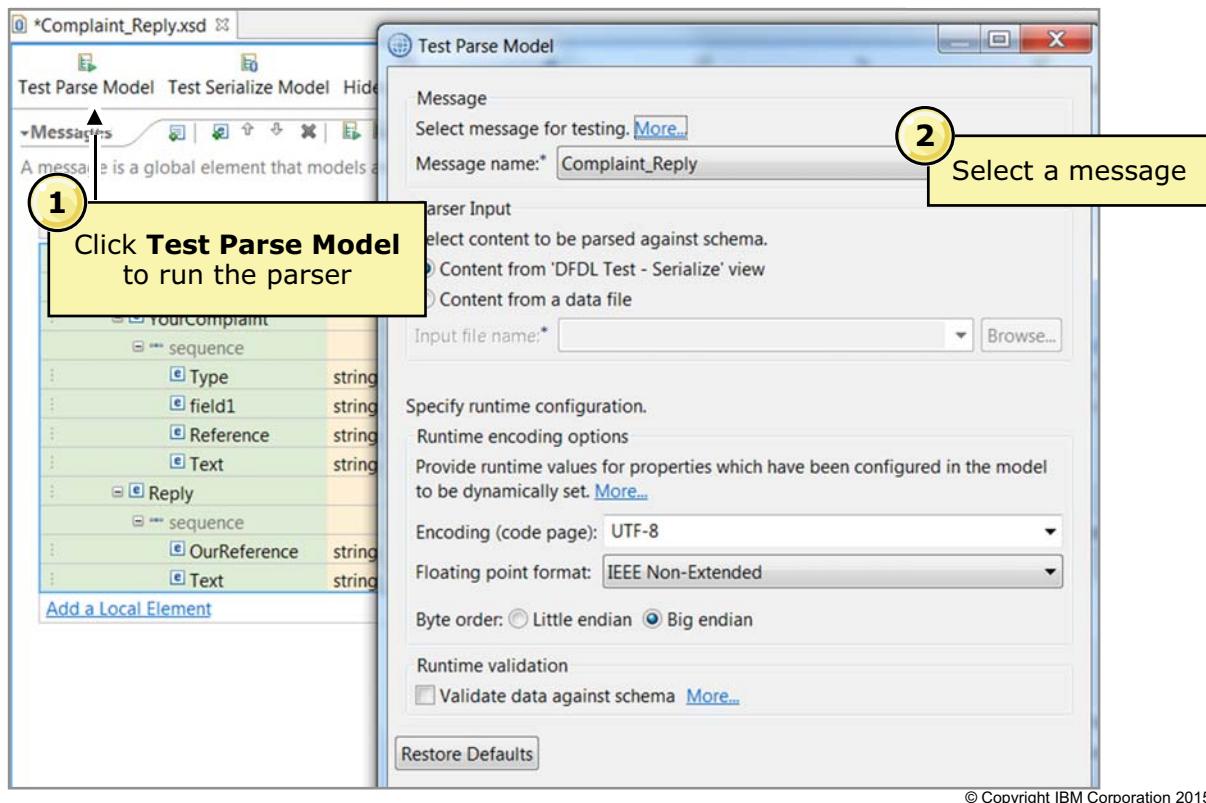
WM666 / ZM6661.0

### Notes:

You can test the accuracy of your model by using the schema to parse some sample data. Parsing the data verifies that the physical bit stream data is converted to the logical message tree as expected. You can also serialize the data; that is, call the parser to convert the logical message tree to a bit stream representation.



## Testing a DFDL model within the editor (1 of 4)



© Copyright IBM Corporation 2015

Figure 6-37. Testing a DFDL model within the editor (1 of 4)

WM666 / ZM6661.0

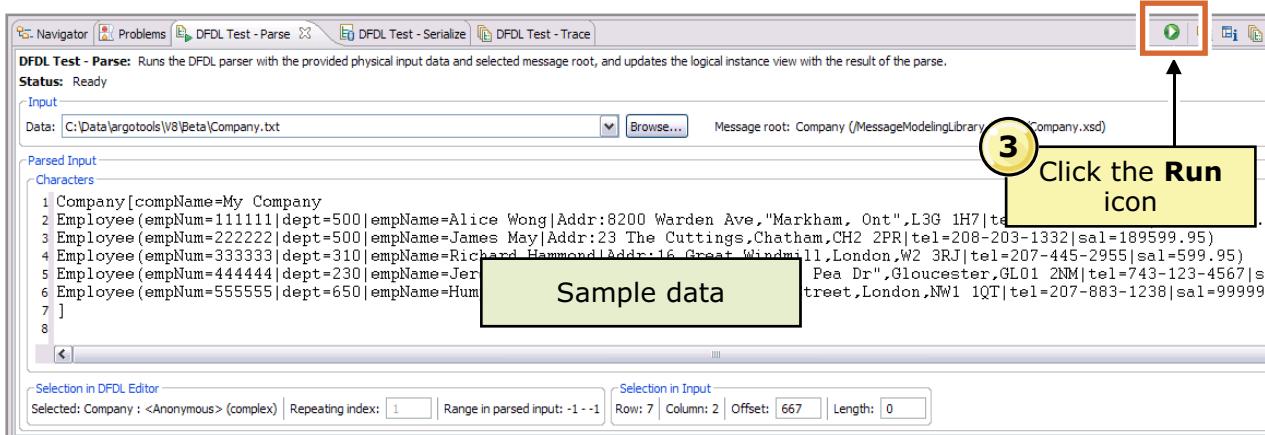
### Notes:

As was mentioned previously, you can test the DFDL schema by parsing or serializing test data. This example shows the steps that you take to test the DFDL schema by parsing some sample data.

1. In the DFDL schema editor, click **Test Parse Model**.
2. Select the sample message that you want to test by specifying the name of the file and characteristics. The test file can be in the workspace or another place on the file system. You also have the option of specifying runtime configuration such as encoding options and validation.

## WebSphere Education

### Testing a DFDL model within the editor (2 of 4)



© Copyright IBM Corporation 2015

Figure 6-38. Testing a DFDL model within the editor (2 of 4)

WM666 / ZM6661.0

**Notes:**

3. After you load the sample data to parse, click **Run** to run the test.



## Testing a DFDL model within the editor (3 of 4)

The screenshot shows the 'DFDL Test - Logical Instance' window. At the top, it displays the 'Data source: <From 'DFDL Test - Parse' view>' and 'Message root: Company (/MessageModelingLibrary\_broker/Company.xsd)'. Below this, there are two tabs: 'Tree View' and 'XML View', with 'Tree View' selected. A table titled 'Name' lists the structure of the XML document:

Name	Type	Value
- Company		
CompanyName	xs:string	My Company
Employee		
EmpNo	xs:integer	111111
Dept	xs:integer	500
EmpName	xs:string	Alice Wong
Address		

Below the table, the status bar shows 'DFDL Test - Parse: Runs the DFDL parser with the provided physical input data and selected message root, and updates the logical instance view with the result of the parse.' and 'Status: Parsing completed: Tue Jun 14 13:49:37 BST 2011'. The 'Input' section shows the file path 'Data: C:\Data\argotools\V8\Beta\Company.txt' and the 'Message root: Company (/MessageModelingLibrary\_broker/Company.xsd)'. The 'Parsed Input' section contains the XML data with delimiters highlighted in purple. The 'Hex' section shows the corresponding hexadecimal representation of the data. Three callout boxes highlight specific areas: 'Parsed 'infoSet'', 'Parsed data with the delimiters highlighted', and 'Hexadecimal view of the data'.

© Copyright IBM Corporation 2015

Figure 6-39. Testing a DFDL model within the editor (3 of 4)

WM666 / ZM6661.0

### Notes:

When the parser test runs, the parsed data is displayed (in both character and hexadecimal format) with the delimiters. You can also view the logical instance of the data in a separate tab to verify that the parsed structure is correct.

This view is used to verify that the schema is correctly defined. The figure shows an example of the parsed data from a parser test.



## Testing a DFDL model within the editor (4 of 4)

**① Parsing completed successfully.**

A message indicates when the test parsing is complete without errors

Tips:

- Selecting an element in the DFDL editor will cause the parsed input to be highlighted.
- The view menu on the view toolbar provides options to control how the parsed input is displayed.
- To view the logical instance that was created by the DFDL parser, click the Logical Instances tab.
- To view the trace captured while running the DFDL parser, click the Trace tab.

© Copyright IBM Corporation 2015

Figure 6-40. Testing a DFDL model within the editor (4 of 4)

WM666 / ZM6661.0

### Notes:

At the end of the test, you receive a success or failure message.

## Test parse logical instance

```

<?xml version="1.0" encoding="UTF-8" ?>
- <expenseaudit
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema
  -instance">
- <record>
  <last_name
    xsi:type="xs:string">Walters</last_name>
  <amount
    xsi:type="xs:decimal">250</amount>
  <expense xsi:type="xs:string">computer
    software</expense>
  <audit_reason xsi:type="xs:string">invalid
    budget center</audit_reason>
</record>
- <record>
  <last_name
    xsi:type="xs:string">Andrews</last_name>

```

A logical instance shows how the file was parsed into XML that the DFDL schema generated

© Copyright IBM Corporation 2015

Figure 6-41. Test parse logical instance

WM666 / ZM6661.0

### Notes:

The **Logical Instance** view shows the results that the DFDL parser got when parsing your input data. The results are in a logical tree format, sometimes called an InfoSet. You can click **Create Logical Instance** to generate data in the view.

The **Logical Instance** view has two tabs, a **Tree** view and an **XML** view. Both tabs display the same data, in different formats. The figure is an example of logical instance **XML** view.

The logical instance can also be used as input data to the DFDL serializer.



## Test parse failure

Status: ✖ Parsing completed with processing errors: Mon Jul

**✖ DFDL Processing Error**

Processing errors were encountered during parsing.  
You are advised to read the DFDL Trace to find out the root cause.

▼ Processing Errors

If parsing fails, the error message indicates the reasons

CTDP3108E: When parsing, the IE ParsedDataRegion[SimpleExpenseLineItem] failed to parse the element 'last\_name' because it did not have a matching start tag.

Parsed Input	Parsed file shows delimiters and the last element that was correctly parsed
Characters	<pre> 1 Walters,250.00,computer software,invalid budget center 2 Andrews,6.50,copying,wrong budget center 3 Epten,3504.28,hardware,not authorized 4 Pickens,2.50,miscellaneous,invalid expense code 5 Palman,18250.00,furniture,no second line manager approval 6 Cork,548.19,travel,invalid budget center 7 Will,19.95,miscellaneous,<b>no such last name in EMF</b> 8   </pre>

© Copyright IBM Corporation 2015

Figure 6-42. Test parse failure

WM666 / ZM6661.0

### Notes:

If an error occurs during parsing, the following resources are available to help correct the model:

- An error message is displayed. The message summarizes the problem, and contains links to view the trace and partial logical instance that was created.
- The **DFDL Test - Parse** view shows the location of the error in the data. The top of this view has a hyperlink that opens the **DFDL Test - Trace** view.
- The object in error is marked in the Editor.
- The **DFDL Test - Logical Instance** view shows that the InfoSet parsed up to the error.



## Debugging a DFDL model test failure (1 of 2)

The screenshot shows the WebSphere Studio Application Developer interface with two main windows:

- DFDL Test - Logical Instance** window (right side):
  - Data source: <From 'DFDL Test - Parse' View>
  - Message root: Company (/MessageModelingLibrary\_broker/Company.xsd)
  - Tree View: Displays a hierarchical tree of schema elements.
  - XML View: Displays XML data for the 'Company' element, including fields like CompanyName (My Company), Employee (EmpNo: 111111, Dept: 500, EmpName: Alice Wong), and Address (Tel: 905-347-5649).
  - Error message box: A green box containing the text "Error message".
- DFDL Editor** window (left side):
  - Shows the XML schema definition for 'Company.xsd'.
  - A red error icon is present next to the 'Employee' element in the schema editor.
  - An **DFDL Parse Error** dialog box is open, stating: "Errors were encountered during parsing. See the DFDL trace for details." It includes a list of errors and a checkbox: "Do not display this message again".
  - Parse trace output is shown at the bottom of the editor.

© Copyright IBM Corporation 2015

Figure 6-43. Debugging a DFDL model test failure (1 of 2)

WM666 / ZM6661.0

### Notes:

If a parsing error occurs during testing (most likely because the schema was incorrectly defined for the data), the editor displays messages to help you determine where the parse failure occurred. You can also review the parse trace for more details.

 WebSphere Education 

## Debugging a DFDL model test failure (2 of 2)

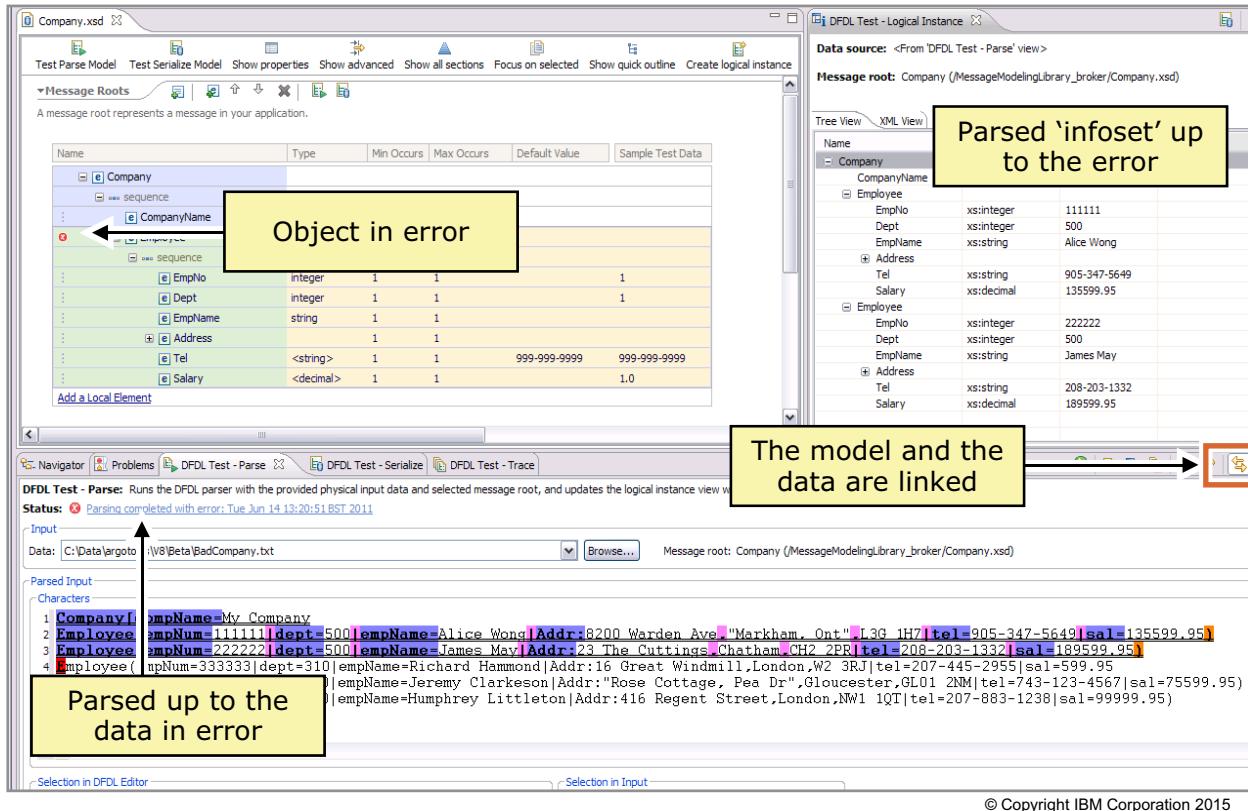


Figure 6-44. Debugging a DFDL model test failure (2 of 2)

WM666 / ZM6661.0

### Notes:

The **DFDL Test - Parse** view shows that the parse operation was partially successful. The **Logical Instance** view shows the data that was successfully parsed. This view can help you to determine where parsing failed.

## Trace console

DFDL Trace Console

```

06-14-2011 13:07:53 info: Offset: 386 Found separator "|" for element "GROUP SEQUENCE".
[fdl = /MessageModelingLibrary_broker/Company.xsd, scd = #xscd(/schema

06-14-2011 13:07:53 info: Offset: 387 Starting to process element "/Company[1]/Employee[3]/".
[fdl = /MessageModelingLibrary_broker/Company.xsd, scd = #xscd(/schema

06-14-2011 13:07:53 info: Offset: 387 Found initiator "sal=" for element "Salary".
[fdl = /MessageModelingLibrary_broker/Company.xsd, scd = #xscd(/schema

06-14-2011 13:07:53 info: Offset: 391 Found delimited value: "599.95"
Employee(empNum=444444" for element "Salary".
[fdl = /MessageModelingLibrary_broker/Company.xsd, scd = #xscd(/schema

06-14-2011 13:07:53 error: CTDP3043E: Text to number conversion error for string: "599.95
Employee(empNum=444444". Character "
" at offset 6 is unparseable.

```

- A fine-grained trace shows details of the test parsing operations
- Generated for every test run, whether it is successful or it fails
- Useful for detecting errors in the model

© Copyright IBM Corporation 2015

Figure 6-45. Trace console

WM666 / ZM6661.0

### Notes:

Another place to view parse operations is in the **DFDL Test - Trace** view. You use this view to identify any DFDL parsing errors that occurred during test parsing of sample data, or any DFDL serializing errors that occurred during test serialization of sample data.

In the example, the test failed because of a text to number conversion failure.

# WebSphere Education

## Test serialize model



Complaint\_Reply.xsd

Test Parse Model Test Serialize Model Show properties Show advanced Show all sections 3

Messages A message is a global element that models an entire document of data.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Complaint_Reply	sequence	1	1		
YourComplaint	string	1	1		body_value1
sequence	string	1	1		body_value2
Type	string	1	1		body_value1
Reference	string	1	1		body_value2
Text	string	1	1		body_value1

DFDL Test - Logical Instance Data source: <From 'DFDL Test - Parse' view> Message: Complaint\_Reply (/Workspace/WM665/ComplaintRe... Tree View XML View Name Type Value Complaint\_Reply YourComplaint Type xs:string Delivery Reference xs:string XYZ123ABC Text xs:string My order was delivered Reply OurReference xs:string C01-COM684a2da-384 Text serialized Your complaint has been received

Navigator Problems DFDL Test - Parse DFDL Test - Serialize DFDL Test - Trace

DFDL Test - Serialize: Runs the DFDL serializer with the provided logical instance data, and displays the resulting physical data. Status: Serialization completed: Wed Jul 17 15:52:37 EDT 2013 Input Logical instance: <From 'DFDL Test - Logical Instance' view> Browse... Encoding (code page): UTF-8 Schema: /ComplaintReply/Complaint\_Re... Serialized Output Characters 1 Delivery+++XYZ123ABC+++My order was delivered in time, but the package was torn and dirty|C01-0

- Use the DFDL **Test** perspective to create sample data from the selected DFDL message to verify that the DFDL schema is correct for creating output
  - Test with the populated **DFDL Test - Logical Instance** view
  - Test with content from a file in the workspace or on the file system

© Copyright IBM Corporation 2015

Figure 6-46. Test serialize model

WM666 / ZM6661.0

### Notes:

If you are using the DFDL schema to create output, you should test it with the **Test Serialize Model** option in the **DFDL Test** perspective.

- To start the test, click **Test Serialize Model**.
- In the **Serializer Input** section, select the location of the logical instance data that you want to test-serialize:
  - Select **Content from 'DFDL Test - Logical Instance'** if you populated the **DFDL Test - Logical Instance** view with data.
  - Select **Content from a logical instance file** to select a data file from your workspace or from the file system.

As an option, you can select **Validate data against schema** to check for validation errors.

The **DFDL Test Perspective** opens, and the results of your test-serialize are displayed in the **DFDL Test - Serialize** view.



## DFDL points of uncertainty

- DFDL parser is a recursive-descent parser; it uses look-ahead when it encounters “points of uncertainty,” such as:
  - A choice
  - An optional element
  - A variable array of elements
- Parser attempts to parse data until an object is either “known to exist” or “known not to exist”
  - When a parsing error occurs, the parser attempts to suppress the error, backtrack, and try another alternative
  - Use a discriminator annotation to assert that an object is “known to exist” to prevent unnecessary backtracking during parsing
  - Use an initiator to assert that an object is “known to exist”

© Copyright IBM Corporation 2015

Figure 6-47. DFDL points of uncertainty

WM666 / ZM6661.0

### Notes:

A DFDL parser is a recursive-descent parser that uses lookahead to resolve points of uncertainty such as a choice, an optional element, or a variable array of elements.

At points of uncertainty, the parser must speculatively try to parse data until an object is either 'known to exist' or 'known not to exist'. Until the condition is met, a processing error causes the parser to suppress the error, backtrack, and try an alternative.

To prevent incorrect backtracking, use a discriminator to assert that an object is 'known to exist'. Initiators are also able to assert that an object is 'known to exist'. If all alternatives have an initiator, set **Initiated Content** property to **Yes** on the sequence or choice element, instead of using a discriminator.



## Getting started with DFDL

- DFDL home page:

<http://www.ogf.org/dfdl/>

- DFDL 1.0 specification:

<http://www.ogf.org/documents/GFD.174.pdf>

- DFDL schemas on GitHub

<https://github.com/DFDLSchemas>

- IBM DeveloperWorks: Get Started with Data Format Description Language

<http://www.ibm.com/developerworks/library/se-dfdl/>

© Copyright IBM Corporation 2015

Figure 6-48. Getting started with DFDL

WM666 / ZM6661.0

### Notes:

If you want more information about DFDL, consult the DFDL home page and the DFDL specification.

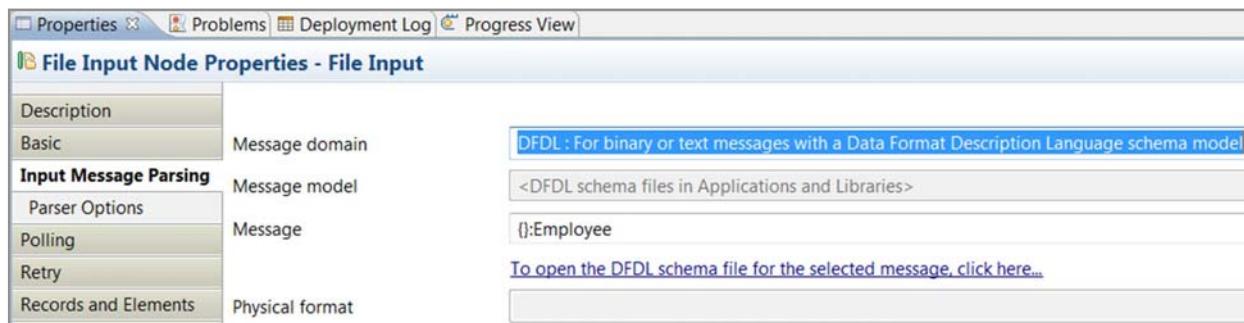
An IBM DeveloperWorks site contains video tutorials and more information about the IBM DFDL implementation.

The IBM Knowledge Center for IBM Integration Bus also describes the use of DFDL and the IBM extensions to it.



## Assigning the DFDL parser to a message

- IBM Integration Bus uses `Root.Properties` to determine the parser and the dictionary for parsing (In) and for serialization (Out)
- Which information is needed?
  - Message domain
  - Message



© Copyright IBM Corporation 2015

Figure 6-49. Assigning the DFDL parser to a message

WM666 / ZM6661.0

### Notes:

If the message flow must work with the message content, you must assign a parser to the message. You generally assign the parser at design time. In the example, the parser (message domain) is set to **DFDL** on the **Input Message Parsing** tab in the File Input node properties. If you use the DFDL parser, you must also specify the DFDL schema to use to parse the data in the **Message** property.

A parser can be assigned dynamically at run time.

## DFDL message tree

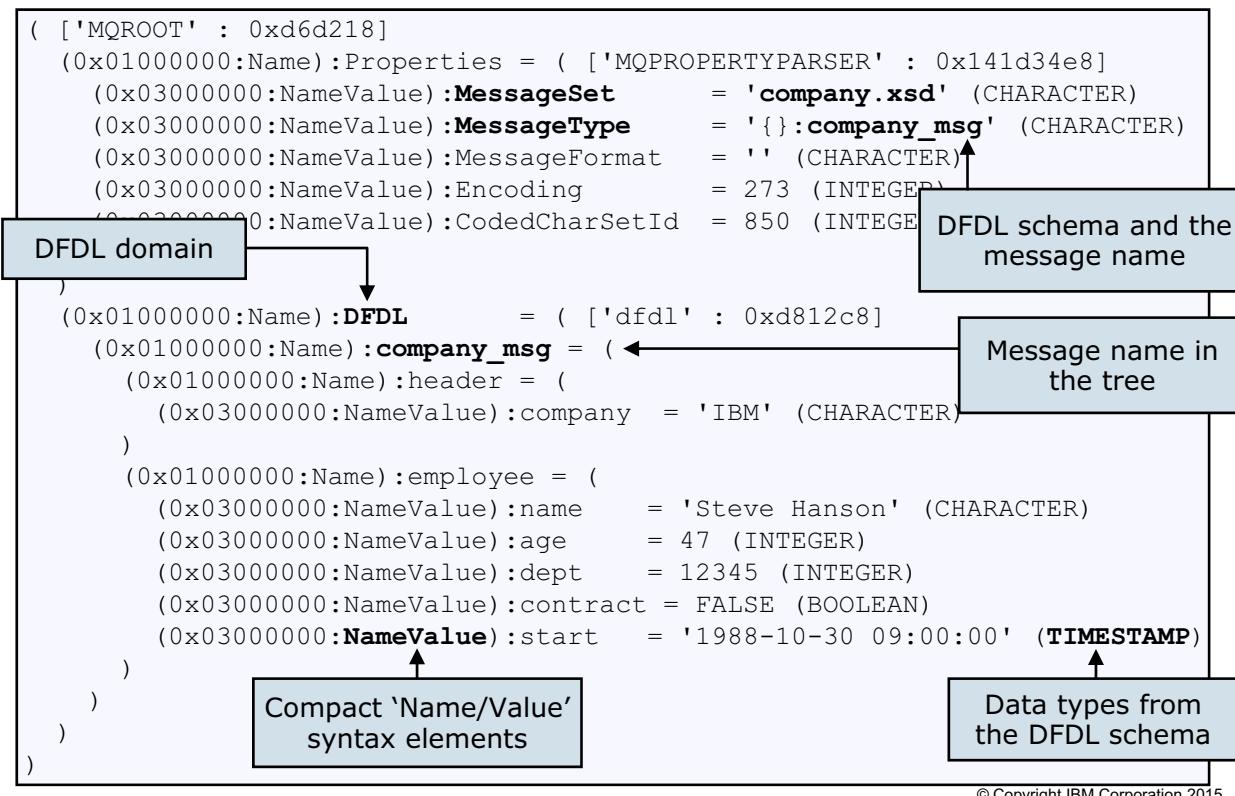


Figure 6-50. DFDL message tree

---

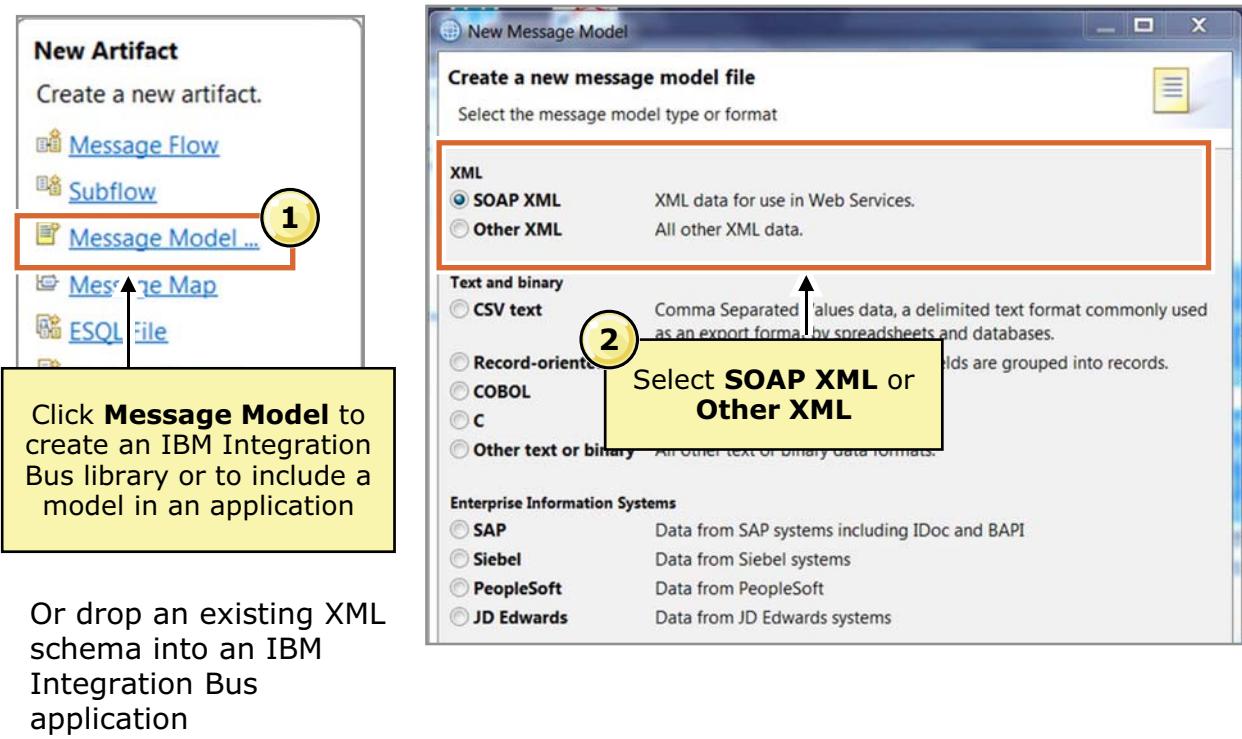
WM666 / ZM6661.0

### **Notes:**

This figure shows an example of the DFDL message tree as it is shown in a trace. The **MessageSet** object in the trace is the name of the DFDL schema file. The **MessageType** is the top-level element in the schema.



## Creating an XML message model



Click **Message Model** to create an IBM Integration Bus library or to include a model in an application

Or drop an existing XML schema into an IBM Integration Bus application

© Copyright IBM Corporation 2015

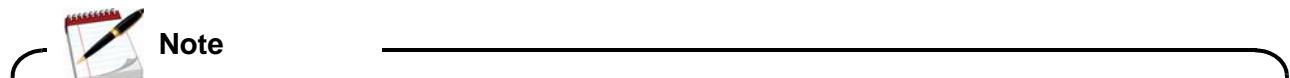
Figure 6-51. Creating an XML message model

WM666 / ZM6661.0

### Notes:

You can create an XML model for SOAP XML or other XML in the IBM Integration Toolkit:

1. Click **Message Model** from the New Artifact options or click **File > New > Message Model**.
2. Select **SOAP XML** for XML data for use in web services or **Other XML** for all other XML data.



If you have an XML schema that describes the data, it is not necessary to create a message model by using the Message Model wizard.

## Parsing XML data

- In most cases, use the XMLNSC parser
  - Fast memory-saving parser
  - Builds a compact message tree with options to preserve the white space
  - Supports opaque parsing where an opaque element (an element in a message that a message flow never references) and its subtree are parsed as a single string; reduces parse time
  - Can operate both as a programmatic and a model-driven parser
  - An XML schema is created from the message set and deployed to the integration node
  - Optional validation compliant with the XML Schema V1.0 specification
- Use MRM only with the wire format XML for non-XML data that is parsed by MRM CWF or TDS, which is rendered as XML with no further transformation
- Use XMLNS only if the message tree must conform as closely as possible to the XML data model (for example, by using certain XPath expressions), or to preserve the inline DTDs

© Copyright IBM Corporation 2015

Figure 6-52. Parsing XML data

WM666 / ZM6661.0

### Notes:

You can model XML messages by using the XML importer or by creating an MRM model. At run time, you can specify the XMLNSC parser or the MRM parser.

The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML schema validation. The XMLNSC parser has a range of options that make it suitable for most XML processing requirements. Some of these options are only available in the XMLNSC parser.

Although the XMLNSC parser can parse XML documents without an XML schema, extra features of the parser become available when it operates in model-driven mode.

The XMLNSC parser is the most efficient, so even though a message is modeled in the MRM, the XMLNSC parser can still be used at run time. To use the XMLNSC parser at run time, specify the default message domain XMLNSC in the `messageSet.mset` file.

The MRM parser must be used with a message dictionary that is generated from a message set. This message dictionary enables the MRM parser to provide the following capabilities.

For more information about the features of the XMLNSC, XMLNS, and MRM parsers, see “Which parser should you use” in the IBM Knowledge Center.



## Configuring nodes for an XMLNSC parser (1 of 2)

The screenshot shows the 'MQInput Node Properties - MQInput' dialog. On the left, a sidebar lists tabs: Description, Basic, Input Message Parsing (which is selected and highlighted with a red box), Parser Options, Advanced, Validation, and Security. The main panel displays properties for the selected tab:

- Message domain:** XMLNSC : For XML messages (namespace aware, validation, low memory use)
- Message set:** TestSOAP (ELC0510002001)
- Message type:** [empty]
- Message format:** [empty]

A callout box points to the 'Input Message Parsing' tab with the text: "Select the XMLNSC domain and the message set on the **Input Message Parsing** tab".

The screenshot shows the same dialog, but the 'Validation' tab is now selected and highlighted with a red box. The main panel displays properties for the Validation tab:

- Validate:** None
- Failure action:** Exception

A callout box points to the 'Validation' tab with the text: "Enable validation to parse and validate against the XML schema in the **Validation** tab".

© Copyright IBM Corporation 2015

Figure 6-53. Configuring nodes for an XMLNSC parser (1 of 2)

WM666 / ZM6661.0

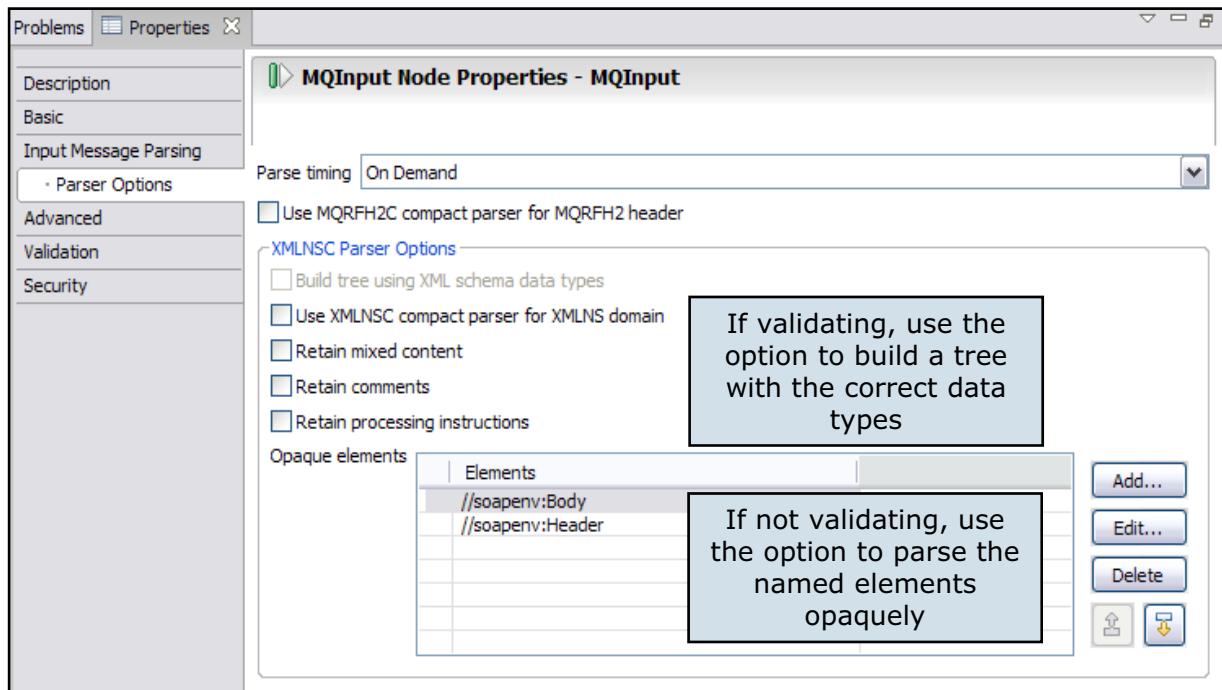
### Notes:

The XMLNSC domain can validate against real XML schema. If XMLNSC is operating in validating mode, it has the XML schema and can identify the XML schema data type of the XML data. Syntax elements are built with a data type that matches the XML schema data type.

The **Validate** property on the **Validation** tab determines whether to validate the data against the model. If validation is selected, you must supply the location of the schema on the **Input Message Parsing** tab on the **Message set** property. The **Message type** property is not required because the name of the root element in the document is used. The **Message format** property is not required because the XMLNSC parser uses the logical model.



## Configuring nodes for an XMLNSC parser (2 of 2)



© Copyright IBM Corporation 2015

Figure 6-54. Configuring nodes for an XMLNSC parser (2 of 2)

WM666 / ZM6661.0

### Notes:

When validation is disabled, you can use *opaque parsing*, which parses specific elements, which improves performance.

With opaque parsing, certain named XML elements are not fully parsed. They are skipped; instead, and the raw bit stream is inserted into the tree as a Unicode string. The **Parser Options** tab identifies the opaque parsing elements by using XPath.

Do not parse an element opaquely in any of the following cases:

- The message flow must access one of its child elements.
- The message flow changes the namespace prefix in a way that affects the opaque element or one of its child elements and the element is copied to the output bit stream.
- The element, or any child element, contains a reference to an internal entity that is defined in an inline DTD and the element is copied to the output bit stream.
- The element contains child attributes that have default values that are defined in an inline DTD and the element is copied to the output bit stream.

## Message validation

- **What** can be validated?
  - Runtime data is validated against a dictionary or an XML schema
  - Content: Cardinality, complex type content, composition, and data type
  - Value: Data fields conform to the value constraints in a message model
- **When** does validation occur?
  - Validate an **incoming** message in the **XXinput** node
  - Validate a constructed **output** message in the **XXoutput** node
  - Validate immediately or request validation when a message is converted to a bitstream
- **Failure** action choices
  - Throw an **Exception** on the first validation failure and stop validation
  - Log all validation failures in the **ExceptionList** and throw an **Exception** after completion of a parse or a write operation
  - Log all validation failures in **Usertrace** or **LocalErrorLog** and continue

© Copyright IBM Corporation 2015

Figure 6-55. Message validation

WM666 / ZM6661.0

### Notes:

Message flows are designed to transform and route messages that conform to certain rules. By default, most parsers validate some of the message when it is parsed to ensure the integrity of the parsing operation. It is possible to do more stringent validation of the message against the message model by specifying validation options on some nodes in the message flow.

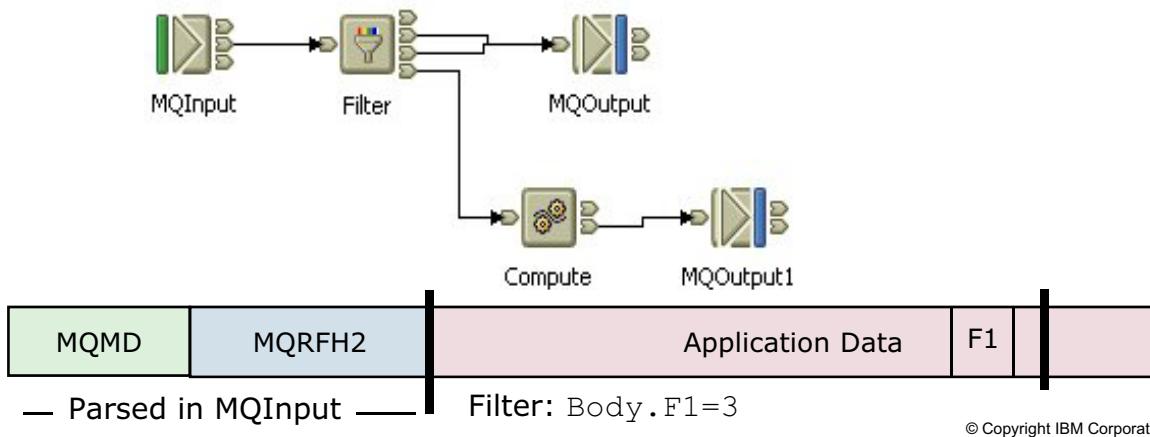
The integration node provides validation that is based on the message models for predefined messages that are modeled and deployed to the integration node. The MRM and IDOC parsers validate predefined messages against the message dictionary that is generated from a message set. The XMLNSC and SOAP domains validate predefined messages directly against XML schema that is generated from a message set.

Validation options apply to the message body only. Validation does not fix any problems. Validation is a costly operation, so consider whether to do it centrally by the integration node, or the sending/receiving application.

The **Failure action** on the node identifies the actions to take if an error occurs.

## Parse timing option on input nodes

- Parsing is an expensive operation
- On demand (default):
  - When a field is first referred in ESQL
  - Only up to this field
  - Debugger or Trace node ( `${Body}` ) forces parsing of the entire body
- Complete: Validate everything
- Immediate: Validate CHOICE and nested messages later



© Copyright IBM Corporation 2015

Figure 6-56. Parse timing option on input nodes

WM666 / ZM6661.0

### Notes:

An input message can be of any length. To improve the performance of message flows, a message is parsed only when necessary to resolve the reference to a particular part of its content. If none of the message content is referenced within the message flow, the message body is not parsed. This parse type is known as *partial* or *on-demand* parsing.

*Complete* parsing and *immediate* parsing both fully parse the input message, regardless of which fields in the message are referenced. The difference between the two parse methods occurs when validation is enabled on an element that cannot be resolved at parse time is encountered.

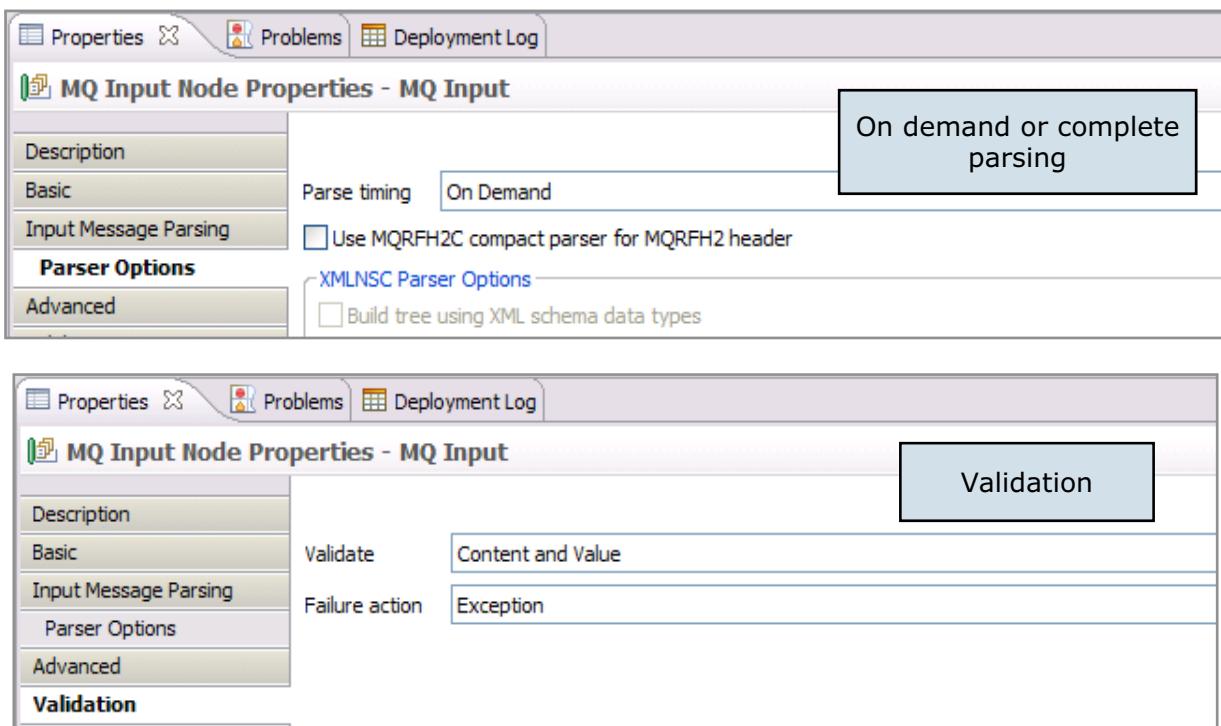
If you are using the debugger or a Trace node that specifies  `${Body}` , a complete parse of the message also occurs. A complete parse can trigger errors that previously were undetected if the entire message was not fully parsed previously.

If a parser can parse an input bit stream on demand, the **Parse Timing** property of a message flow node controls the on-demand behavior of the parser.

Parsing is one of the most costly operations at run time in IBM Integration Bus. It is a good practice to parse only the data that is required to process a message. Parsers that use on-demand or partial parsing can help to reduce the amount of processor and memory that is needed at run time.

# WebSphere Education

## Parsing and validation



© Copyright IBM Corporation 2015

Figure 6-57. Parsing and validation

WM666 / ZM6661.0

### Notes:

Message flows are designed to transform and route messages that conform to certain rules. By default, parsers do some validity checking on a message, but only to ensure the integrity of the parsing operation. However, you can validate a message more stringently against the message model that is contained in the message set by specifying validation options on certain nodes in your message flow.

Validation applies only to messages that you modeled and deployed to the integration node. Specifically, the message domains that support validation are DFDL, MRM, XMLNSC, SOAP, and IDOC.

The integration node does not provide any validation for self-defining messages. The MRM and IDOC parsers validate predefined messages against the message dictionary that is generated from a message set. The DFDL, XMLNSC, and SOAP domains validate predefined messages directly against an XML schema that is generated from a message set.

Along with specifying parsing information about the input node of a message flow, it is possible to request validation as well. You can use the following validation options, if you enable validation:

- **Content** indicates that you want to check content, such as content validation and composition.

- **Content and Value** indicate that you want to check content, such as Content validation and Composition, and value checks, such as whether the value conforms to data type, length, range, and enumeration.



**Note**

Even if **Content** is selected, the SOAP and XMLNSC domains always do **Content and Value** validation.

Many nodes provide a validation option; you can choose to enable validation at many points during a message flow. If you do enable validation at multiple points in the message flow, give careful consideration as to how you handle validation errors that might occur.

## Default and fixed values for elements

- Logical property of an element
- IBM Integration Bus applies default or fixed values only when necessary to create a parsable bit stream
  - 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
- Or use ResetContentDescriptor node to serialize data by using properties and then reparse by using the message domain that is supplied in node properties

© Copyright IBM Corporation 2015

Figure 6-58. Default and fixed values for elements

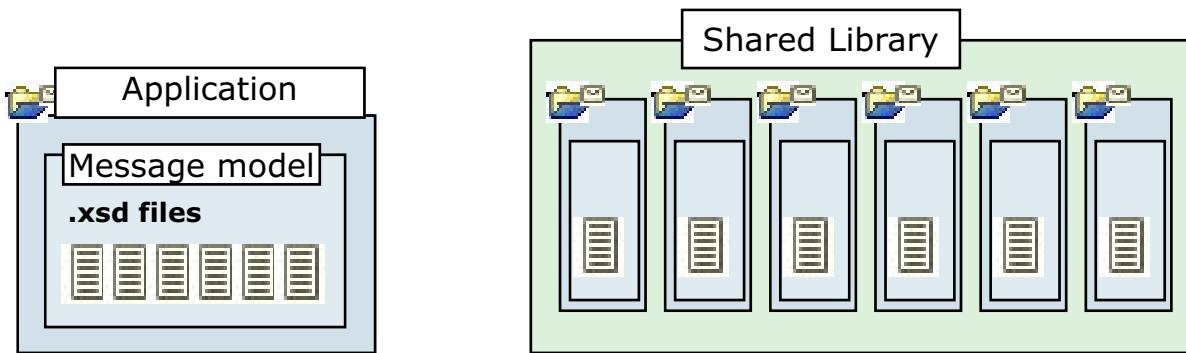
WM666 / ZM6661.0

### Notes:

Earlier in this unit, 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 parses the message by using a different parser. 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.

## Organizing message models



- Integration Bus application provides runtime isolation so that the visibility of a resource is restricted to the application in which it is contained
- Use applications to ensure that updates to one group of deployed resources do not affect another group
- Consider the use of shared libraries to share message models across multiple teams, projects, or integration nodes

© Copyright IBM Corporation 2015

Figure 6-59. Organizing message models

WM666 / ZM6661.0

### Notes:

You can create your message models in applications or in libraries. The method that you use depends on your implementation.

If you are creating message models that many applications use, create them in libraries and then reference them at run time.

Store the message model in the application to ensure that updates to the model are isolated and do not affect other message flows.

## Unit summary

Having completed this unit, you should be able to:

- Explain the concepts of message models and how they are used to help message transformation
- List the parsers that are available for use within IBM Integration Bus
- Create and modify a DFDL model
- Use importers to create data models
- Choose the appropriate message validation options
- Organize and administer message models
- Reference message models in message flows

© Copyright IBM Corporation 2015

Figure 6-60. Unit summary

WM666 / ZM6661.0

### Notes:



## Checkpoint questions

1. True or false: You must deploy the DFDL “Helper” schema with the user DFDL schema at run time.
2. True or false: When writing a message, the DFDL parser generates a DFDL formatted bit stream from a DFDL domain logical message tree.

© Copyright IBM Corporation 2015

Figure 6-61. Checkpoint questions

WM666 / ZM6661.0

### Notes:

Write your answers here:

1.

2.



## Checkpoint answers

1. True or false: You must deploy the DFDL “Helper” schema with the user DFDL schema at run time.

**Answer: True.**

2. True or false: When writing a message, the DFDL parser generates a DFDL formatted bit stream from a DFDL domain logical message tree.

**Answer: False.** The DFDL serializer generates a DFDL formatted bit stream.

© Copyright IBM Corporation 2015

Figure 6-62. Checkpoint answers

WM666 / ZM6661.0

### Notes:

## Exercise 5



© Copyright IBM Corporation 2015

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

10.1

Figure 6-63. Exercise 5

WM666 / ZM6661.0

### Notes:

## Exercise objectives

After completing this exercise, you should be able to:

- Create a DFDL message model schema file in a shared library
- Define the logical structure and physical properties of the message model elements
- Test a DFDL schema by parsing test input data
- Test a DFDL schema by serializing test data to create an output file

© Copyright IBM Corporation 2015

Figure 6-64. Exercise objectives

WM666 / ZM6661.0

### Notes:

See the Student Exercises Guide for this course for detailed instructions.



# Unit 7. Processing file data

## What this unit is about

In this unit, you learn how to use IBM Integration Bus to process file data.

## What you should be able to do

After completing this unit, you should be able to:

- Describe the file processing nodes
- Describe the record detection options for splitting files into multiple records
- Use a file as a message flow source and target
- Include file input and output nodes that use File Transfer Protocol (FTP) and secure FTP (SFTP) to transfer data

## How you will check your progress

- Checkpoint questions
- Lab exercise

## References

IBM Knowledge Center



## Unit objectives

After completing this unit, you should be able to:

- Describe the file processing nodes
- Describe the record detection options for splitting files into multiple records
- Use a file as a message flow source and target
- Include file input and output nodes that use File Transfer Protocol (FTP) and secure FTP (SFTP) to transfer data

© Copyright IBM Corporation 2015

Figure 7-1. Unit objectives

WM666 / ZM6661.0

### Notes:



## File processing nodes

- Generic
  - FileInput
  - FileRead
  - FileOutput
- IBM Sterling Connect: Direct
  - CDInput
  - CDOOutput
- IBM MQ File Transfer Edition
  - FTEInput
  - FTEOutput

© Copyright IBM Corporation 2015

Figure 7-2. File processing nodes

WM666 / ZM6661.0

### Notes:

The integration node reads files with the FileInput, FileRead, FTEInput, and CDInput nodes. It writes files with the FileOutput, CDOOutput, and FTEOutput nodes.

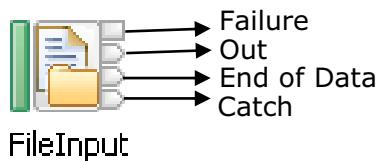
You can use the FTEInput and FTEOutput nodes to receive or send files to a destination on a WebSphere MQ File Transfer Edition network.

You can use the CDInput and CDOOutput nodes to receive or send files to a destination on an IBM Sterling Connect:Direct network.

This unit describes the generic file nodes: FileInput, FileGet, and FileOutput.

## FileInput node

- Receives messages from files in the file system of the integration node or by using FTP or SFTP, in a remote file system
- Starts the message flow when a new file arrives
- Focus is on processing rather than message delivery
- If connected, the **End Of Data** terminal is triggered after all the messages in a file are processed, with an empty BLOB message and a `LocalEnvironment.File` structure



© Copyright IBM Corporation 2015

Figure 7-3. FileInput node

WM666 / ZM6661.0

### Notes:

It is possible for the integration node to read and write files from and to the local file system. It is also possible to read and write files to and from an FTP server. The focus of the file support in IBM Integration Bus is on message processing, rather than message delivery.

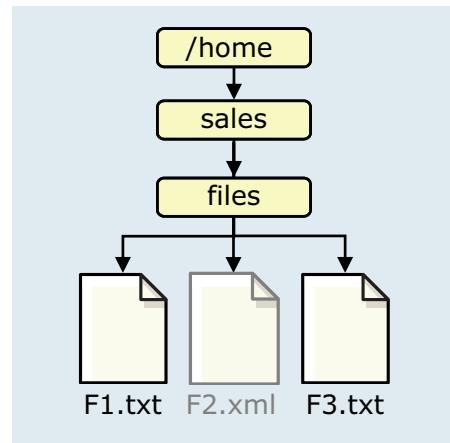
Support for the FileInput and FileOutput nodes is available on all supported operating systems. On z/OS, files are resolved in the HFS or ZFS file system, and file names are treated in the case-sensitive UNIX manner.

There are specific terminals, both on the File Input and File Output nodes, for handling the end of the file. On the File Input node, the **End of Data** terminal is given control when the integration node detects that it reaches the end of the input file. The End of Data message comprises an empty BLOB message together with a `LocalEnvironment.File` structure. It allows explicit end-of-flow processing in another part of the flow.

On the FileOutput node, the **Finish File** terminal is driven to indicate to write the file to the file system.

## FileInput node: Basic algorithm

- Scans a preconfigured directory for files that match a particular specification  
Example:  
Input directory: /home/sales/files  
File name or pattern: \*.txt
- Oldest files are processed first
- When the directory is parsed, the algorithm repeats
- Select **Include local subdirectories** to process the files under the specified **Input directory**
- The **mqsitransitin** subdirectory holds and locks input files while they are being processed
  - Check this subdirectory for partially processed or unprocessed files



© Copyright IBM Corporation 2015

Figure 7-4. FileInput node: Basic algorithm

WM666 / ZM6661.0

### Notes:

The FileInput node can read through a preconfigured directory on the file system of the integration node. The directory name is configured on the **Basic** tab of the node and must be entered in the format that the integration node file system expects. The integration node scans the directory for files. By default subdirectories of the directory are not scanned but you can configure FileInput node to include local subdirectories.

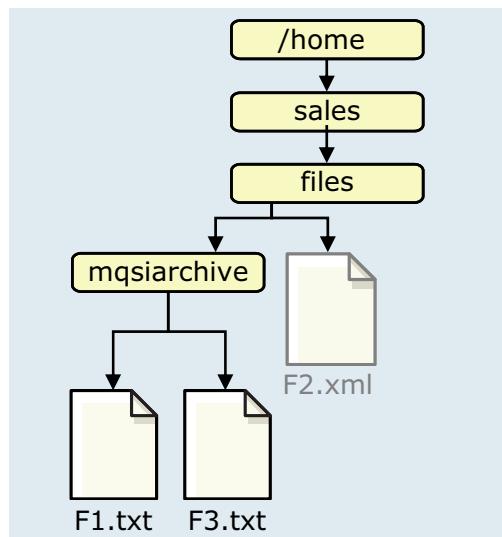
The files to process must be present in the specified directory. The file name can be specified as a wildcard. Acceptable wildcards on the file name pattern are an asterisk (\*), which matches multiple characters, and a question mark (?), which matches a single character. The value for the directory must either be a fully qualified directory path, or a directory relative to \$MQSI\_WORKPATH/file.

After the first file in the directory is processed, the next file in the directory is processed. After all of the files are processed, the directory is rescanned. If there is an operating system lock on a file, it is not read until the lock is removed. If any files that were previously locked are now unlocked, the integration node processes them.

If a directory scan yields no matches, the node waits for a preconfigured amount of time before rescanning. The default value for this interval is 5 seconds.

## FileInput node: Successful processing

- File is either deleted or moved to a `mqsiarchive` subdirectory
- LocalEnvironment contains details on the file:
  - `LocalEnvironment.File.Directory`, `Name`, `LastModified`, `TimeStamp`
  - `LocalEnvironment.Wildcard.WildcardMatch`



© Copyright IBM Corporation 2015

Figure 7-5. FileInput node: Successful processing

WM666 / ZM6661.0

### Notes:

After the file is parsed, an option on the node specifies what to do with file. The options are to delete the input file, or move it to an archive subdirectory of the input directory. If the archive subdirectory (`mqsiarchive`) does not exist, it is created.

If an archive file exists, a property on the FileInput specifies the actions to take. If you do not set the option to overwrite the file, the node generates an exception if it tries to move a file that is successfully processed with the same name into the archive subdirectory. To prevent duplicate file names, you can select the options to rename the file and include time stamp value in the name. The time stamp includes the time to millisecond granularity, ensuring the uniqueness of file names in the archive directory. When the file is placed into the archive directory, the contents of the file are not altered.

Each time the contents of a file are propagated, the FileInput node stores information about the file inside the Local Environment, which is available in subsequent nodes in the message flow. Also, the FileInput node copies the characters in the file name that the wildcards match, together with any intermediate characters, to the **WildcardMatch** element. Thus, if the match expression was `file*.txt` and the file read was `file02.txt`, then the value of **WildcardMatch** would be `02`. An output node can use this information, where the output name can be derived from the input name.

## Configuring the FileInput node

1. On the **Basic** tab, enter the directories and files that the FileInput node processes and action to take if duplicate files exist
2. On the **Input Message Parsing** tab, set values for the properties that the node uses to determine how to parse the incoming message
3. On the **Polling** tab, enter a value for the polling interval
4. On the **Retry** tab, define how retry processing is done when a message flow fails
5. On the **Records and Elements** tab, specify how each file is interpreted as records

© Copyright IBM Corporation 2015

Figure 7-6. Configuring the FileInput node

WM666 / ZM6661.0

### Notes:

The FileInput node reads raw bytes from the file that came from the file system before passing it on to the parser. The **Input Message Parsing** property tab on the FileInput node has two nonstandard fields for handling character data: **Message coded character set ID** and **Message Encoding**. These properties make it possible to parse information from an operating system other than the one in which the integration node is running.

At most, one instance and node processes each file. More instances and nodes might concurrently process other files in the same directory. The FileInput node has an **Instances** tab to enable the use of a dedicated thread pool for the node. This feature is useful in message flows that have multiple input nodes, as the pool of threads of the flow can lead to the starvation of one or more input nodes. In particular, FTP transfers might take a significant amount of time to complete, during which time the thread is busy. For this reason, it is important to tune the additional instances parameter to maximize throughput rates. If no files are found during the directory scan, the node transfers files from the configured FTP server to the local directory that is being used for file input. Transferred files are deleted from the FTP server.

## Record detection

- FileInput node can split each file into separate records, and the message flow is then started multiple times
- Only necessary for one record in memory at any one time
  - Allows for efficient streaming of large files
  - Streaming possible only with DFDL, XMLNSC, and MRM (CWF and TDS) parsers
- Handling options on the **Records and Elements** tab

© Copyright IBM Corporation 2015

Figure 7-7. Record detection

WM666 / ZM6661.0

### Notes:

The FileInput node can split each file into multiple records that are propagated separately.

One particular benefit of record detection is that, depending on the parser that is being used, the FileInput node does not need to store the whole file in memory at any one time. Only the records currently being processed are stored in memory. This streaming allows for large files to be processed. Such files can be several gigabytes in size. The DFDL, MRM, and XMLNSC parsers support streaming.

Record detection allows a file to be split into individual parts by using several techniques. Alternatively, the whole file can be processed as a single entity. Also, a single thread only within the integration node can process a file. Therefore, individual records can be processed only sequentially, not in parallel.

## Record detection options

- **Whole file:** Each file propagates a single message
- **Fixed length:** Specify the length of each record in bytes up to 100 MB
- **Delimited**
  - Integration node system line-end or a custom delimiter
  - Infix versus postfix
- **Parsed record sequence**
  - File contains one or more structures that the parser that is specified on the **Inbound Message Parser** properties can parse
  - FileInputStream node propagates each matching structure as a separate message

© Copyright IBM Corporation 2015

Figure 7-8. Record detection options

WM666 / ZM6661.0

### Notes:

**Whole File** reads the entire contents of the file into memory and parses it in its entirety, according to the **Inbound Message Parsing** options. If this option is used, you must ensure that the available memory in the integration node is sufficient to accommodate the whole file.

**Fixed Length** breaks the file into a set number of bytes and propagates each chunk as a separate message. The last message in any file can be smaller than the specified length. If you use this option, specify the record size on the FileInputStream node.

**Delimited** specifies a delimiter that is used to break the records in a file. In this case, you can specify a standard DOS or UNIX end-of-line delimiter, or a custom delimiter by specifying the hexadecimal characters that indicate the end of line for the particular file. If you specify **Delimiter type=Infix** and the last data in the file ends with a delimiter, the FileInputStream node propagates an empty record to indicate the presence of the delimiter. If you specify **Delimiter type=Postfix**, the FileInputStream node does not propagate an empty record, whether the last data in the file ends with a delimiter.

**Parsed Record Sequence** breaks the file into messages, each of which conforms to a single structure as defined on the **Inbound Message Parsing** options.



## Record detection examples

```
Belgian Bun|10|Jam tart|9|Gingerbread man|8|
```

- Whole file (input message parsing = ' | ' delimited file) propagates one message:  
"Belgian Bun|10|Jam tart|9|Gingerbread man|8 |"
- Fixed length (size = 10 bytes) propagates five messages:  
"Belgian Bu", "n|10|Jam t", "art|9|Ging", "erbread ma",  
"n|8 |"
- Delimited (character = ' | ')
  - Infix: Propagates seven messages  
"Belgian Bun", "10", "Jam tart", "9", "Gingerbread man",  
"8", ""
  - Postfix: Propagates six messages  
"Belgian Bun", "10", "Jam tart", "9", "Gingerbread man", "8"
- Parsed record sequence with the parser set to XMLNSC propagates three messages:

```
<cakes><cake name="belgian bun" rating="10"></cakes>
<cakes><cake name="jam tart" rating="9"></cakes>
<cakes><cake name="gingerbread man" rating="8"></cakes>
```

© Copyright IBM Corporation 2015

Figure 7-9. Record detection examples

WM666 / ZM6661.0

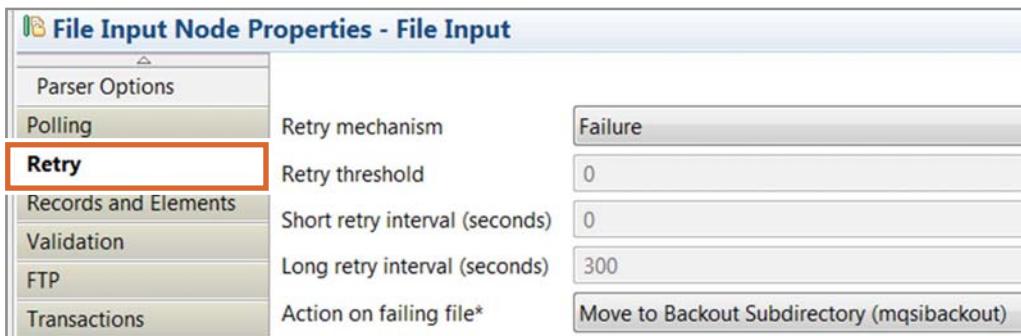
### Notes:

By using the **Record detection** options, an example file, which is shown at the top of the figure, can be processed in different ways.

- The file can be processed as a whole file. In this case, the entire file is passed to the message flow as a single piece of data in the message tree.
- The file can be split into sections of 10 bytes each. In this case, the FileInput node propagates five messages to the flow: the first four messages have 10 bytes each. The last message has 4 bytes.
- The file can use a delimited parser, with a vertical bar as a delimiter. In this case, if the **Delimiter Type** is infix, seven messages are propagated, where the last record is an empty record. If the **Delimiter Type** is postfix, six messages are propagated.

For the parsed record sequence example, the input file content is not a single valid XML document, but rather 3 distinct XML documents that are propagated separately. On their own, the contents of the file would not pass a validation test of the XML. But, but when it is split into multiple instances of XML, each single instance would be valid. In this case, if the parser is set to XMLNSC, then three messages are propagated to the message flow.

## FileInput node error handling



- For failures, the node attempts to reprocess a configurable number of times
- If processing still fails, the **Action on failing file** option is to move the file to the `mqsi-backout` subdirectory or delete the file
- If an attempt to write to `mqsi-backout` fails, the message flow stops
- If the node fails before the message is propagated to the **Out** terminal, the message is sent to the **Failure** terminal
- Transaction mode** property on **Transactions** tab defines whether the message flow outputs messages under a coordinated transaction
  - Completed writes are not backed out if a failure occurs

© Copyright IBM Corporation 2015

Figure 7-10. FileInput node error handling

WM666 / ZM6661.0

### Notes:

Each FileInput node has a configurable **Retry Mechanism**:

- Failure** (default): If the file cannot be read, then it is processed as a failure.
- Short Retry**: The integration node tries a number of times before processing as a failure.
- Short and long retry**: After the number of short tries are exhausted, the node continues to try indefinitely, waiting for the **Long retry interval** period between tries.

The **Action on failing file** option determines failure processing:

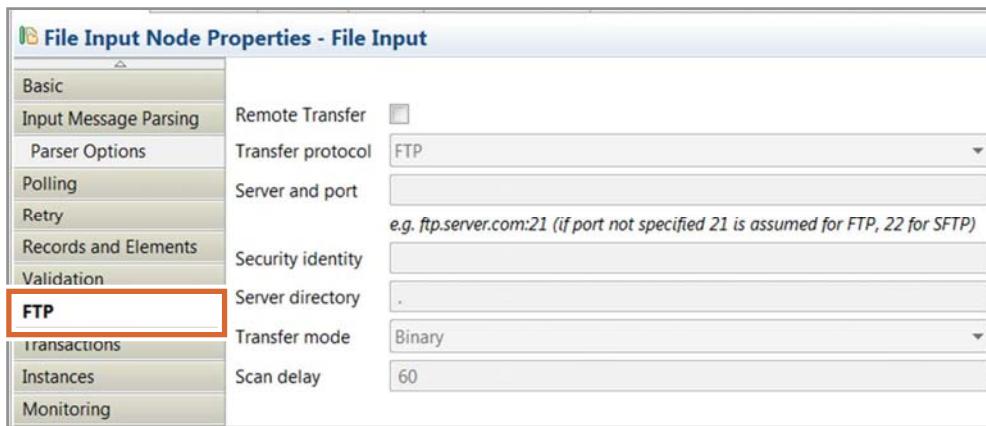
- Move to Backout Subdirectory** writes the file to a backout subdirectory, which is created as a subdirectory of the input directory. If a file of that name exists in the backout directory, the message flow fails, and the failure is reported.
- Delete** deletes the input file and processes the next file to process.
- Add timestamp and move to Backout Subdirectory** renames the failing file so it includes a time stamp. This process helps ensure that duplicate files cannot exist in the `mqsi-backout` subdirectory.



## FileInput node that uses FTP

- On the **FTP** tab, select the **Remote Transfer** property and then select the **Transfer Protocol** to read files from an FTP or SFTP server
- When active, FTP settings cause the node to periodically transfer files on a remote server to the local directory for input
- Security identity is set by using the `mqsisetdbparms` command:

```
mqsisetdbparms InteNodeName -n ftp::Security_identity
                  -u ftpUser -p ftpPassword
```



© Copyright IBM Corporation 2015

Figure 7-11. FileInput node that uses FTP

WM666 / ZM6661.0

### Notes:

It is possible to use the FileInput to read files from an FTP server. As shown in the figure, FTP, or SFTP support is enabled by selecting **Remote Transfer** on the **FTP** tab of the FileInput node. The **Transfer protocol** property is **FTP** by default. If you want to use SFTP, change the **Transfer protocol** property to **SFTP**.

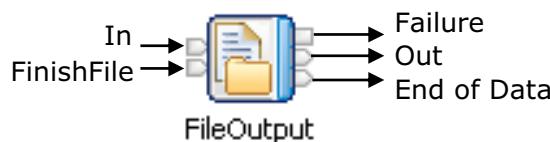
When using the FTP/SFTP function, the integration node operates in a way that is similar to normal file processing. The FTP server directory is scanned for new files, and any files that are detected are moved into the local file directory that is specified on the FileInput node. The integration node then processes these files as a local file.

The security authorization for the FTP server can be specified by using the **Security identity** property on the **FTP** tab. The value that is specified for this option must match the parameter that is specified on the `mqsisetdbparms` command. When using the `mqsisetdbparms` command, note the format of the `ftp` parameter. A double colon (:) must follow `ftp`, which is then followed by the name that is specified in the **Security identity** property on the **FTP** tab of the File node. For example:

```
mqsisetdbparms IBNODE -n ftp::SecurIdent -u user1 -p MyPassword
```

## FileOutput node

- Writes messages to a file in the file system of the integration node or by using FTP or SFTP in a remote file system
- Values in the **Record definition**, **Delimiter**, and **Delimiter type** properties create a single file from multiple messages
- Writes accumulated messages to a file, and places it in the output directory at either of the following times:
  - After each record, if the file is to contain a single record
  - When the **Finish File** terminal receives a message



© Copyright IBM Corporation 2015

Figure 7-12. FileOutput node

WM666 / ZM6661.0

### Notes:

It is possible for the integration node to write files to the local file system. It is also possible to write files to an FTP server.

Support for the FileOutput node is available on all supported operating systems. On z/OS, files are resolved in the HFS or ZFS file system, and file names are treated in the case-sensitive UNIX manner.

There are specific terminals on the FileOutput node for handling the end of the file.

- The **Finish File** input terminal is driven to indicate to write the file to the file system.
- The **End of Data** output terminal is given control when the integration node detects that it reaches the end of the input file. The End of Data message comprises an empty BLOB message together with a `LocalEnvironment.File` structure. It allows explicit end-of-flow processing in another part of the flow.

## FileOutput node processing

- Received message body is written to the preconfigured file
  - File location information can be provided in the message tree
  - When writing to the output file, any wildcard is replaced with the value of `LocalEnvironment.Wildcard.WildcardMatch` to preserve elements of a file name during processing
- If the file exists, options are:
  - **Replace Existing File** (default)
  - **Append to Existing File**
  - **Fail if File Exists**: New file is created in the transit directory and exception BIP3307 is logged
  - **Archive and Replace Existing File**: Existing file is moved to the archive directory before the new file is created in the output directory
  - **Time Stamp, Archive, and Replace Existing File**: Existing file name is augmented with a time stamp and then moved to the archive directory

© Copyright IBM Corporation 2015

Figure 7-13. FileOutput node processing

WM666 / ZM6661.0

### Notes:

The FileOutput node is similar to the FileInput node in that, in its simplest form, you can specify both the directory to which files are placed and a file name expression.

The file name expression can include a single asterisk (\*) wildcard character, which is replaced with the value of the `LocalEnvironment.Wildcard.WildcardMatch` element in the tree. For example, in a FileInput to FileOutput scenario, the `WildcardMatch` element is set during the FileInput node processing. This match element can be used to preserve the name of the input file on the FileOutput node. You can also use standard tree manipulation logic in a transformation node to set the value of the `WildcardMatch` element to whatever you want, regardless of whether you used a FileInput node. For example, this value can be used to specify the name of the output file.

It is also possible to override the target directory and file name information. By default, this information is obtained from the local environment, although it can be any place in the message tree. These parameters override any hardcoded values that you specified on the **Basic** tab.

When you are writing to a file with a name that exists in the output directory, various **Output File Actions** can be selected in the **Basic** tab:

- **Replace the existing file** overwrites the existing file, with no further warning.

- **Create the file normally, but fail if the file exists** passes control to the Failure output terminal.
- **Archive any existing file before over-writing** it copies the original file to an archive directory.
- **Archive previous file, with timestamp** is similar to the previous option, but you can also change the name of the original file by adding a time stamp to the file name. As with the FileInput node options, this option avoids overwriting an existing file, since the time stamp is unique.



## FileOutput node Request tab

- If file location information is provided in the message tree:
  - Specify the location of the data in the message tree
  - Override the directory and file name on the **Basic** tab by specifying the location in the message tree as XPath or ESQL expressions

 A screenshot of a software interface titled "FileOutput Node Properties - FileOutput". On the left is a vertical toolbar with tabs: Description, Basic, Request (which is selected), Records and Elements, Validation, and FTP. The main panel contains three input fields: "Data location\*" with value "\$Body", "Request directory property location" with value "\$LocalEnvironment/Destination/File/Directory", and "Request file name property location" with value "\$LocalEnvironment/Destination/File/Name".

© Copyright IBM Corporation 2015

Figure 7-14. FileOutput node Request tab

WM666 / ZM6661.0

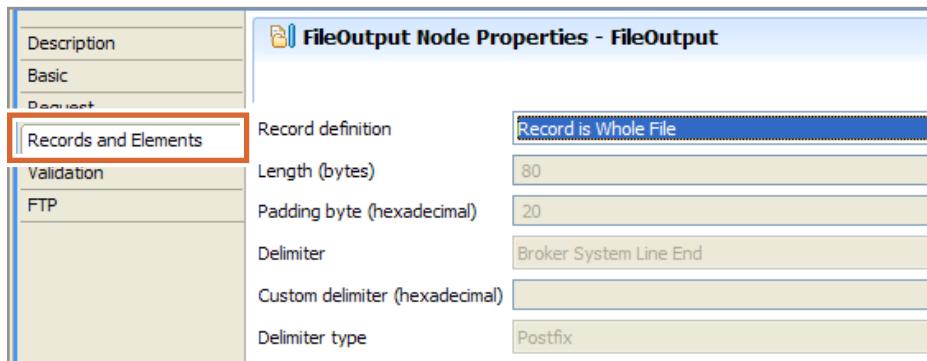
### Notes:

You can dynamically set the destination directory and file name in the message flow. By default, this information is obtained from the local environment. The node can be configured to get this information from any place in the message tree. Either the default or customized locations can then have content that is set dynamically in the message flow.

The tree element that is serialized to the output file is obtained from the value of the **Data location** property on the **Request** tab. The default value is \$Body.

## Appending records with the `FileOutput` node

- **Records and Elements** tab defines how multiple writes to the same file are handled
  - Record definition options:
    - **Record is Whole File**: Close file automatically after the first write
    - **Record is Unmodified Data**: The message bitstream is appended to the file
    - **Record is Fixed-Length Data**: Specify the length in bytes and padding character
    - **Record is Delimited Data**: Specify the delimiter and the infix/postfix option
  - Unless **Record is Whole File** is selected, the file is closed when the Finish File terminal is triggered
  - “Finish File” message is propagated to the End Of Data terminal of FileOutputStream



© Copyright IBM Corporation 2015

Figure 7-15. Appending records with the FileOutputStream node

WM666 / ZM6661.0

## **Notes:**

The **Records and Elements** tab (shown in the figure) define how the record is appended to a file.

## FileRead node



- Read one record or the entire contents of a file from within a message flow
- Provides keyed access to identify a record
- Use any parser to parse the contents of any file and propagate the contents as a message tree
- By default, after the contents of the file are successfully propagated, the file is deleted from the file system
- No other file node can access the file when the read node starts reading data from it

© Copyright IBM Corporation 2015

Figure 7-16. FileRead node

WM666 / ZM6661.0

### Notes:

You can use the FileRead node to read one record, or the entire contents of a file, from within the middle of a message flow. The FileRead node is like the FileInputStream node, which reads a file from the start of a message flow, except it is driven to read the file by an incoming message.

When the contents of the file are propagated, the file is deleted from the file system. No other file node can access the file when the read node starts reading data from it.

If a file that matches the pattern does not exist, then the original message is propagated to the 'No match' terminal. If the terminal is not connected, then an exception occurs.

At the end of processing, it is possible to configure the node not to delete the file. In this mode, any other file read node can also access the file if running in the same browse only mode.



## Configuring the FileRead node

1. On the **Basic** tab, enter the directories and files that the FileRead node processes, and select the action if duplicate files exist
2. On the **Request** tab, specify the location of the data to be read by using an XPath or ESQL expression
3. On the **Result** tab, set values for the properties that determine where the result is stored
4. On the **Input Message Parsing** tab, set values for the properties that the node uses to determine how to parse the incoming message
5. On the **Parser Options** subtab, change the **Parse timing** property to **Immediate** or **Complete**, if the entire message should be parsed immediately
6. On the **Records and Elements** tab, specify how each file is interpreted as records

© Copyright IBM Corporation 2015

Figure 7-17. Configuring the FileRead node

WM666 / ZM6661.0

### Notes:

The figure lists the steps for configuring the FileRead node. The properties are similar in function to the properties on the FileInput node.



## Unit summary

Having completed this unit, you should be able to:

- Describe the file processing nodes
- Describe the record detection options for splitting files into multiple records
- Use a file as a message flow source and target
- Include file input and output nodes that use File Transfer Protocol (FTP) and secure FTP (SFTP) to transfer data

© Copyright IBM Corporation 2015

Figure 7-18. Unit summary

WM666 / ZM6661.0

### Notes:

## Checkpoint questions

1. When does the FileOutput node write accumulated messages to a file?
  - a. When an “End of File” character is received
  - b. After each record
  - c. When the “Finish File” terminal is triggered
2. True or False: The FileOutput node can write messages to files on a network file system that is local to the integration node only.

© Copyright IBM Corporation 2015

Figure 7-19. Checkpoint questions

WM666 / ZM6661.0

### Notes:

Write your answers here:

- 1.
- 2.



## Checkpoint answers

1. When does the FileOutput node write accumulated messages to a file?

- a. When an “End of File” character is received
- b. After each record
- c. When the “Finish File” terminal is triggered

**Answer: c**

2. True or False: The FileOutput node can write messages to files on a network file system that is local to the integration node only.

**Answer: False.** The FileOutput node can also write messages to a local file system.

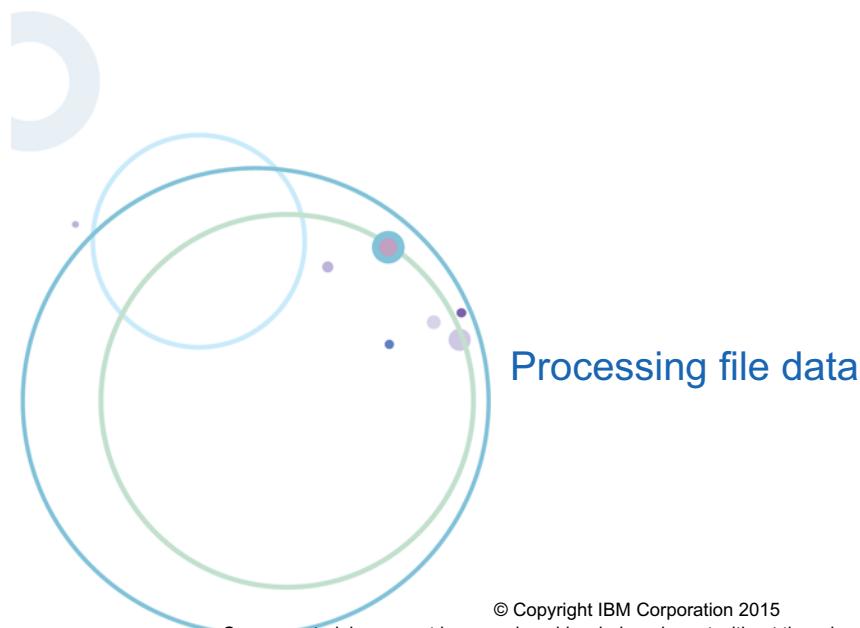
© Copyright IBM Corporation 2015

Figure 7-20. Checkpoint answers

WM666 / ZM6661.0

### Notes:

## Exercise 6



© Copyright IBM Corporation 2015

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

10.1

Figure 7-21. Exercise 6

WM666 / ZM6661.0

### Notes:



## Exercise objectives

After completing this exercise, you should be able to:

- Use a FileInput node in a message flow
- Configure the FileInput node so that each record in the file is processed as a separate transaction
- Reference a library in a message flow application
- Use the IBM Integration Bus Toolkit Flow exerciser to view multiple output messages

© Copyright IBM Corporation 2015

Figure 7-22. Exercise objectives

WM666 / ZM6661.0

### Notes:

See the *Student Exercise Guide* for detailed instructions.

# Unit 8. Using problem determination tools and help resources

## What this unit is about

In this unit, you learn about runtime errors in message flow applications. You learn how IBM Integration Bus responds to a runtime exception and what happens to the data that is being processed. You also learn how transactions can be coordinated. The unit also introduces some of the tools and techniques that IBM Integration Bus offers for problem determination and debugging, and how to support explicit error handling within a message flow application.

## What you should be able to do

After completing this unit, you should be able to:

- Use the TryCatch and Throw nodes to implement explicit error handling within a message flow
- Describe the structure of the ExceptionList component of the message assembly, and the role it plays in runtime error handling
- Use problem determination tools to debug message flows
- Use help resources to learn more about the product and find information about resolving problems

## How you will check your progress

- Checkpoint questions
- Lab exercises

## References

IBM Knowledge Center



## Unit objectives

After completing this unit, you should be able to:

- Use the TryCatch and Throw nodes to implement explicit error handling within a message flow
- Describe the structure of the ExceptionList component of the message assembly, and the role it plays in runtime error handling
- Use problem determination tools to debug message flows
- Use help resources to learn more about the product and find information about resolving problems

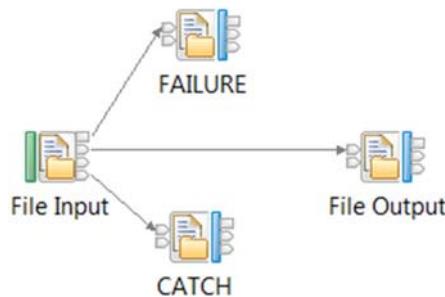
© Copyright IBM Corporation 2015

Figure 8-1. Unit objectives

WM666 / ZM6661.0

### Notes:

## Message flow result dependencies



- Wired **Failure** and **Catch** terminals
- Input node type
- **Transaction mode** property on some input node types determines whether the rest of the nodes in the flow are run under sync point

© Copyright IBM Corporation 2015

Figure 8-2. Message flow result dependencies

WM666 / ZM6661.0

### Notes:

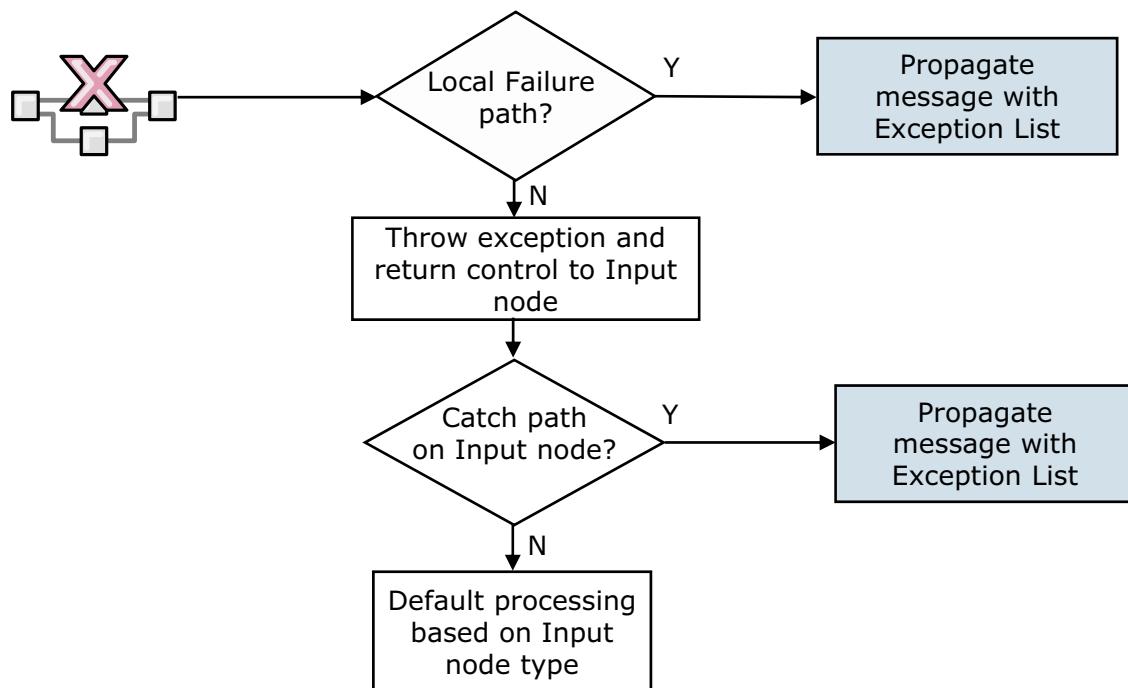
If a message flow completes successfully, messages almost always end up on the output as designed. However, depending on which other terminals are connected and if there is a problem, the message can go to other destinations.

Many factors, such as wired terminals and the input node type, can affect the path that is taken if a runtime error occurs.

Some nodes also support the **Transaction mode** property, which controls whether the incoming message is received under sync point control.

- If you select **Automatic**, the incoming message is received under sync point if it is marked persistent; otherwise, it is not received under sync point. If a failure occurs, the message is only discarded if it is marked as non-persistent.
- If you select **Yes**, the incoming message is received under sync point. If there is a failure, a rollback can occur regardless of message persistence.
- If you select **No**, the incoming message is not received under sync point. If a failure occurs, the message is discarded regardless of message persistence.

## Basic error handling



© Copyright IBM Corporation 2015

Figure 8-3. Basic error handling

WM666 / ZM6661.0

### Notes:

The figure shows the typical events that can occur when an error occurs in a processing node.

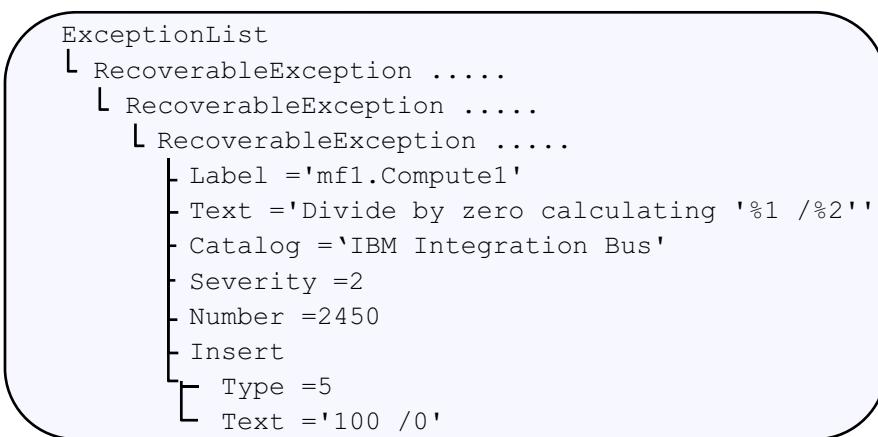
One factor that determines the events is whether a node terminal is connected.

If the **Failure** terminal of the node where the exception occurred is not connected, the next step is to check whether the **Catch** terminal of the input node is connected.

If the **Catch** terminal of the input node is not connected, the type of input node controls the default processing. For example, if the input node supports the **Transaction mode** property then this property is checked. If the message is not transactional, it is discarded. Transactional messages are rolled back to the input queue, and the message flow restarts. The second attempt might not work and ends with another backout.

## ExceptionList

- Contains information about exceptions that occur when a message is processed when an exception is caught on the local **Failure** path or **Catch** path
- Can be accessed, modified, and traced
- The ExceptionList tree is a part of the logical message tree
  - Consists of a set of zero or more exception descriptions
  - The innermost child is the original cause of an error



© Copyright IBM Corporation 2015

Figure 8-4. ExceptionList

WM666 / ZM6661.0

### Notes:

The ExceptionList is part of the logical message tree that the integration node fills whenever an exception condition occurs. You can use the information in the ExceptionList to handle these exceptions explicitly, either in the local **Failure** path of a node, or in a **Catch** path of an input node.

In most cases, the important information of the ExceptionList is in the last, innermost structure, which is the original error that occurred.



## Options for catching exceptions

- Connect the **Failure** terminal of the node to a sequence of nodes that processes the node's internal exception
- Connect the **Catch** terminal of the node to a sequence of nodes that processes exceptions that are generated beyond it
- Insert one or more TryCatch nodes at specific points in the message flow to catch and process exceptions
- Include a Throw node to generate an exception
- Ensure that all the messages received by an MQInput node are:
  - Either processed in a transaction, or are not processed in a transaction
  - Either persistent, or are not persistent

© Copyright IBM Corporation 2015

Figure 8-5. Options for catching exceptions

WM666 / ZM6661.0

### Notes:

When you design a message flow application, you should consider the options that are listed in this figure to identify and report on exceptions.

For example, you can connect a Failure terminal to a subflow that processes the packages the error message in the ExceptionList and the message and sends an email to an administrator.

## TryCatch node



- By default, a runtime exception returns control to the input node of a message flow
- Use the TryCatch node in the flow to control errors locally
  - TryCatch node has one input terminal and two output terminals (**Try** and **Catch**)
  - Message is passed to the **Try** branch and runs to completion unless an exception occurs
  - Acts as a decision point
  - Does not modify message
- If an exception occurs in the **Try** branch:
  - Integration node updates the ExceptionList
  - Existing ExceptionList information is written to a local error log
  - Root and LocalEnvironment are reset to the state at the start of the TryCatch node
  - Any database changes and MQPUTTs in the **Try** branch are not undone
  - Enter the **Catch** branch; if it is not connected, return control to the Input node

© Copyright IBM Corporation 2015

Figure 8-6. TryCatch node

WM666 / ZM6661.0

### Notes:

With the TryCatch node, you can specify a target path and, if an exception occurs, catch it and send it down another path in the flow. When you use the TryCatch node, the message is not rolled back to the original input node.

Information about the type of exception can be found in the ExceptionList, which is part of the logical message. You can refer to this information in subsequent processing nodes (for example, filter on exception type, update a message field with error information, or write the ExceptionList to a database).

Use the TryCatch node with caution. For example, if a database update operation is part of the transaction and you insert a TryCatch in the flow, the database update on the **Try** path is still done. You must do a manual rollback later or insert a Throw node in the **Catch** path to enforce a controlled rollback.

## Message flow error behavior with TryCatch

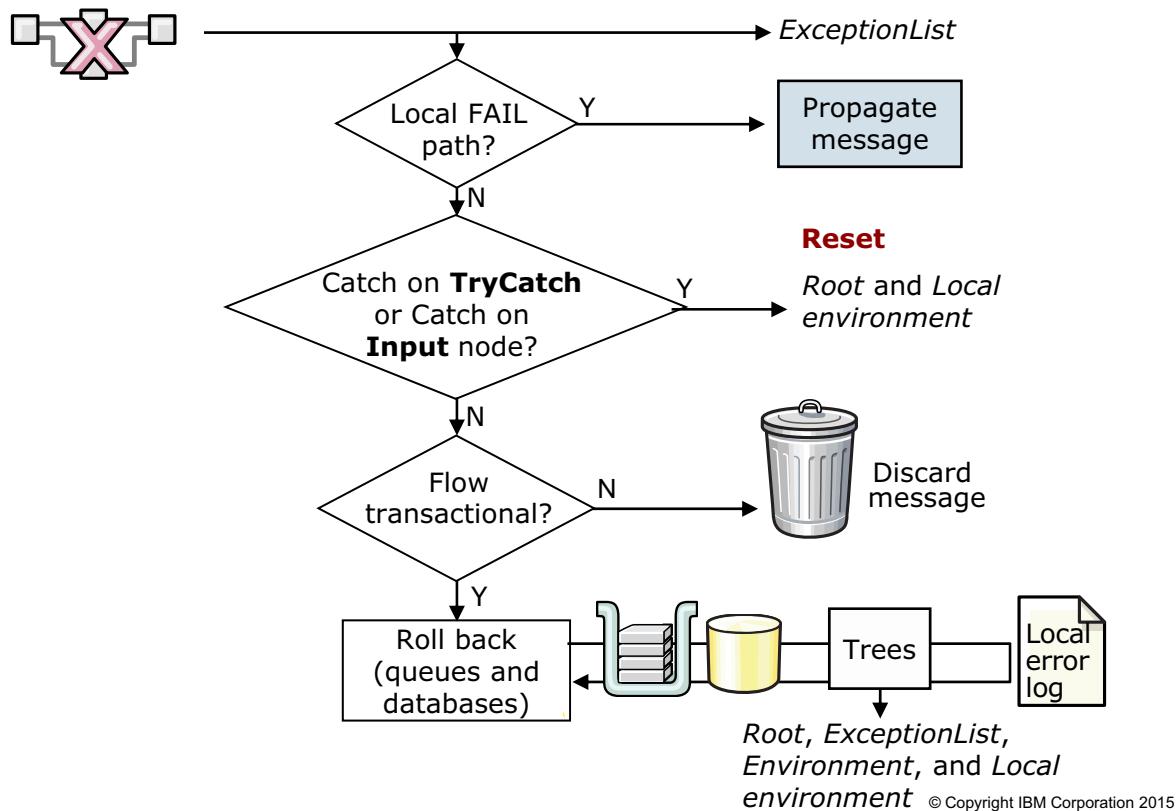


Figure 8-7. Message flow error behavior with TryCatch

WM666 / ZM6661.0

### Notes:

Earlier in this unit, you saw how IBM Integration Bus responds by default if a runtime exception occurs. Now that you understand TryCatch processing, review what happens if you include one or more TryCatch nodes in a message flow.

This figure shows the flow of events if an error is generated in a message flow node.

If the **Failure** terminal of the node where the error occurs is connected, the message is propagated to that terminal. Diagnostic information is available in the **ExceptionList**. Otherwise, if it is *not* connected, an exception is generated and is thrown back towards the input node.

If a TryCatch node is encountered on this route back to the input node, the flow proceeds to its **Catch** terminal. The Root and LocalEnvironment trees are reset to the values that they had **before** the TryCatch node. Here, too, you can refer to the diagnostic information in the **ExceptionList** with ESQL. If there is *no* TryCatch node, the exception reaches the input node.

If the **Catch** terminal of the input node is connected, the message is propagated there. **ExceptionList** entries are available. Root and LocalEnvironment trees are reset to their initial values. Otherwise, if it is *not* connected, the transaction status of the message is considered.

If the message is *not* transactional, the message is discarded. This action is not explicitly documented; the system log contains information only about the original error. The processing from this point depends on the type of the input node.

## Controlled throwing and logging of exceptions



- Throw node is typically connected at the end of a **Catch** path
- ESQL THROW function
  - Halts processing in a Compute node
  - Explicitly handles database exceptions
  - Database return codes can be captured with ESQL functions if the advanced property **Throw exception on database error** is not selected

```
THROW USER EXCEPTION
CATALOG 'BIPv610' MESSAGE 2950 VALUES (
  'SQL State:', SQLSTATE,
  SQLCODE, SQLNATIVEERROR, SQLERRORTEXT);
```

- JavaCompute node class `MbService`
  - Provides Event log and syslog methods `logError()`, `logWarning()`, and `logInformation()`
  - Message text is held in Java resource bundles

© Copyright IBM Corporation 2015

Figure 8-8. Controlled throwing and logging of exceptions

WM666 / ZM6661.0

### Notes:

The Throw node has one input terminal and no output terminal. Wire a Throw node as the last node in a **Catch** path to throw a user-defined exception.

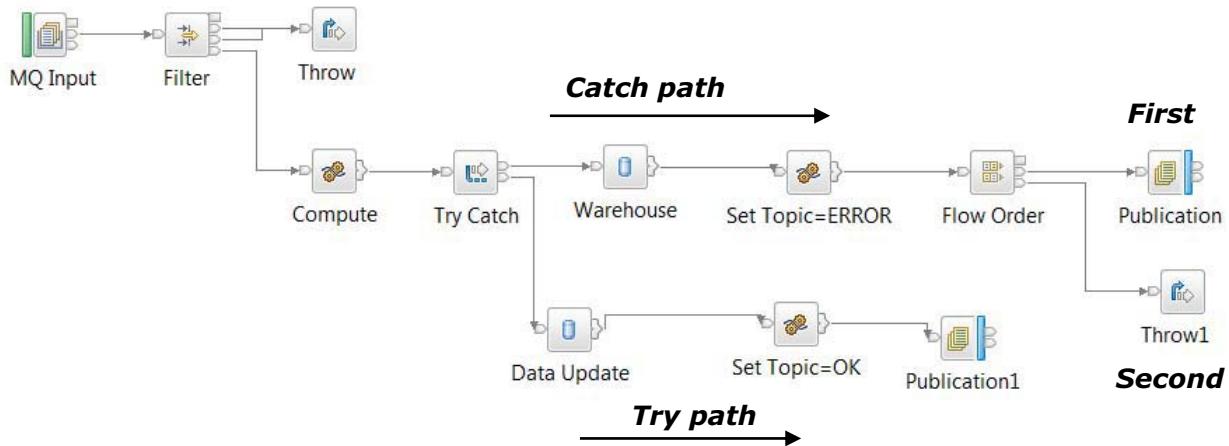
You can configure the Throw node to output specific diagnostic information to the system log or the `ExceptionList`.

The same behavior is obtained by using the ESQL `THROW` function in a Compute, Filter, or Database node. Database state indicators can be retrieved by using ESQL functions, but only if Compute and Database nodes are explicitly configured NOT to throw exceptions on database errors.

For diagnostic errors, you can use the IBM Integration Bus supplied message catalog, or build your own catalog.

In a JavaCompute node, you do not need to catch exceptions that are thrown. The integration node handles exceptions automatically. If you catch an exception in the code, throw it again, allowing the integration node to construct an exception list and propagate the message to the **Failure** terminal, if it is connected. For application-specific errors, use the `MbService` class for writing to the event log or system log.

## TryCatch node example



© Copyright IBM Corporation 2015

Figure 8-9. TryCatch node example

WM666 / ZM6661.0

### Notes:

This figure shows an example of a message flow with a TryCatch node with a wired **Try** path and a wired **Catch** path.

All changes to a logical message that occur in the **Try** branch are reset when an exception is caught, but in this example, the TryCatch node does back out the message. Because no back out occurs, any database updates or MQPut calls that are made in the **Try** branch (before the exception occurred) are *not* rolled back.

To force a rollback, you must explicitly throw an exception on the **Catch** branch. In this case, the order of processing is important. A FlowOrder node forces a prescribed order of processing of the messages that this node propagates.

## Message flow error tips

- Message flows work like logic diagrams; be sure to consider **all** relevant outcomes
- Messages are discarded in case of error during nontransactional flows if:
  - **Failure** terminal of the node where error occurs is not connected
  - **Catch** terminal of input node is not wired or another error exists
- Be sure that you understand node behavior; default behavior is not always as expected

© Copyright IBM Corporation 2015

Figure 8-10. Message flow error tips

WM666 / ZM6661.0

### Notes:

If you are going to rely on the default behavior of a message flow, that is, leave terminals unwired, make sure that you understand the default behavior.

Be sure that you understand the behavior of the input node when an error occurs. For example, if the input node is an MQInput node, the message can enter a loop if no dead-letter queue is configured for the queue manager.

## Problem determination tools

Tool	Description
Log files	<ul style="list-style-type: none"> <li>• Primary source of information</li> <li>• Automatically records all errors</li> <li>• No increase in processor usage</li> </ul>
IBM Integration Toolkit Flow exerciser	<ul style="list-style-type: none"> <li>• Development tool for viewing message path and structure and content of the logical message tree at any point in a message flow</li> </ul>
User trace	<ul style="list-style-type: none"> <li>• Find where the message was routed and why</li> <li>• Most comprehensive tool when used with the trace node</li> </ul>
Trace node	<ul style="list-style-type: none"> <li>• Shows any part of a message at any point in the flow</li> <li>• Best suited for large messages</li> </ul>
IBM Integration Toolkit Test Client and Message flow debugger	<ul style="list-style-type: none"> <li>• Set breakpoints and step through the flow</li> <li>• Examine and change message, ESQL, and Java variables</li> <li>• Requires local IBM MQ queue manager</li> </ul>
RfhUtil (SupportPac IH03)	Put and get test messages
IBM Knowledge Center	Help resources, error message descriptions, and solutions

© Copyright IBM Corporation 2015

Figure 8-11. Problem determination tools

WM666 / ZM6661.0

### Notes:

The table in the figure summarizes the testing and problem determination tools available in IBM Integration Bus. It might be necessary to use a combination of tools and methods to determine the reason for a message flow failure.

Each of these tools is described in detail in this unit.

IBM MQ SupportPacs and IBM Developer Works have more test utilities. For more information, see the following websites:

<http://www.ibm.com/software/integration/support/supportpacs>

<http://www.ibm.com/developerworks/>

## Local error log (syslog)

- Multiple messages per runtime error; must review all of them
  - Status of integration node
  - Runtime errors
  - Some database errors are logged
- Always check after system start, deployment, and during the testing of message flows
- Location differs by operating system
  - On Windows, Event Viewer **Application** log
  - On UNIX, location is based on an entry in /etc/syslog.conf
  - On z/OS, address space job log and z/OS SYSLOG

Level	Date and Time	Source
Information	6/1/2015 8:47:12 AM	IBM Integration v10000
Information	6/1/2015 8:39:17 AM	IBM Integration v10000
Error	6/1/2015 8:39:17 AM	IBM Integration v10000
Error	6/1/2015 8:39:17 AM	IBM Integration v10000
Information	6/1/2015 8:39:17 AM	IBM Integration v10000
Information	6/1/2015 8:39:17 AM	IBM Integration v10000
Information	6/1/2015 8:39:17 AM	IBM Integration v10000

Event 4041, IBM Integration v10000

**General** | **Details**

(TESTNODE\_iibadmin.default) Integration server "default" received an administration request that encountered an exception.

While attempting to process an administration request, an exception was encountered. No updates have been made to the configuration of the integration server.

Review related error messages to determine why the administration request failed.

Log Name: Application  
Source: IBM Integration v10000  
Event ID: 4041  
Level: Error  
User: N/A  
OpCode:  
More Information: [Event Log Online Help](#)

© Copyright IBM Corporation 2015

Figure 8-12. Local error log (syslog)

WM666 / ZM6661.0

### Notes:

The local error log is the primary source of problem determination information for IBM Integration Bus components and message flows. It contains all recorded errors and warnings. Additionally, it is possible that errors can occur in other applications, such as DB2 and IBM MQ.

In Windows, IBM Integration Bus automatically writes to the Application Log in the Windows Event Viewer. To display the Application Log:

1. Select **Administrative Tools > Event Viewer**.
2. Expand **Windows Logs**.
3. Click **Application**.

On Windows, the Event Viewer might contain more than one message for an error with each message that provides more detail; always check each message that is identified as an error.

In Linux and UNIX, the system log (syslog) must be explicitly configured. The system log contains only the first line of the message. You must look up the remainder of the message in the IBM Knowledge Center.

Check the local error log regularly; errors can potentially fill the error log if you do not act on them.



## IBM Integration Bus activity logs

- Give immediate, basic information about what is happening in the message flows, and how they are interacting with external resources
- Enabled by default
- For longer term use, option to write activity logs to a file

© Copyright IBM Corporation 2015

Figure 8-13. IBM Integration Bus activity logs

WM666 / ZM6661.0

### Notes:

Another source of useful information about potential problems with message flows are the *activity logs*. These logs provide an overview of recent activities in your message flows and associated external resources. They contain immediate, basic information about what is happening in the message flows, and how they are interacting with external resources. For example, they can help you understand why your message flow is not processing any messages from a remote resource.

Logs can be generated for individual message flows or resource types. You can save an activity log to a file in comma-separated value format or copy and paste a selection of the data.

Consider reviewing activity logs as one of your first points of investigation when something unexpected happens in a message flow. Since you do not need to turn on activity logs (it is enabled by default), it can provide an early indication of changed behavior, such as broken connections to external resources.

## Writing Activity logs to files

- Define an ActivityLog configurable service to write Activity logs to a file
- Option1: Use the IBM Integration web user interface
  - Right-click **Operational Policy > Configurable Services** and then click **Create**
  - Enter a configurable service name and then select **Activity Log Template**
  - Populate template and then click **Save**
- Option 2: Use the `mqsicreateconfigurableservice` command

Properties	
filter	
fileName	
minSeverityLevel	INFO
formatEntries	false
executionGroupFilter	
numberOfLogs	4
enabled	true
maxFileSizeMb	25
maxAgeMins	0

© Copyright IBM Corporation 2015

Figure 8-14. Writing Activity logs to files

WM666 / ZM6661.0

### Notes:

You can use the ActivityLog configurable service to configure the following Activity log file characteristics:

- Whether file logging is enabled
- The maximum number of files that are used for each log
- The maximum size of the log files (before the log rotates to the next file)
- The maximum age of the log files (before the log rotates to the next file)
- Filters that determine the content of the logs
- The severity levels of messages logged
- The log path and file name
- Whether messages are formatted with inserts included in the message text

The configurable service name must not duplicate the name of an existing configurable service or IBM predefined configurable service template.

## Using a command to create an ActivityLog configurable service

- Use the `mqsicreateconfigurableservice` command with the following parameters:
  - Integration node name
  - Specify `-c ActivityLog` to identify the configurable service type
  - Specify `-o` to identify the name of the configurable service
  - Specify name (`-n`) and value (`-v`) pairs to override the default properties

**Example:** Create an ActivityLog configurable service that is named MyActivityLog on an integration node that is named IBNODE and that writes the logs to a file that is named `ActivityLogs` in the `C:\IBNodeLogs\` directory

```
mqsicreateconfigurableservice IBNODE -c ActivityLog
-o MyActivityLog -n filepath -v "C:\IBNodeLogs\ActivityLogs"
```

© Copyright IBM Corporation 2015

Figure 8-15. Using a command to create an ActivityLog configurable service

WM666 / ZM6661.0

### Notes:

You can use the `mqsicreateconfigurableservice` command or the IBM Integration web user interface to create an Activity Log configurable service.

The `filepath` variable in the `mqsicreateconfigurableservice` command is a string value that specifies the fully qualified path and the file name for Activity log files. This value is not set by default. If it is not set, file logging is turned off. To enable file logging, set the value, which must be unique for each ActivityLog configurable service instance.

The most recent activities are logged to a file that is identified with the `filepath` attribute. The names of the other files in the circular log are constructed from file name and standard suffixes. For example, a value of `filePath\myLog` results in log files that are named `myLog` (the current log file), `myLog.1`, and `myLog.2`, where files with higher prefixes contain older log entries. These files are written to the directory specified by the `filepath` variable.

## Other logs

- Eclipse log shows failures with IBM Integration Toolkit and plug-ins
  1. Switch to the **Plug-in Development** perspective
  2. From the main menu, click **Window > Show view > Other**
  3. Click **General > Error Log**
  4. Errors are displayed in the **Problems** view; double-click items for more detail
- TDS log shows errors in using tagged or delimited stream physical format messages
  - TDS format rules are checked at deployment time
  - Error message BIP1836 is displayed in the IBM Integration Toolkit
  - Details of the error are written to the `TDS.log` file in the `install_dir/log` directory
- If appropriate for the message flow, also check:
  - IBM MQ logs
  - Database management system log

© Copyright IBM Corporation 2015

Figure 8-16. Other logs

WM666 / ZM6661.0

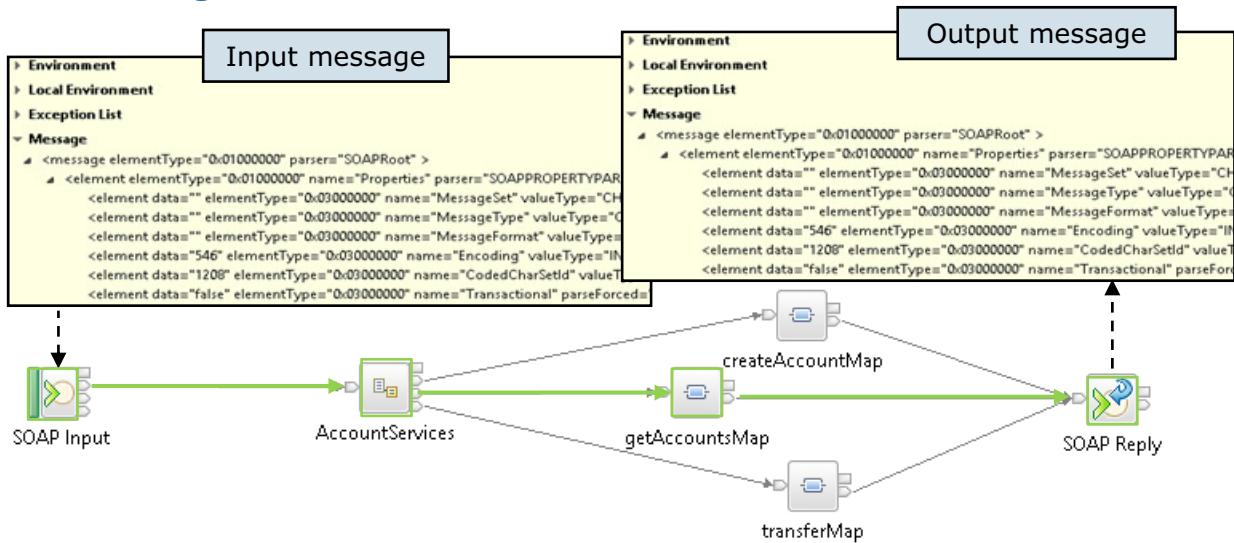
### Notes:

Other IBM Integration Bus logs might contain information that can help you troubleshoot a problem. While these logs might not be the first places that you check when errors occur, they can be useful in particular instances. These logs include the Eclipse log, TDS log, and external application logs.

If you engage IBM support, you might be asked to review these logs, or provide them to the support team.



## IBM Integration Toolkit Flow exerciser



1. Create and send a message to the input node of the message flow
2. Highlight the message path on the message flow and any subflows that are associated with the message flow
3. Display the content of the logical message tree for a message that passed through a connection in the message flow
4. Save the content of the logical message tree as a recorded message that you can send to the message flow later

© Copyright IBM Corporation 2015

Figure 8-17. IBM Integration Toolkit Flow exerciser

WM666 / ZM6661.0

### Notes:

To check that a message flow or integration service is processing messages as expected, you can send messages to the flow by using the Flow Exerciser or an external client, such as RFHUtil. You can then use the Flow Exerciser to show the path that each message took, and view the structure and content of the logical message tree at any point in a message flow.

## User trace (1 of 2)

- User trace records all flow activities
  - Standard tool for message flow analysis
  - Helps in understanding activities in the flow and finding parser or modeling problems
  - Shows all exceptions in a stack with complete texts and inserts
- Trace levels
  - **Normal** tracks events that affect objects that you create and delete, such as nodes
  - **Debug** tracks the beginning and end of a process and monitors objects that the process affects
  - **None** turns off the user trace



User trace has a negative impact on performance.

© Copyright IBM Corporation 2015

Figure 8-18. User trace (1 of 2)

WM666 / ZM6661.0

### Notes:

Exceptions are often longer than a single line, and they normally come in groups of two or three. User trace shows the full text, with inserts, of all the exceptions in the stack. Some parsers emit user trace messages while parsing. Some parsing or modeling problems are hard to diagnose without this information.

You can usually use user trace for debugging your applications. You can trace integration nodes, integration servers, and deployed message flows.



### Warning

When you turn on user trace, you are causing extra processing for every activity in the component you are tracing. The components generate large quantities of data. You should expect to see some impact in performance while the trace is active. However, you can limit this additional processing by being selective about what you trace, and by restricting the time during which trace is active.

## User trace (2 of 2)

- Enable with `mqsicchangetrace` command

Example: `mqsicchangetrace IBNODE -u -e IS1 -l normal`

Where: `IBNODE` is the integration node name

`-u` specifies user trace

`-e` specifies the integration server

`-l` specifies the level of trace

- Use `mqssireadlog` to read the log file from the file system (`-f`) and generate an XML file (`-o`)

Example: `mqssireadlog IBNODE -u -e IS1 -o IS1.xml -f`

- Use `mqsisformatlog` to convert an XML file into a plain text file

Example: `mqsisformatlog -i IS1.xml -o IS1.txt`

© Copyright IBM Corporation 2015

Figure 8-19. User trace (2 of 2)

WM666 / ZM6661.0

### Notes:

You can activate user trace with the `mqsicchangetrace` command.

Trace data is collected in binary form. After the trace data is collected, you must use the `mqssireadlog` command to process the data and generate an XML file. To get the XML data into plain text, you must run another command, `mqsisformatlog`.

To determine the current trace status, use the `mqssireporttrace` command.

## Trace node

- Provides snapshot of (part of) a logical message
- Destination options:
  - File
  - User Trace log
  - Local system error log
- Content:
  - User-defined text
  - Message content
  - Date/time information
- Can be switched “on” or “off”



You can significantly improve the performance of a flow that includes Trace nodes by switching Trace nodes “off”

© Copyright IBM Corporation 2015

Figure 8-20. Trace node

WM666 / ZM6661.0

### Notes:

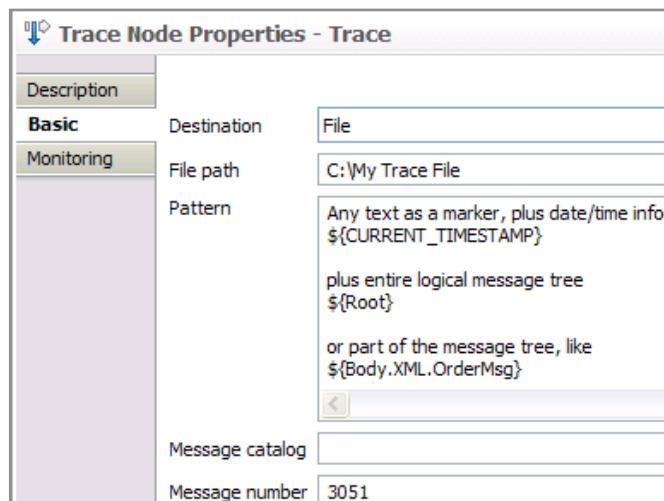
If the flow is processing large messages, it is better to inspect only small parts of the message that is running through the flow by using the Trace node, for example.

The Trace node can be inserted at any point within a message flow to capture data. The information can be sent to the user trace, the local error log, or to a file on the integration node server.

When you configure the Trace node, you select a destination for the trace file. If you select the option to write to a file, the trace file is in a readable text format. This format makes it easier and faster to see output.

## Configuring the Trace node

- Destination options:
  - File
  - User Trace log
  - Local system error log
- Syntax for a trace pattern:  
`my_text ${ESQL-expression}`
- Enabled by default
- Use the `mqsichangetrace` command to control Trace nodes for a specific flow or for all flows on an integration server



```
mqsichangetrace IntNode -e IntServer -n [on|off]
[-f flow -k application]
```

- Use the `mqsireporttrace` command or the IBM Integration web user interface to check the trace settings for integration servers

© Copyright IBM Corporation 2015

Figure 8-21. Configuring the Trace node

WM666 / ZM6661.0

### Notes:

The **Pattern** property on the Trace node is an ESQL pattern that specifies the information that is written to the location specified in the **Destination** and **File Path** properties. You can write plain text, which is copied into the trace record exactly as you entered it. You can identify parts of the message that the trace writes, by specifying the full field identifiers, which are enclosed within the characters  `${ }`. If you want to record the entire message, specify  `${Root}`. You can also use the ESQL functions to provide more information. For example, you can use the ESQL function `CURRENT_DATE` to record the date, time, or date and time at which the trace record was written.

Trace nodes are turned on and off using the `mqsichangetrace` command. You can control the trace nodes at the message flow level or at the integration server level. If the Trace node setting for an integration server is “off”, all Trace nodes in all its flows are turned off.

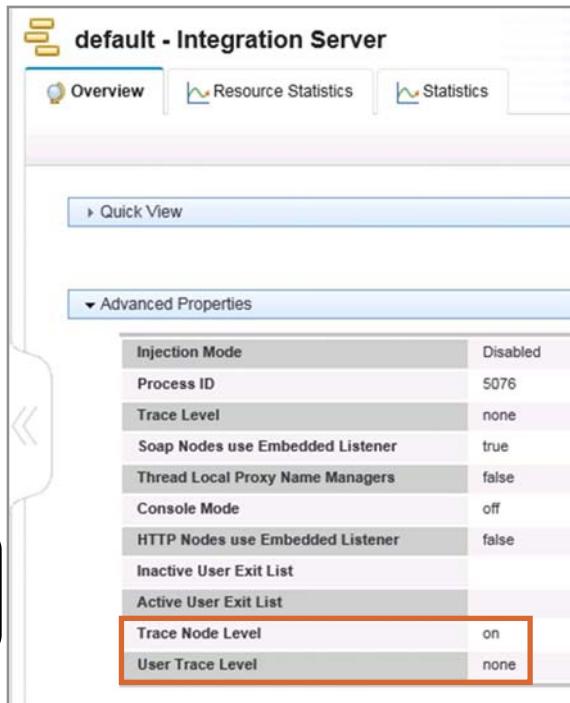
 WebSphere Education 

## Reporting current trace options for an integration node

- IBM Integration web user interface integration server status shows **Trace Node Level** and **User Trace Level** status
- **mqssireporttrace** reports the current trace current options on an active integration node
  - By default, lists all **active** user and service trace settings for the integration node
  - For status of User Trace, specify **-u**
  - For status of Trace node, specify **-n**

Examples:

```
mqssireporttrace IBNODE
mqssireporttrace IBNODE -u
mqssireporttrace IBNODE -n -e default
```



Setting	Value
Injection Mode	Disabled
Process ID	5076
Trace Level	none
Soap Nodes use Embedded Listener	true
Thread Local Proxy Name Managers	false
Console Mode	off
HTTP Nodes use Embedded Listener	false
Inactive User Exit List	
Active User Exit List	
Trace Node Level	on
User Trace Level	none

© Copyright IBM Corporation 2015

Figure 8-22. Reporting current trace options for an integration node

WM666 / ZM6661.0

### Notes:

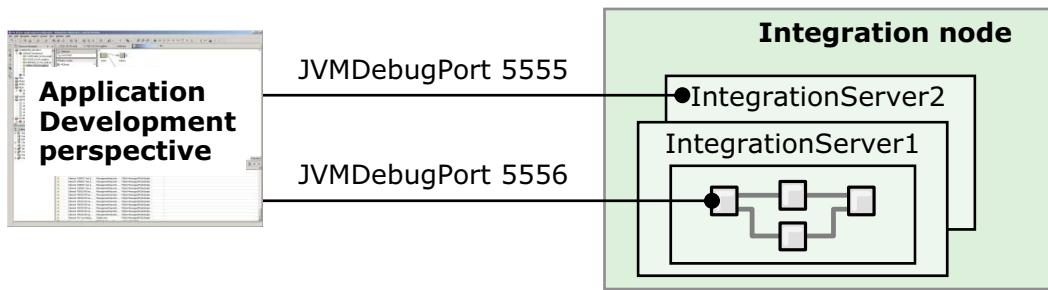
The **mqssireporttrace** command reports the trace options that are currently set for an active integration node.

The integration node name is a mandatory parameter, but if no other parameters are specified, the **mqssireporttrace** command lists all **active** user and service trace settings for that integration node.

Alternatively, you can use the **-u** or **-n** options for more specific reports on the current trace settings.



## Message flow debugger



- Eclipse-based integrated debugging tool attaches to integration servers
  - Can attach to multiple integration servers at the same time, allowing for debugging of multiple flows in multiple integration servers from the same Integration Toolkit
  - Only one Integration Toolkit can debug one integration server at a time
- Debugs flows, subflows, ESQL, Java code, and maps
- A message flow must:
  - Be deployed and running
  - Be open in the Message Flow editor
  - Have at least one breakpoint that is defined in the flow
- Must use the Integration Toolkit Test Client or a third-party tool to send a message to the flow

© Copyright IBM Corporation 2015

Figure 8-23. Message flow debugger

WM666 / ZM6661.0

### Notes:

The message flow debugger provides a convenient, graphical interface for flow testing.

The message flow debugger is based on the IBM Common Debug Architecture (CDA) that other IBM products use. The CDA is a reliable and robust Java Debug Channel transport technology that is based on the standard Java Debug Wire Protocol (JDWP).

You can debug a wide variety of error conditions in flows, including:

- Nodes that are wired incorrectly (for example, outputs that are connected to the wrong inputs)
- Incorrect conditional branching in transition conditions
- Unintended loops in the flow

The message flow debugger is a good choice for normal debugging scenarios. Its main advantage is that special skills or authorizations are not needed on the operating system of integration nodes; however, it cannot do everything. Make user trace the standard tool for diagnosis, which is combined with Trace nodes. The debugger does not always give an accurate representation of the message tree; a Trace node is always accurate, and it can optionally write its output to the user trace.



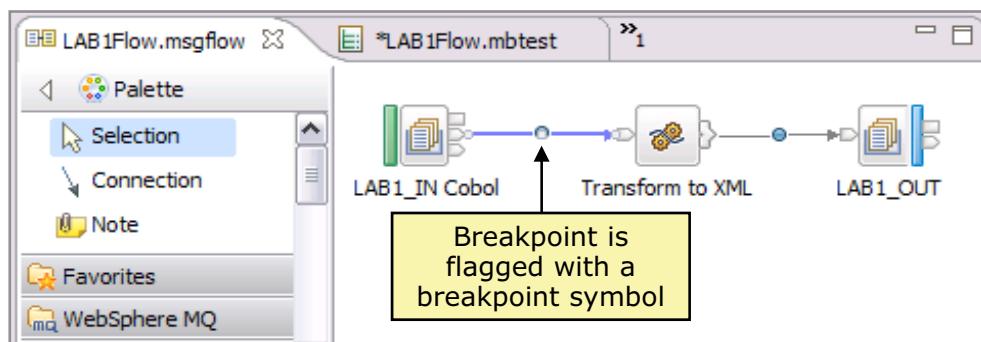
### **Warning**

Do not use the debugger on messages greater than 1 MB, as you might experience performance problems in the workspace because of excessive memory requirements.

When you debug message flows, use an integration node that is not being used in a production environment. Debugging might degrade the performance of all message flows in the same integration server and those flows in other integration servers that use the same integration node because of potential resource contention.

## Breakpoints

- Breakpoints control where the message flow pauses in the debugger
- Add breakpoints to a message flow:
  - Right-click connections to add a breakpoint to the selected connection
  - Right-click the node to add breakpoints to all connections that enter or leave the selected node
- Add breakpoints to code:
  - In Compute, Filter, and Database nodes with ESQL
  - In JavaCompute node with Java code



© Copyright IBM Corporation 2015

Figure 8-24. Breakpoints

WM666 / ZM6661.0

### Notes:

With the message flow debugger, you can set breakpoints in a message flow and then step through the flow. While you are stepping through the flow, you can examine and change the message variables and the variables in ESQL and Java code.

To set a breakpoint to a message flow connection:

1. Open the message flow in the Message Flow editor in the IBM Integration Toolkit
2. Right-click a connection or a node and click **Add Breakpoint**.

You can also set breakpoints in nodes that contain ESQL or Java code.

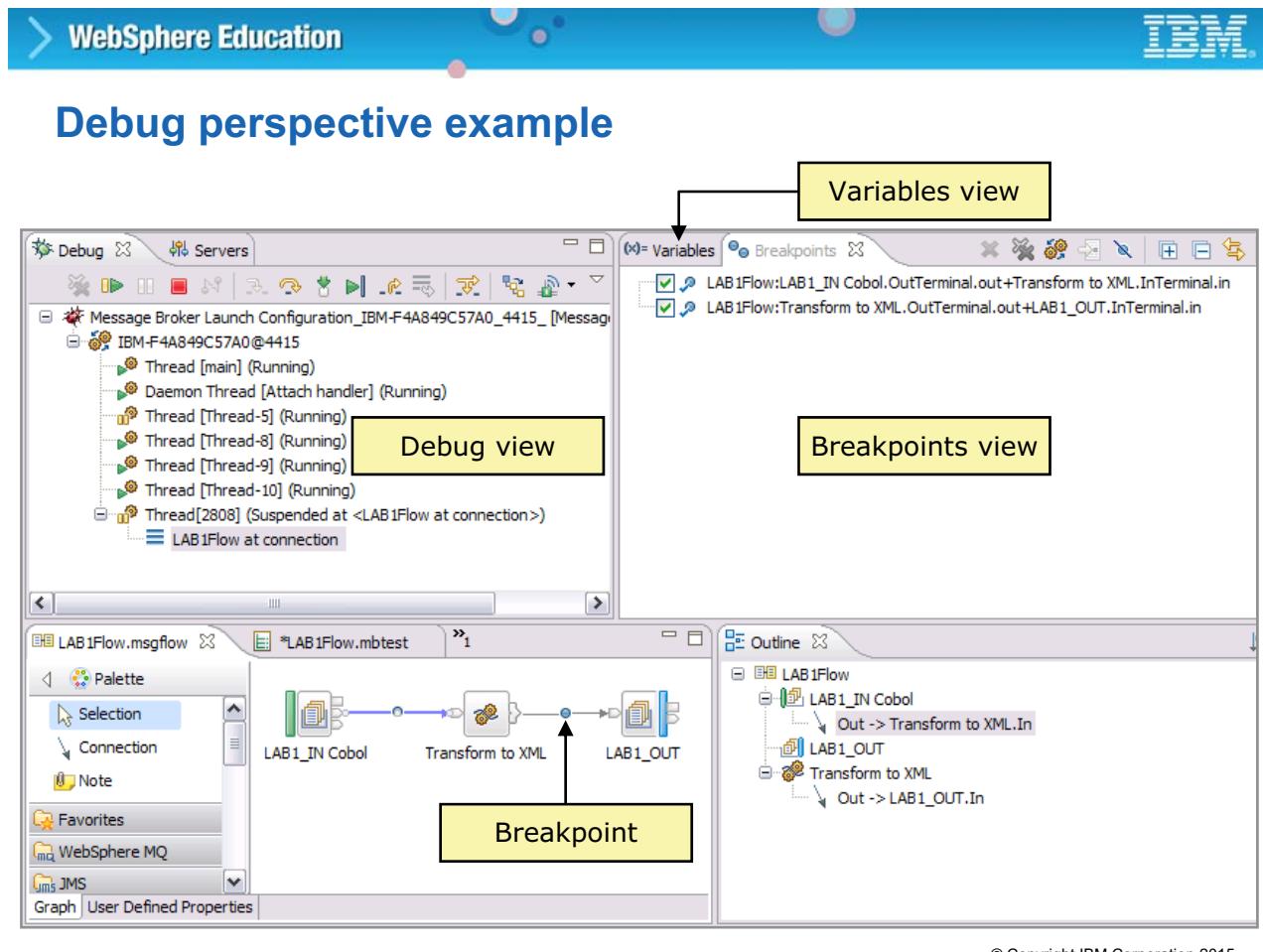


Figure 8-25. Debug perspective example

WM666 / ZM6661.0

## Notes:

The debugger operates in a Debug perspective in the IBM Integration Toolkit and uses Java Debug. The figure includes an example of the Debug perspective.

You use the Debug perspective to set more breakpoints in message flows, view the message flow events in a stack frame, and display information about the variables that are associated with the stack frame.

- In the **Debug** view, you manage the debugging or running of a program in the IBM Integration Toolkit. Each thread represents a node in the tree.
- The **Variables** view shows information about the current message and variables that are associated with the stack frame that travels through the flow.
- The **Breakpoints** view lists all breakpoints that are currently defined in the workspace.

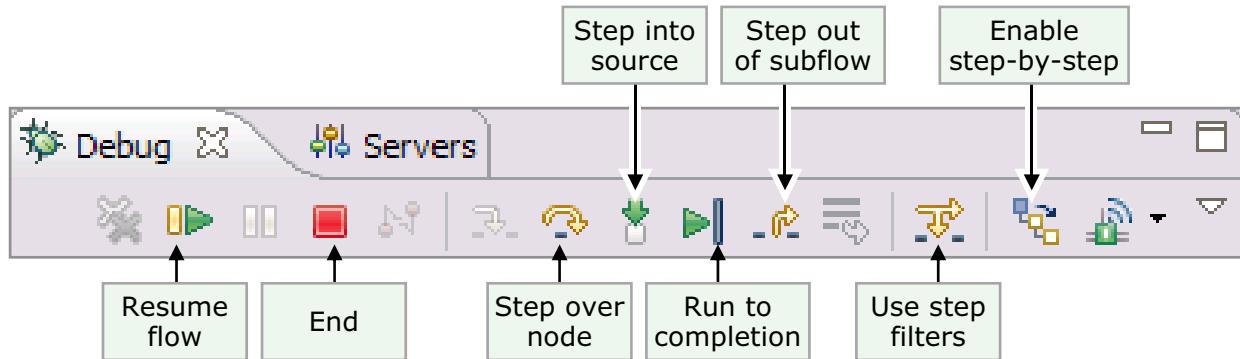
In Debug mode, the message flow runs until it encounters a breakpoint. When a breakpoint is encountered, control is returned to you. The current content of message trees can be examined and modified in the **Variables** view.

If the next node to be run supports source code, such as the Compute node (ESQL) and JavaCompute node, you can step into that node and debug the code at the source level.

The message flow view is updated to give you a graphic indicator of the breakpoints that are set and which breakpoint is current.



## Debug view options



© Copyright IBM Corporation 2015

Figure 8-26. Debug view options

WM666 / ZM6661.0

### Notes:

The Debug view contains shortcut icons for the Debug options. This figure shows the icons and their associated actions.

For a complete list of all the Debug perspective icons, see the IBM Knowledge Center to IBM Integration Bus.

## Message flow debugger configuration

1. Select the integration server where the message flow is running
  2. Set the Java debug port on each integration server that hosts a flow to be debugged
  3. Identify the applications that contain a source for the message flow
  4. Enable debug
- Can set the JVM debug port number with a line command:

```
mqsiclchangeproperties IntNodeName -e IntServerName
-o ComIbmJVMManager -n jvmDebugPort -v port
```

© Copyright IBM Corporation 2015

Figure 8-27. Message flow debugger configuration

WM666 / ZM6661.0

### Notes:

The message flow debugger attaches to the integration server that contains the message flow that you want to debug. Some one-time configuration is required to use the debugger:

- The flow must be deployed to an integration server.
- The Java debug port must be set on each integration server that hosts a flow to be debugged. For the debug port to be active, the integration server (or the integration node) must be restarted. If you set the debugger port in the IBM Integration Toolkit **Integrations Node** view, the integration server is restarted automatically.
- Debugger configuration must select the integration server where the flow is running.
- Debugger configuration must point at the projects that contain source for the message flow.

The debug port can be set by using the IBM Integration Toolkit or the `mqsiclchangeproperties` command.

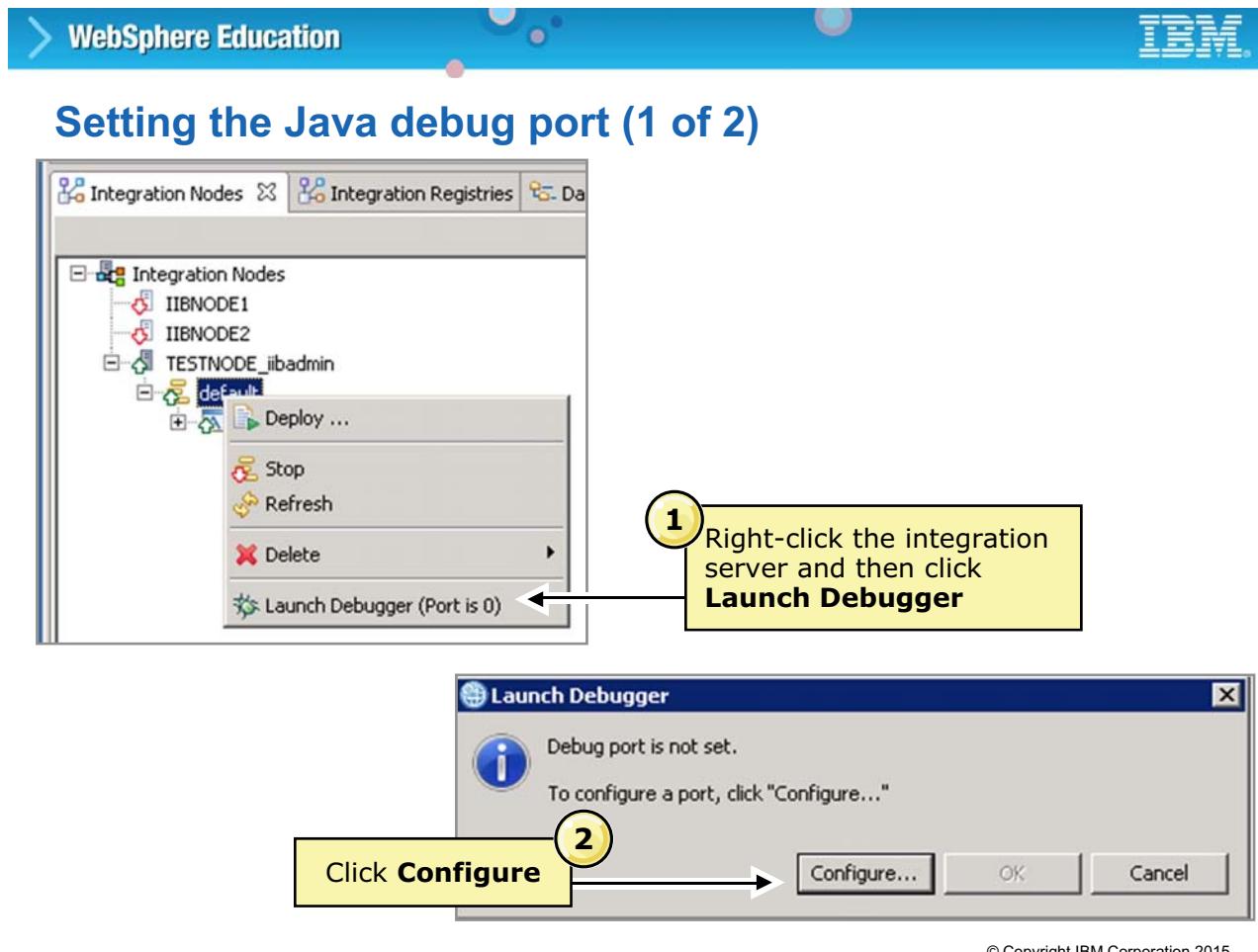


Figure 8-28. Setting the Java debug port (1 of 2)

WM666 / ZM6661.0

## Notes:

Before you can start the debugger, you must configure the debug port. The debug port can be configured in the IBM Integration Toolkit or by using the `mqsichangeproperties` command.

To configure the debug port in the Integration Toolkit:

1. Right-click the integration server in the **Integration Nodes** view and then click **Launch Debugger** from the menu. The Launch Debugger window indicates whether the debug port is set.
2. Click **Configure** on the **Launch Debugger** window.

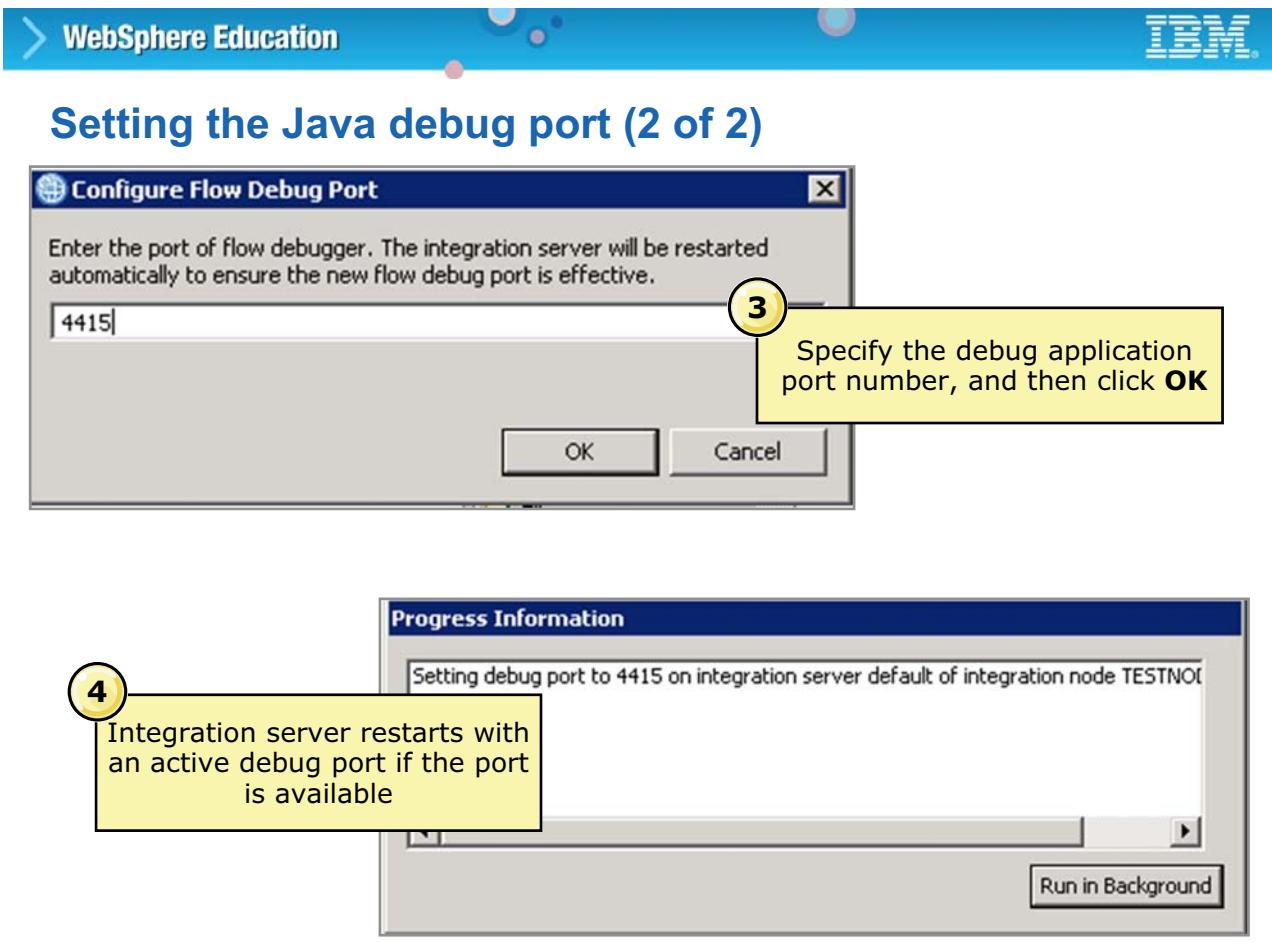


Figure 8-29. Setting the Java debug port (2 of 2)

WM666 / ZM6661.0

### Notes:

3. Specify the port number and click **OK**.
4. The integration server restarts automatically to set the debug port.

To check that the required port is enabled, right-click the integration server. The **Launch Debugger** option on the menu shows the debug port that is configured in Step 3.

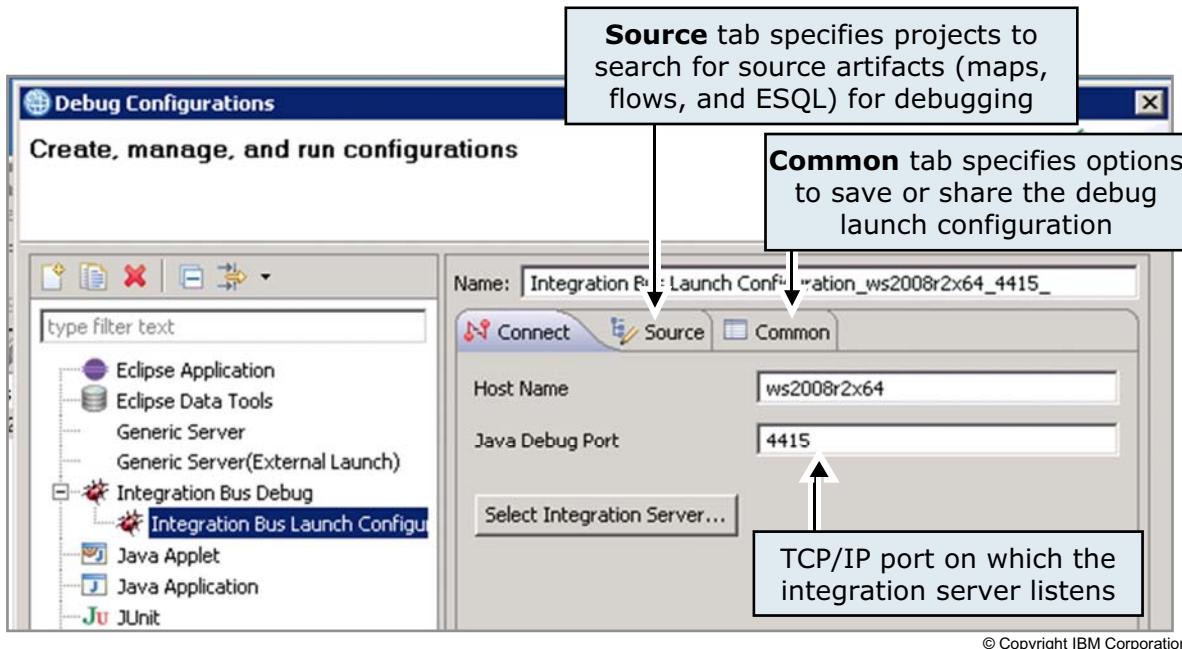
This port can also be specified by using the `mqsichangeproperties` line command. The change of properties becomes active after a restart of the integration server, or the entire integration node. When you use the IBM Integration Toolkit to set the debug port, the integration server is shut down and restarted automatically.



## Identifying projects that contain source artifacts

- Start the Debug configuration from the Debug perspective:

  - Click **Run > Debug configuration**
  - Expand **Integration Bus Debug > Integration Bus Launch Configuration**



© Copyright IBM Corporation 2015

Figure 8-30. Identifying projects that contain source artifacts

WM666 / ZM6661.0

### Notes:

The Debug control window is used to specify the required Java debug port, if it is not already set, and the source code of the message flow that is debugged.

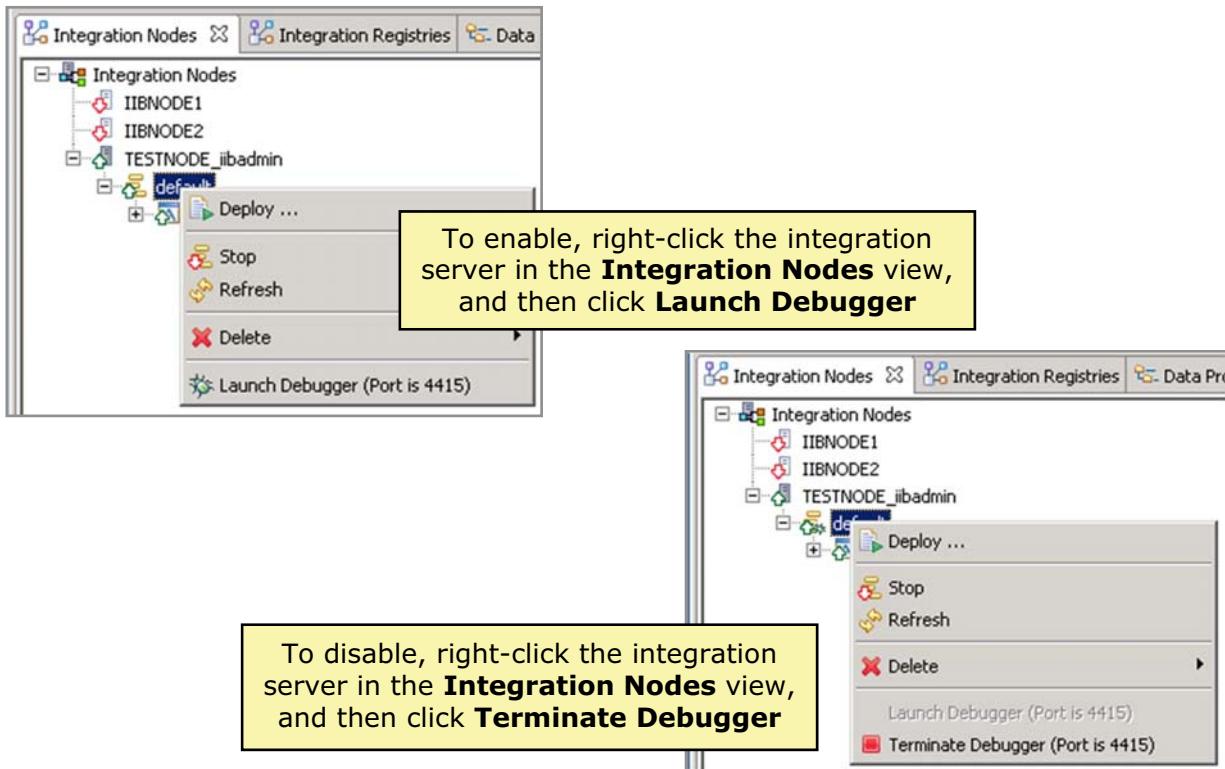
If it is a new configuration, specify the name of the Java debug configuration that you are going to create.

Next, specify the Java debug port. If you already know the port that you are using, manually enter the port by typing into the field. Alternatively, you can click **Select Integration Server**, which then queries the integration node for all available integration servers. From the returned list, select the required port.

Finally, on the **Source** tab, specify the Integration Toolkit location of the message flow that you are debugging.

The **Common** tab is optional, and is used to save local information to enable sharing of the debug configuration.

## Enabling and disabling the message flow debugger



© Copyright IBM Corporation 2015

Figure 8-31. Enabling and disabling the message flow debugger

WM666 / ZM6661.0

### Notes:

This figure shows the steps for enabling and disabling the Debugger in the IBM Integration Toolkit.

## Testing message flows with the Unit Test Client

- Must be enabled in **Integration Development** preferences
- Requires an IBM MQ queue manager that is local to the integration node
- Started from message flow editor on IBM MQ, SOAP, HTTP, JMS input, and SCA nodes
- Optionally creates missing input and output queues
- Sends a test message
- Monitors output transport protocols for message activity
- Displays an output message
- Saves tests in file for documentation and rerun
- Automatically builds and deploys BAR files to the selected integration server
- Saves a BAR file in the **TestClientBarFiles** project



The Unit Test Client fails if the **TestClientBarFiles** project is closed.  
Do not close the **TestClientBarFiles** project.

© Copyright IBM Corporation 2015

Figure 8-32. Testing message flows with the Unit Test Client

WM666 / ZM6661.0

### Notes:

The Test Client does all the steps necessary to test a flow. After some initial configuration, you can rerun the test with a single click.

The Test Client must be enabled in **Test Client** preferences.

1. Click **Window > Preference**.
2. Expand the item for **Integration Development** on the left and click **Unit Test Client**.
3. Click **Enable menus for Test Client** and then click **Apply**.

You can use the Test Client to send test messages to message flows that use any of the following input nodes:

- IBM MQ
- JMS
- HTTP
- SOAP
- SCA

You can test message flows in the context of an application or library, or you can test flows in isolation.

You can select the structure of an XML test message from the contents of message models and sets in your workspace and enter values for the elements. These values can be stored and reused in later test runs.

The Test Client monitors output nodes in the message flow so that you can see which nodes output messages are received on. When an error message is produced as the message passes along the flow, or when a message is received on an output node, a test event is recorded in the Test Client.



**Important**

The Test Client requires an IBM MQ queue manager on the same computer as the integration node.

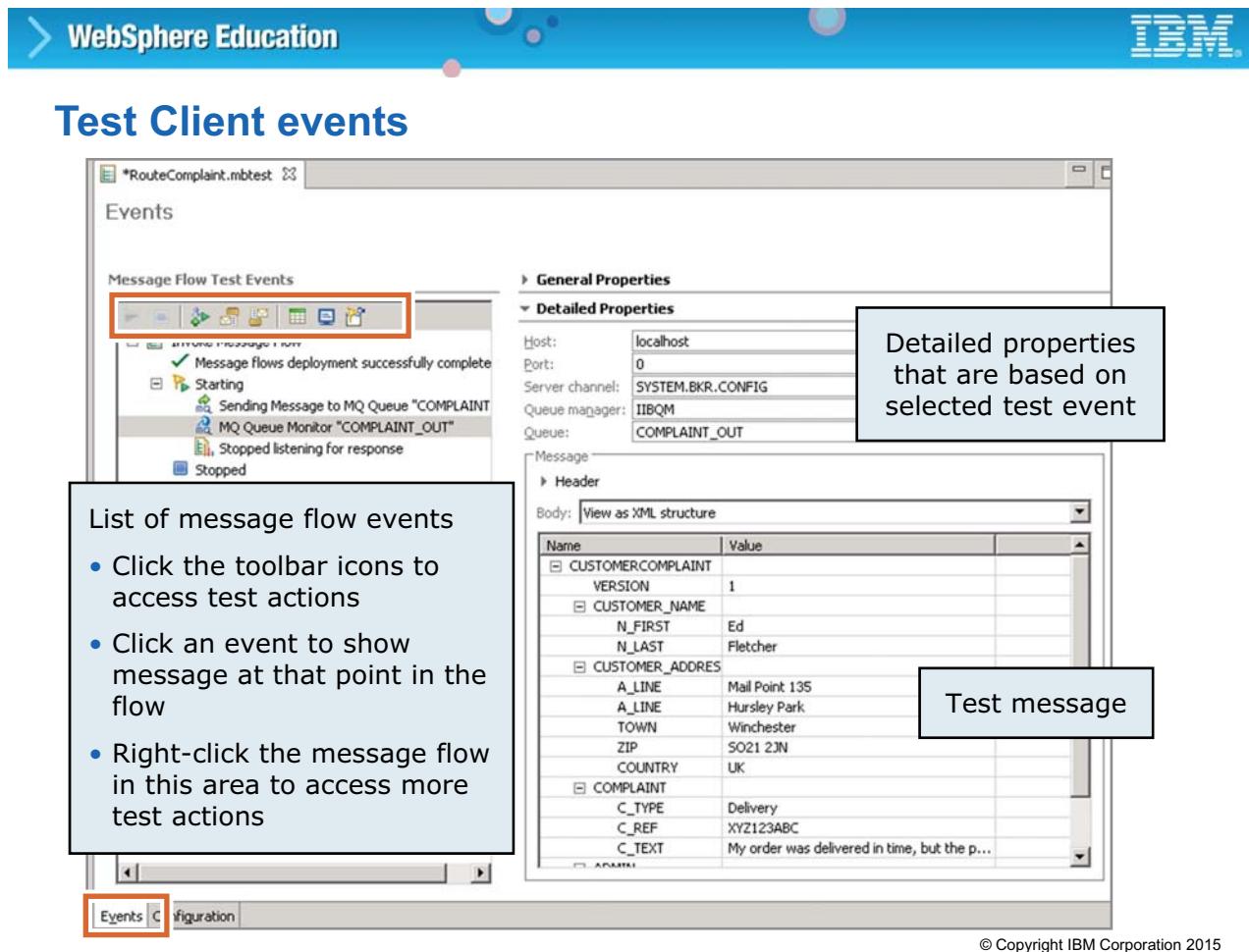


Figure 8-33. Test Client events

WM666 / ZM6661.0

## Notes:

To start the Test Client, right-click the input node of the flow you are testing and click **Test** from the menu. Clicking **Test** opens the Test Client dialog box on the **Events** tab, which is shown in the figure.

While running the Test Client, the list of message flow events is shown on the left side of the view. The right side of the view contains the event properties and information about the test message. You can also edit the properties and content of your test messages.

Several icons are provided on the upper right of the **Events** tab:

- **Invoke** starts a new “Invoke Message Flow” event where you can enter a request message and start the test.
- **Enqueue** puts the message to the specified queue manager, queue, port, and host name as defined in the **Detailed Properties** section on the right of the page.
- **Dequeue** gets the message from the specified queue manager, queue, port, and host name as defined in the **Detailed Properties** section on the right of the page.
- **Saved Message** displays the Data Pool editor in which you can select values that you used in a previous test session.

- **Stop** stops the current test.
- **Show Event Viewer** displays the Event Viewer if the operating system is Windows.
- **Show Console** opens the IBM Integration Bus runtime console view. This view shows more details of the test run.

More actions can be initiated on the **Events** tab, by right-clicking on the message flow in **Message Flow Test events** area:

- **Rerun** clones and reruns a previously started test message.
- **Duplicate** duplicates the current message. To duplicate a previously started test message, right-click the message flow and click **Duplicate**.
- **Invoke** restarts the current message.

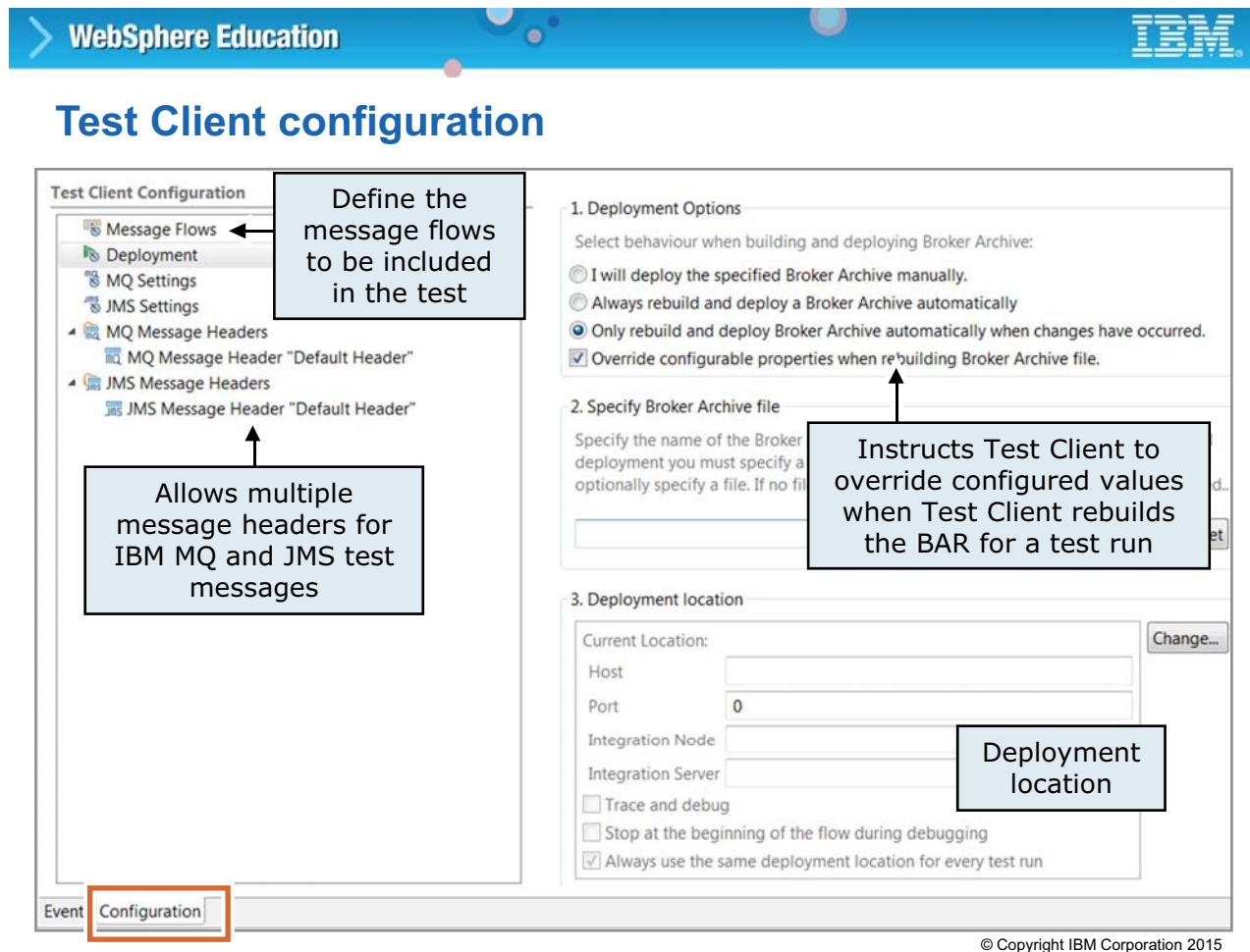


Figure 8-34. Test Client configuration

WM666 / ZM6661.0

## Notes:

Before running the flow with the Test Client, it might be necessary to specify other parameters. More parameters can be entered by switching to the Test Client **Configuration** tab, as shown in the figure.

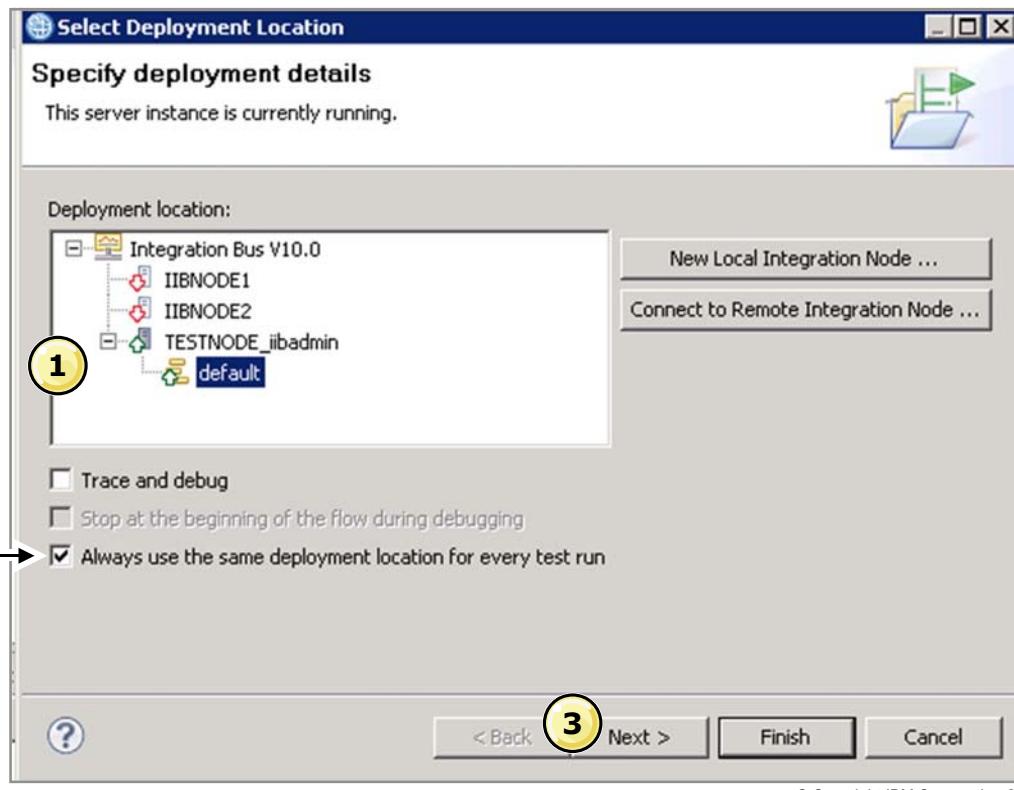
For example, you can specify how you want the Test Client to manage the deployment of the message flow that is being tested. You can choose to deploy the message flow manually. You can also choose to have the Test Client deploy it if there are changes, or have the Test Client deploy the BAR file every time.

You can also specify the values for IBM MQ and JMS message headers. If you want to test alternative header values, you can add more headers and select the header that was used in a previous test run.

You can manually open the BAR file editor to configure some configurable properties for the message flows being tested. The **Override configurable properties** when rebuilding BAR file option identifies whether user-configured values are preserved when the Test Client rebuilds the BAR file during the test.



## Specifying the Test Client deployment location (1 of 2)



© Copyright IBM Corporation 2015

Figure 8-35. Specifying the Test Client deployment location (1 of 2)

WM666 / ZM6661.0

### Notes:

The first time that you send a test message to an input node, you identify the integration server to deploy the message flow by using the **Deployment location** wizard.

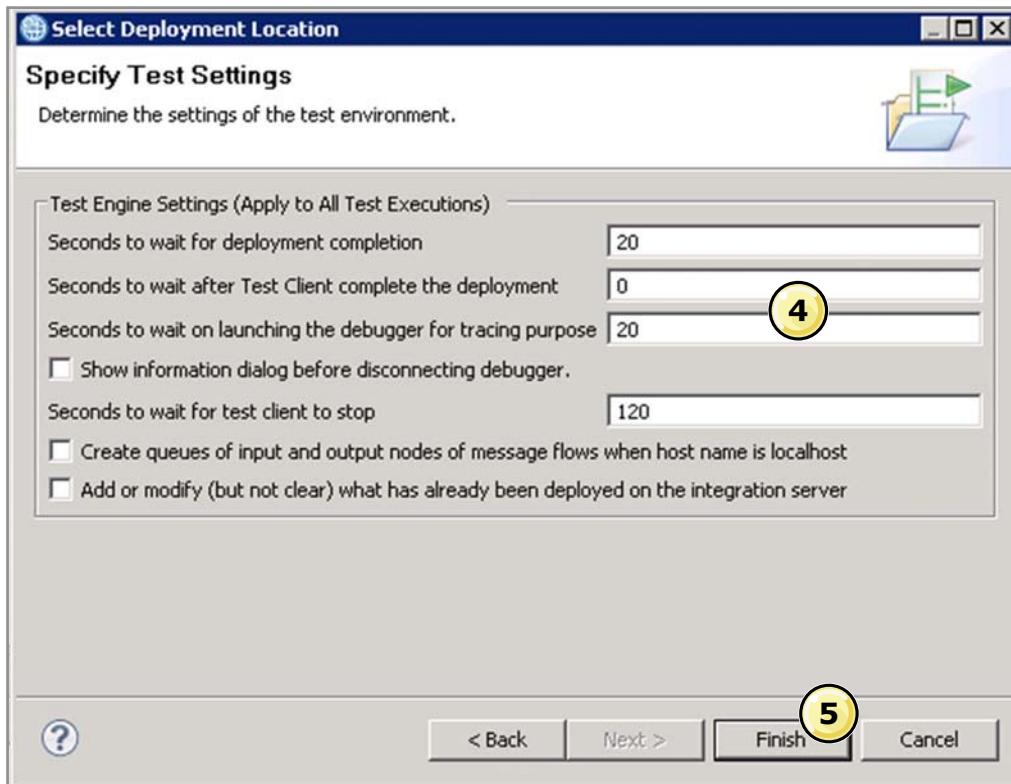
You can configure the deployment options to override the default behavior of the Test Client to deploy the message flow manually. Or you can have the Test Client deploy the message flow every time that you pass a test message to the message flow.

To specify the Test Client deployment location on the **Events** tab:

1. Specify a deployment location for a local integration node or click **New local integration node** to define a new local integration node and deployment location.
2. Optionally, select the **Always use the same deployment location** for every test run.
3. Click **Next**.



## Specifying the Test Client deployment location (2 of 2)



© Copyright IBM Corporation 2015

Figure 8-36. Specifying the Test Client deployment location (2 of 2)

WM666 / ZM6661.0

### Notes:

- Specify any options that you want to use for the Test Client, such as timeout values, or whether you want the Test Client to automatically create missing queues in IBM MQ.
  - Seconds to wait for deployment completion** is the amount of time in seconds to allow for deployment. The default value is 20.
  - Seconds to wait after Test Client complete the deployment** is the amount of time in seconds to wait after the Test Client completes deployment. The default value is 0.
  - Seconds to wait on launching the debugger for tracing purposes** is the amount of time in seconds to wait before starting the debugger. The default value is 20.
  - Show information dialog before disconnecting the debugger** displays a dialog box when you use the Test Client in run mode if the flow debugger is already connected.
  - Seconds to wait for test client to stop** is the amount of time in seconds to wait before ending the test. When the number of seconds elapses, monitoring of output nodes is stopped, and a Timeout event and a Stop event are displayed in the Test Client. The default value is 120.

- **Create queues of input and output nodes for message flows when host name is localhost** creates queues that are used in the message flow for the local host if they do not exist.
  - **Add or modify (but not clear) what has already been deployed on the integration server** adds or changes what was previously deployed to the current integration server.
5. Click **Finish**. The deployment information is updated.

WebSphere Education 

## Test and debug by using the Test Client (1 of 2)

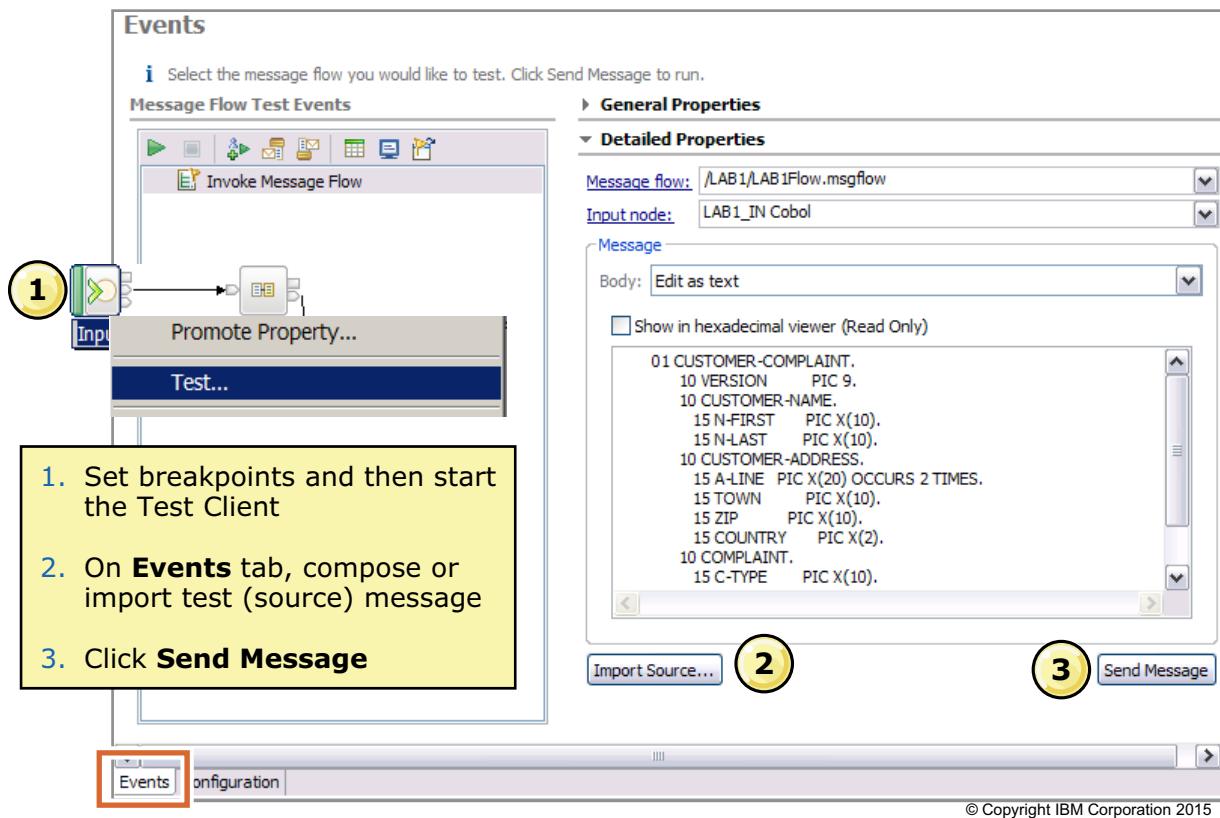


Figure 8-37. Test and debug by using the Test Client (1 of 2)

WM666 / ZM6661.0

### Notes:

You can also start the message flow debugger from the Test Client. The steps for using the Test Client are shown on this figure and the following figures.

1. After setting at least one breakpoint in the message flow, start the Test Client by right-clicking the input node in the Message flow editor and then clicking **Test** from the menu.
2. To select the source for a node, click **Import Source** and browse to the source file. The test data properties vary depending on the input node.

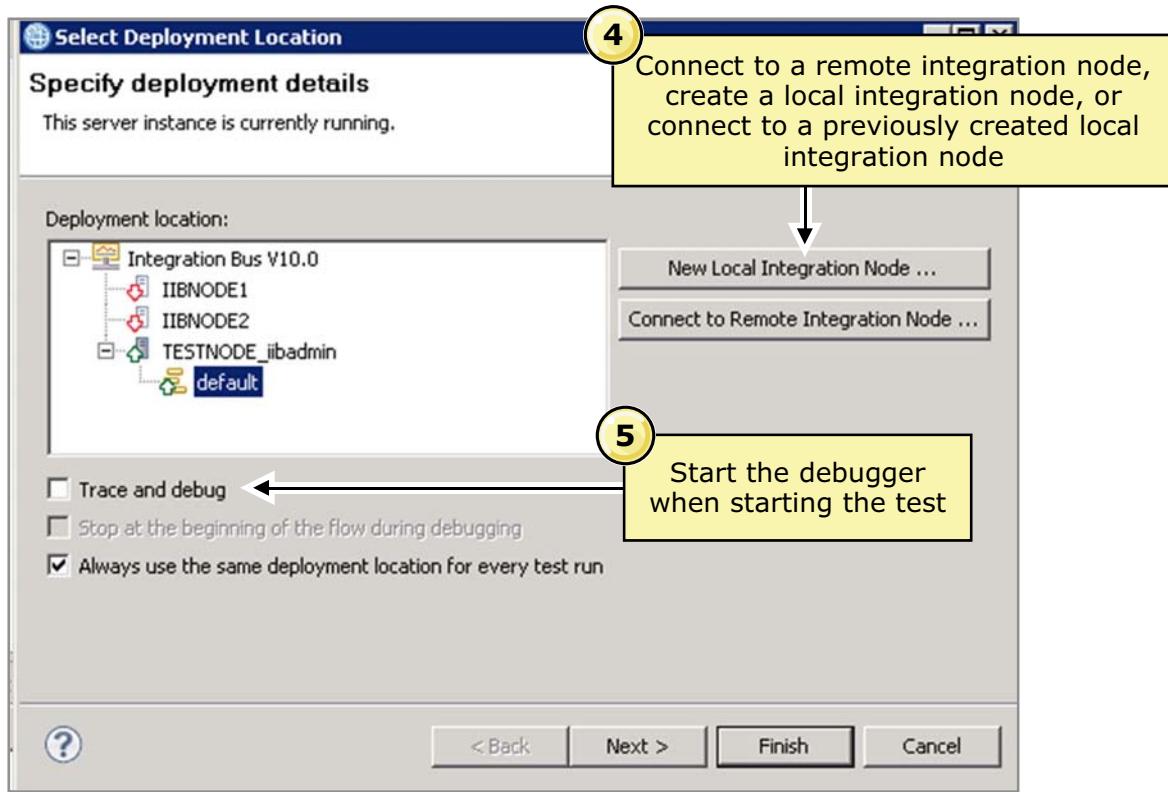
If the node is an IBM MQ or JMS input, and if the message type was not specified on the input node, select the message by choosing the **Add Message Part** action.

When you right-click the **Message Body**, you are presented with a number of options, depending on the type of input node.

You can select the structure of an XML test message from the contents of message models in your workspace and enter values for the elements. These values can be stored and reused in later test runs. If you have a prebuilt file that contains the test data input (XML or any other format), you can load the file by clicking the **Viewer** menu, and then clicking **Source**.

3. Click **Send Message**.

## Test and debug by using the Test Client (2 of 2)



© Copyright IBM Corporation 2015

Figure 8-38. Test and debug by using the Test Client (2 of 2)

WM666 / ZM6661.0

### Notes:

4. The next step for starting the Test Client is to identify the integration node to use for the test. You can select an integration node from the list of known integration nodes, or click **Connect remote integration node**.
5. Click **Trace and debug** to enable tracing and the message flow debugger. This option assumes that you set at least one breakpoint in the message flow and that the debug port is configured for the integration server.

Optionally, you can click **Stop at the beginning of the flow during debugging** if you want the debugger to stop at the beginning of the flow instead of at the first breakpoint.

## Test Client component trace

- Provides a message node level trace
  - Monitors the message as it passes through the message flow
  - Helps to quickly pinpoint the problematic area in the message flow
- Receives trace information from the Java debug port that is configured for the debugger

© Copyright IBM Corporation 2015

Figure 8-39. Test Client component trace

WM666 / ZM6661.0

### Notes:

The Test Client provides a function that traces a message flow as it runs, and records the output from that trace into the Test Client. It shows the path that the message flow took, and records all debug information that was generated throughout the flow.



#### Warning

Do not use the Test Client component trace on messages greater than 1 MB, as you might experience performance problems in the workspace because of excessive memory requirements.

## Test Client trace events

- **Node Exit** occurs when a message flows through the connection, as it passes from one message node to another
- **Node Exception** occurs when an exception is raised at the node; it is not handled and is rolled back to the input node
- Trace events data provides information about:
  - The message node and the terminal where the message entered or exited
  - The message contents (Root tree and ExceptionList)
- Trace events can be saved in a test file (.mbtest) when the Test Client is saved
  - Open the test file later to view events
  - Move the test file to another workspace without losing any information

© Copyright IBM Corporation 2015

Figure 8-40. Test Client trace events

WM666 / ZM6661.0

### Notes:

The two key trace points in the Test Client events are “Node Exit” and “Node Exception.” If an exception is raised, but the message flow does not handle it, the exception data is caught and stored in the Test Client.

The Test Client component trace collects the message properties, message headers, and contents of the message body as it passes through the flow, from node to node. As with other aspects of the test client, the message properties can be saved into the .mbtest file.

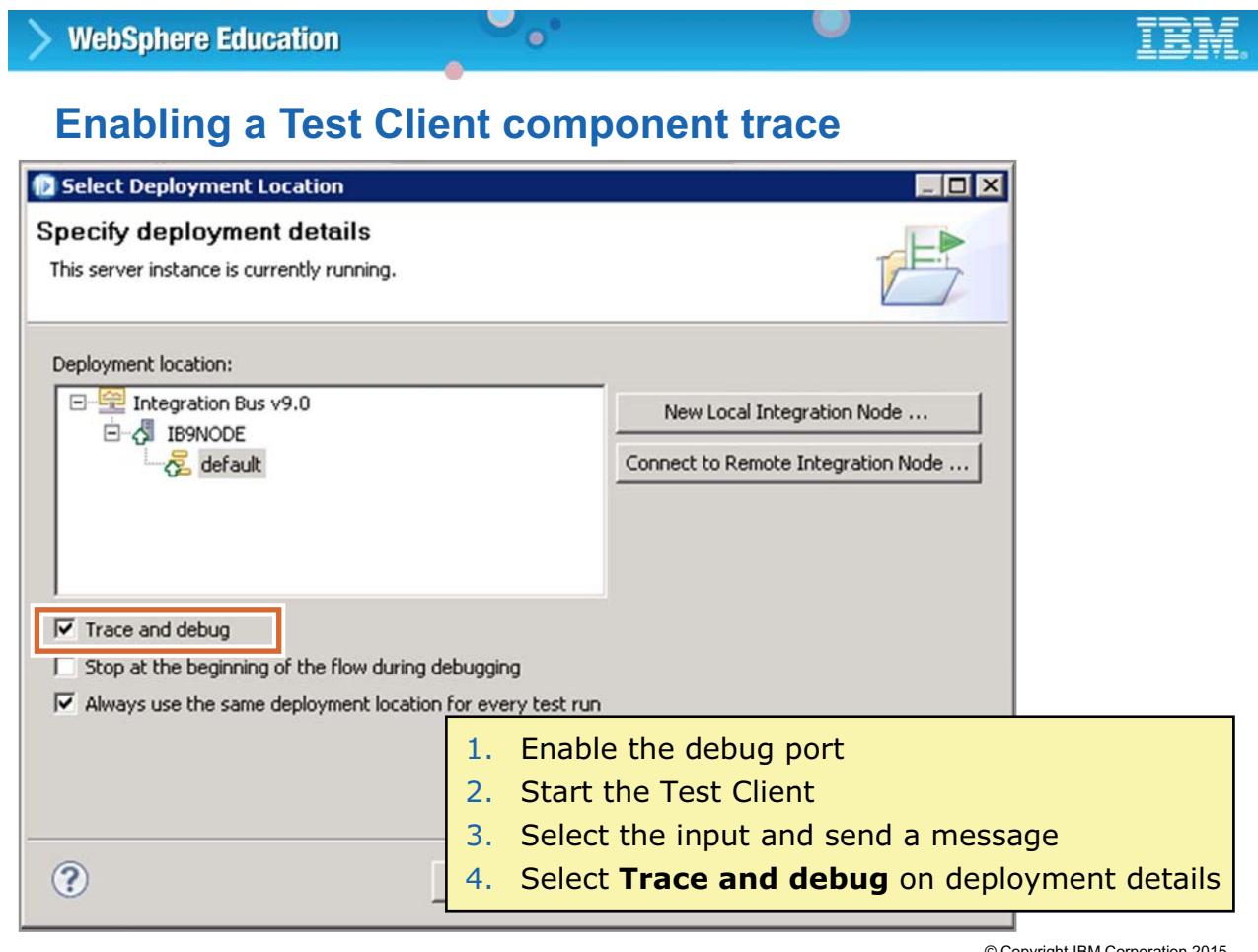


Figure 8-41. Enabling a Test Client component trace

WM666 / ZM6661.0

### Notes:

The Test Client requires that the debug port is configured in advance.

To enable component trace in the Test Client, select **Trace and debug** on the **Select Deployment Location** dialog box.

## Sample component trace: Successful flow

The screenshot shows the WebSphere Education interface with the following components:

- Message Flow Test Events:** A tree view of events. The "Starting" node is expanded, showing:
  - Sending Message to MQ Queue "LAB1\_IN"
  - Message from [LAB1\_IN Cobol : out] to [Transform\_to\_XML : in]
  - Message from [Transform\_to\_XML : out] to [MQ Queue Monitor "LAB1\_OUT"]
  - MQ Queue Monitor "LAB1\_OUT"
- General Properties:** A section with a header "General Properties". It contains a table with the following rows:
 

Message Flow:	/LAB1/LAB1Flow.msgflow
Source message node:	Transform_to_XML
Source terminal:	out
Target message node:	MQOutput
Target terminal:	in
- Detailed Properties:** A section with a header "Detailed Properties". It contains a table with the following rows:
 

Message	Viewer: XML Structure
Name	Value
+ Prop	
+ MQM	
+ MRM	
- Message:** A table with columns "Name" and "Value". It contains three rows:
 

Name	Value
+ Prop	
+ MQM	
+ MRM	

A yellow callout box with the text "Click the hyperlinks to select the corresponding message node and the terminal for the selected event" points to the "Message Flow:" link in the General Properties table.

© Copyright IBM Corporation 2015

Figure 8-42. Sample component trace: Successful flow

WM666 / ZM6661.0

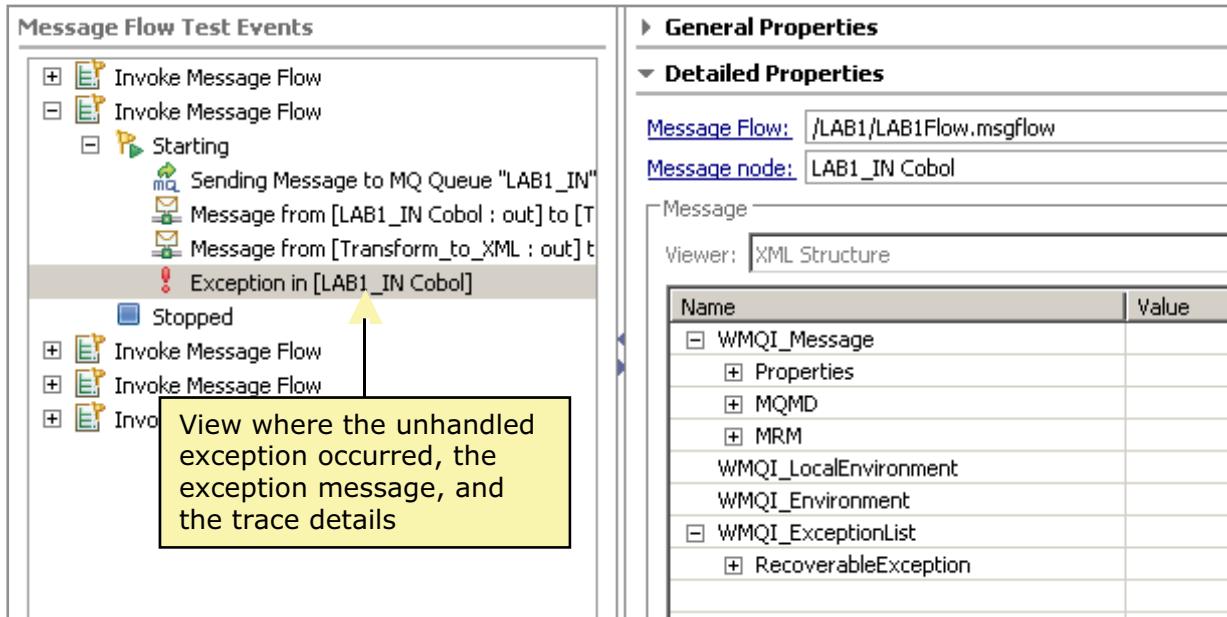
### Notes:

The figure shows a flow that completed successfully.

Each line under the **Message Flow Test Events** column indicates that control moved from one node to the next. Highlighting each node enables the message data to be displayed in the right pane. You can then click any of the hyperlinks to get more information about the message flow, source message node, source terminal, target message node, and target terminal.

 WebSphere Education 

## Sample component trace: Failed flow



**Message Flow Test Events**

- + Invoke Message Flow
- Invoke Message Flow
  - Starting
    - Sending Message to MQ Queue "LAB1\_IN"
    - Message from [LAB1\_IN Cobol : out] to [T]
    - Message from [Transform\_to\_XML : out] to [T]
    - Exception in [LAB1\_IN Cobol]
- + Stopped
- + Invoke Message Flow
- + Invoke Message Flow
- + Invoke

View where the unhandled exception occurred, the exception message, and the trace details

**General Properties**

**Detailed Properties**

Message Flow: /LAB1/LAB1Flow.msgflow

Message node: LAB1\_IN Cobol

Message

Viewer: XML Structure

Name	Value
WMQI_Message	
Properties	
MQMD	
MRM	
WMQI_LocalEnvironment	
WMQI_Environment	
WMQI_ExceptionList	
RecoverableException	

© Copyright IBM Corporation 2015

Figure 8-43. Sample component trace: Failed flow

WM666 / ZM6661.0

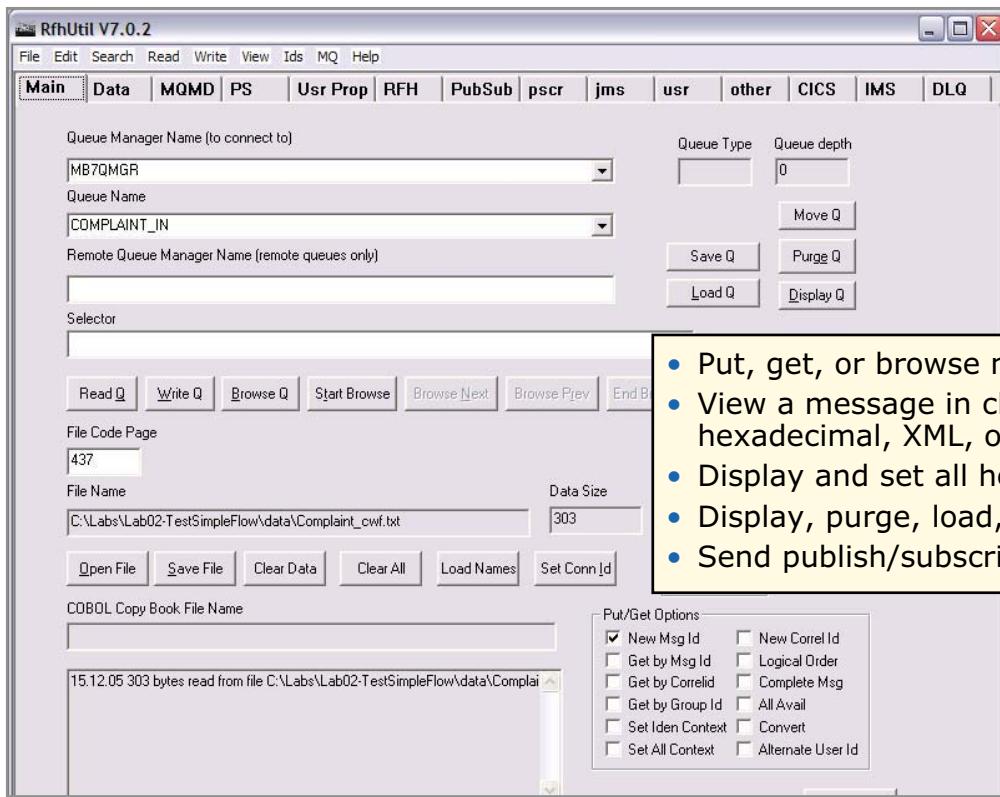
### Notes:

This figure shows a message flow that failed.

As with the successful flow, all message data is available for viewing. In addition, at the point of failure, the exception message is available in the Test Client. This information is the same as the information that would be available in a user trace file.



## RFHUtil (SupportPac IH03)



© Copyright IBM Corporation 2015

Figure 8-44. RFHUtil (SupportPac IH03)

WM666 / ZM6661.0

### Notes:

The RFHUtil program is provided in IBM MQ SupportPac IH03. RFHUtil reads data from files, queues or both. It also writes data to files, queues or both, and displays data in various formats. You can use RFHUtil with IBM Integration Bus to put and get messages to message queues when testing message flows that contain IBM MQ nodes.

The user data portion of the message can be displayed in various formats, but it cannot be changed. Another program must be used to create or change the user data.

The utility program can add rules and formatting headers (RFH) to messages or files. It writes and formats these headers when they are found in messages or files that it reads. The headers can include publish/subscribe commands.

RFHUtil runs on Windows only.

You can download the IH03 SupportPac that contains RFHUtil from the IBM MQ SupportPac website at: <http://www.ibm.com/support>.

The screenshot shows the IBM Knowledge Center interface for the IBM Integration Bus Version 10.0.0. The left sidebar contains a 'Table of Contents' with various sections like Welcome, Start here, Product overview, Scenarios, etc. The main content area displays the 'IBM Integration Bus Version 10.0' welcome page, which includes a brief introduction and instructions about how to use the documentation. At the bottom of the main content area, there are three navigation links: 'Getting started' (with a blue arrow icon), 'Common tasks' (with a document icon), and 'Troubleshooting and support' (with a wrench and gear icon). A yellow callout box highlights the 'View, browse, and search online information' section, which includes two sub-points: 'Within the IBM Integration Toolkit' and 'From the Internet on the IBM public website' with the URL [http://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 2015

Figure 8-45. IBM Knowledge Center

WM666 / ZM6661.0

## Notes:

It is important to know where to go to get more information about the product and product application. The IBM Knowledge Center contains a complete set of online application documentation.

The IBM Knowledge Center for IBM Integration Bus includes a “Troubleshooting and support” section that describes some common problems and solutions.

## Integrated Help

- Press **F1** in Windows; press **Shift + F1** in Linux and UNIX or
- Click **Dynamic Help** from the **Help** menu or
- Click the “Information” icon

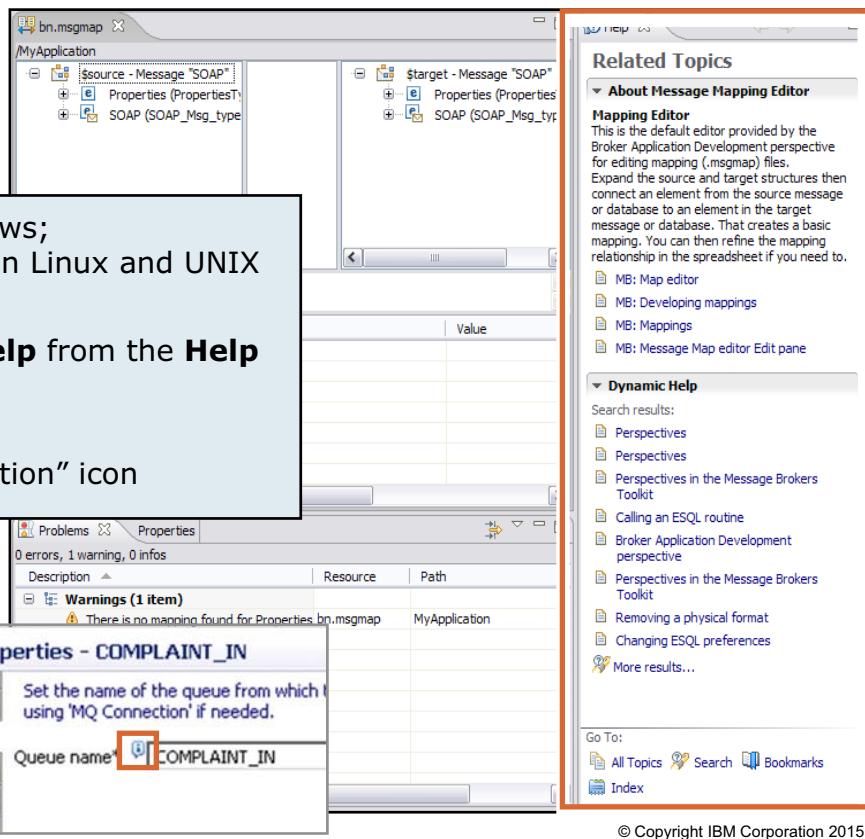


Figure 8-46. Integrated Help

WM666 / ZM6661.0

### Notes:

A Help view is provided in the IBM Integration Toolkit. To view the Help view, press F1 in Windows, press SHIFT+F1 in Linux, or click **Help > Dynamic Help** in the IBM Integration Toolkit.

The default view for Help contains **Related Topics**. In the top section, there is a small description of the interface element (such as a view) or an object (such as a node) that you last selected with the mouse. This description also contains links to topics in the IBM Knowledge Center. Click a link to open any topic in the Help view. Right-click the topic link and click **Open in Help Contents** to open the topic in the IBM Knowledge Center. The topic Help opens in a separate window.

The bottom section of the Help View contains relevant links from the IBM Knowledge Center. These links are generated by using a search on the values of the perspective you are in and the name of the interface element that you selected.

Context-sensitive help is also available for node properties, as indicated by the “Information” icon.

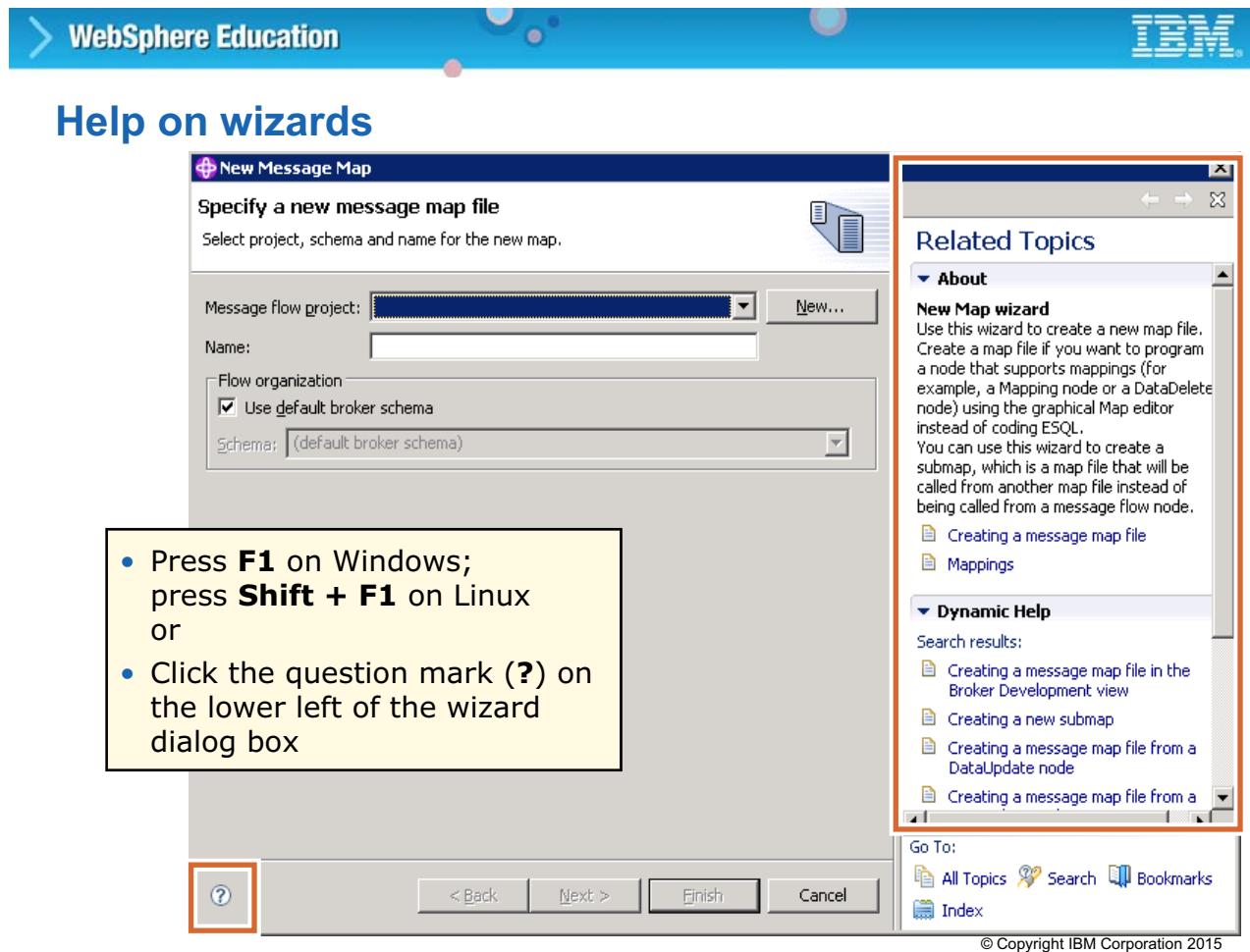


Figure 8-47. Help on wizards

WM666 / ZM6661.0

## Notes:

You can also open the Help view on wizards to display related information and links. Either click the ? icon on the lower left of the wizard, press F1 in Windows, or press SHIFT+F1 in Linux to display the Help view on the wizard.

When you click any of the related topics links in the Help view, the topic is also displayed on the wizard. You can right-click the link to a topic and click **Open in Help Contents** to open the topic in the IBM Knowledge Center.

## Unit summary

Having completed this unit, you should be able to:

- Use the TryCatch and Throw nodes to implement explicit error handling within a message flow
- Describe the structure of the ExceptionList component of the message assembly, and the role it plays in runtime error handling
- Use problem determination tools to debug message flows
- Use help resources to learn more about the product and find information about resolving problems

© Copyright IBM Corporation 2015

Figure 8-48. Unit summary

WM666 / ZM6661.0

### Notes:



## Checkpoint questions

1. Which message flow configuration might cause the loss of a message in IBM Integration Bus?
  - a. An output node is not wired
  - b. No **Catch** path exists
  - c. The **Failure** terminal on the output node is not wired
  
2. Which problem determination tools show the path of the message through the flow?
  - a. IBM Integration Bus activity log
  - b. Integration Toolkit Test Client
  - c. Integration Toolkit Flow exerciser
  - d. Integration Toolkit Message flow debugger

© Copyright IBM Corporation 2015

Figure 8-49. Checkpoint questions

WM666 / ZM6661.0

### Notes:

Write your answers here:

- 1.
  
- 2.

## Checkpoint answers

1. Which message flow configuration might cause the loss of a message in IBM Integration Bus?

- a. An output node is not wired
- b. No CATCH path exists
- c. The FAILURE terminal on the output node is not wired

**Answer: a.**

2. Which problem determination tools show the path of the message through the flow?

- a. IBM Integration Bus activity log
- b. Integration Toolkit Test Client
- c. Integration Toolkit Flow exerciser
- d. Integration Toolkit Message flow debugger

**Answer: c and d.**

© Copyright IBM Corporation 2015

Figure 8-50. Checkpoint answers

WM666 / ZM6661.0

### Notes:

## Exercise 7

Using problem determination tools



© Copyright IBM Corporation 2015

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

10.1

Figure 8-51. Exercise 7

WM666 / ZM6661.0

### Notes:



## Exercise objectives

After completing this exercise, you should be able to:

- Enable a user trace and retrieve the collected trace data
- Add a Trace node to a message flow application
- Use RFHUtil to send test data to a message flow and view messages on an IBM MQ queue
- Use the IBM Integration Toolkit Test Client and message flow debugger view to step through a message flow application
- Examine the IBM Integration Bus logs and system logs to diagnose problems

© Copyright IBM Corporation 2015

Figure 8-52. Exercise objectives

WM666 / ZM6661.0

### Notes:

See the *Student Exercise Guide* for exercise instructions.

## Exercise 8

Implementing explicit error handling



© Copyright IBM Corporation 2015

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

10.1

Figure 8-53. Exercise 8

WM666 / ZM6661.0

### Notes:



## Exercise objectives

After completing this exercise, you should be able to:

- Implement a generic error handling routine in the form of a subflow
- Use a ResetContentDescriptor node to force the message to be reparsed according to the parser domain that is specified in the node properties
- Use the TryCatch node to provide a special handler for exception processing

© Copyright IBM Corporation 2015

Figure 8-54. Exercise objectives

WM666 / ZM6661.0

### Notes:

See the *Student Exercise Guide* for instructions.



# Unit 9. Mapping messages with the Graphical Data Mapping editor

## What this unit is about

In this unit, you learn how to use the Graphical Data Mapping editor to create and edit graphical data maps.

## What you should be able to do

After completing this unit, you should be able to:

- Use the Graphical Data Mapping editor to map logical messages
- Run message maps within message flows

## How you will check your progress

- Checkpoint questions
- Lab exercise

## References

IBM Knowledge Center



## Unit objectives

After completing this unit, you should be able to:

- Use the Graphical Data Mapping editor to map logical messages
- Run message maps within message flows

© Copyright IBM Corporation 2015

Figure 9-1. Unit objectives

WM666 / ZM6661.0

### Notes:



## Message maps

- Transform a logical message without writing code by providing a visual image of the transformation, and simplifying its implementation and ongoing maintenance
  - Concentrate on structural transformations of the message, not on sequencing steps
- Use a message map to complete any of the following actions:
  - Transform any part of the logical message assembly (headers, properties, body)
  - Enrich a logical message with data available in an external database
  - Modify data in an external database
  - Route a logical message based on data content

© Copyright IBM Corporation 2015

Figure 9-2. Message maps

WM666 / ZM6661.0

### Notes:

Maps provide a simple way to transform logical messages. You transform an input message into an output message with a map, by following these steps:

1. Design and define the map to do the required transformations.
2. Call the map by using a Mapping node that calls the map.

The mapping options available are not the same for all node types. Target assignments can be calculated or combined from various input sources, which can be in the form of messages or database tables, as with many-to-one or many-to-many mapping. Target assignments can also be conditional based on a combination of factors that can be applied to any type of mapping.

You must select one of three different map models, which cannot be changed later:

- Complete message maps properties, all headers, message body, and parts of the LocalEnvironment (DestinationList).
- Simplified message copies all existing headers. You must map only the properties, body, and LocalEnvironment.
- Submaps map (parts of) the message body only; it cannot be used in nodes directly.

It is important to commit changes to message models and database tables *before* using the messages or tables in maps.

In IBM Integration Bus, the Graphical Data Mapping editor is used to create maps.



## Message map implementation

- Used in various message flow nodes
  - Map target = Message (Mapping node)
  - Map target = Database
- Reusable and correspond with nodes
  - Nodes refer to a message map for properties, headers, and body
- Stored in \*.map files and rendered as XML schemas
  - Must be referenced by a message flow application
- Submaps map a complex type in the message body

© Copyright IBM Corporation 2015

Figure 9-3. Message map implementation

WM666 / ZM6661.0

### Notes:

A message map is the IBM Integration Bus implementation of a graphical data map.

Maps are used to map a message in a Mapping node. You can also use a message map to enrich, or conditionally set the output message with data from a database table.

You can use a message map to graphically transform a message assembly, message body, and properties, according to the transforms and XPath functions defined in the message map.

A map is based on XML schema and XPath 2.0 standards. It is stored in a .map file.

A submap is a reusable form of graphical data map. Submaps can use a set of mapping functions in multiple graphical data maps to transform a common set of elements in the input object to the output object. You can use a submap to reuse common data transformations.

## Mapping node

- Use the Graphical Data Mapping editor to construct and populate one or more new messages
  - Build a new logical message
  - Copy logical messages between parsers
  - Transform a logical message from one format to another
- Contained in the **Transformation** drawer of the palette
- **Mapping routine** property contains the statements to run against the database or the logical message tree
- Double-click the node, or right-click the node and then click **Open Map** to open the mapping routine
  - If the mapping routine does not exist, the **New Message Map** wizard opens, with some default values entered
  - If the mapping routine exists, it opens in the Graphical Data Mapping editor

© Copyright IBM Corporation 2015

Figure 9-4. Mapping node

WM666 / ZM6661.0

### Notes:

Use a map with the Mapping node to construct one or more new messages and populate them with various types of information.

You can populate the new messages with:

- New information
- Modified information from the input message
- Information that is taken from a database

You can modify elements of the message body data, its associated environment, and its exception list.

The Graphical Data Mapping editor is used to create the map and define the transformations.

## Graphical Data Mapping editor (1 of 2)

- Map the elements and the attributes between the input and output objects, and then apply a transform to the mapping that specifies the actions on the input object
- Designed as a standard for IBM products
- A rich feature set and simplicity make Graphical Data Mapping editor a good default transformation choice
- Maps are available to user patterns
  - Graphical creation of flows that require transformation logic
  - Invocation of the editor when pattern instances are generated
  - User guidance through the HTML pattern help and task list
- Full XPath 2.0 expression support

© Copyright IBM Corporation 2015

Figure 9-5. Graphical Data Mapping editor (1 of 2)

WM666 / ZM6661.0

### Notes:

The Graphical Data Mapping editor is used to create and edit graphical maps. These maps take input (source) messages and transform them into output (target) messages.

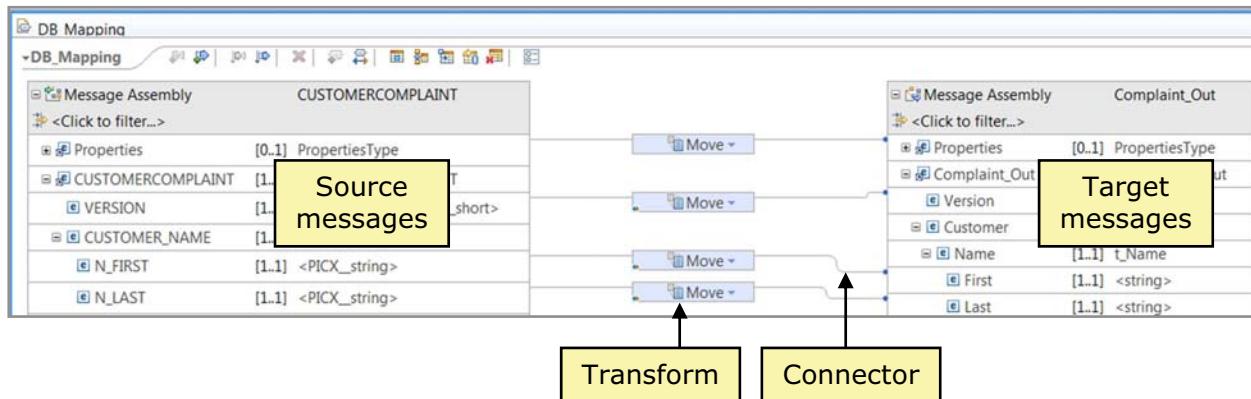
The graphical data map represents the input and output message data. You drag actions to make connections, select transforms, and build logic to transform your message data without programming.

The Graphical Mapping Editor is the standard mapping tool for many IBM products.

Maps are used in many of the IBM Integration Bus patterns.



## Graphical Data Mapping editor (2 of 2)



- Graphical Data Mapping editor is a visual transformation tool
  - Models that are referenced in the map must be in the workspace
  - If a map references a database, a database definition file (.dbm) must be in the workspace

© Copyright IBM Corporation 2015

Figure 9-6. Graphical Data Mapping editor (2 of 2)

WM666 / ZM6661.0

### Notes:

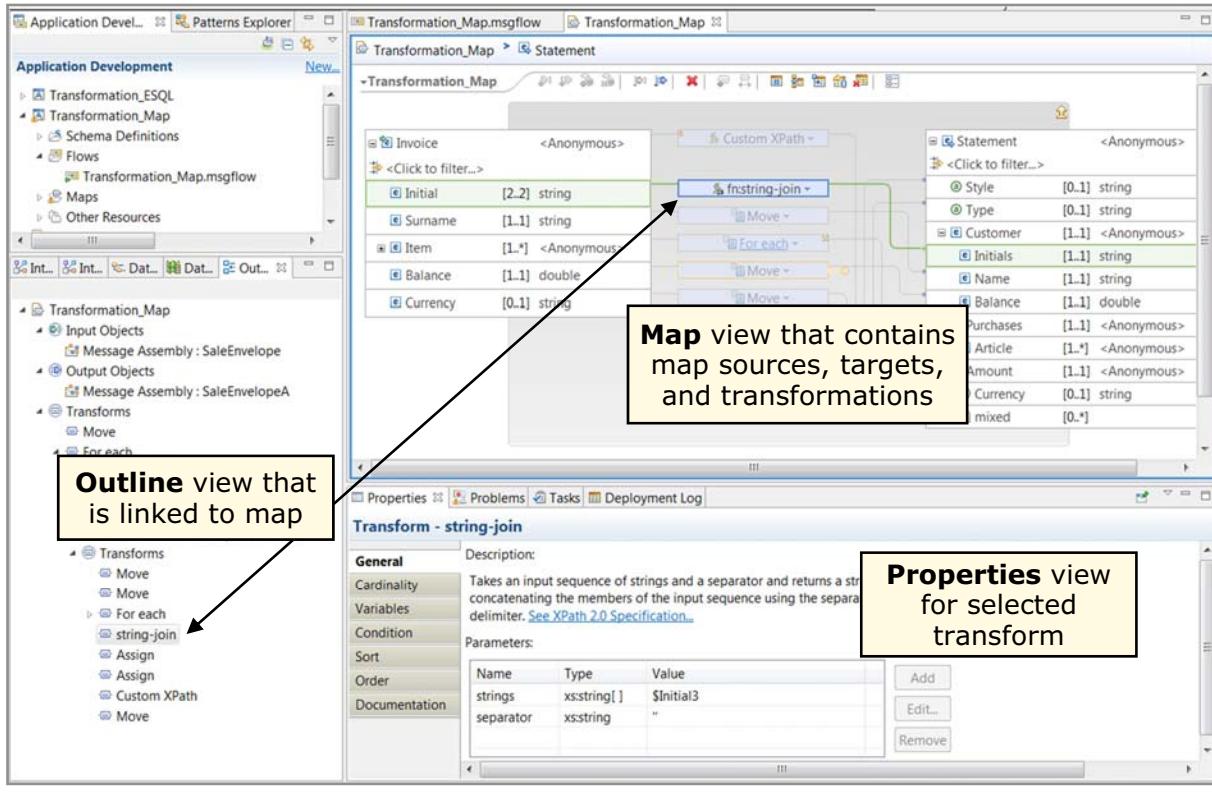
When you use the Graphical Data Mapping editor, the source and target messages are shown in a design window. By drawing connectors from the source to the target, you can specify what fields are copied, and what transforms are applied.

After you create a map with the Graphical Data Mapping editor, the mappings are automatically validated and compiled, and are ready for adding to a BAR file, and subsequent deployment to an integration server.

You must have message models for any messages that you want to include in a mapping. You can select the message model from existing message models in an application or library when you create a message map. The mapping facility supports message models that are provided in DFDL and XML schema files or MRM message sets.



## Map navigation



© Copyright IBM Corporation 2015

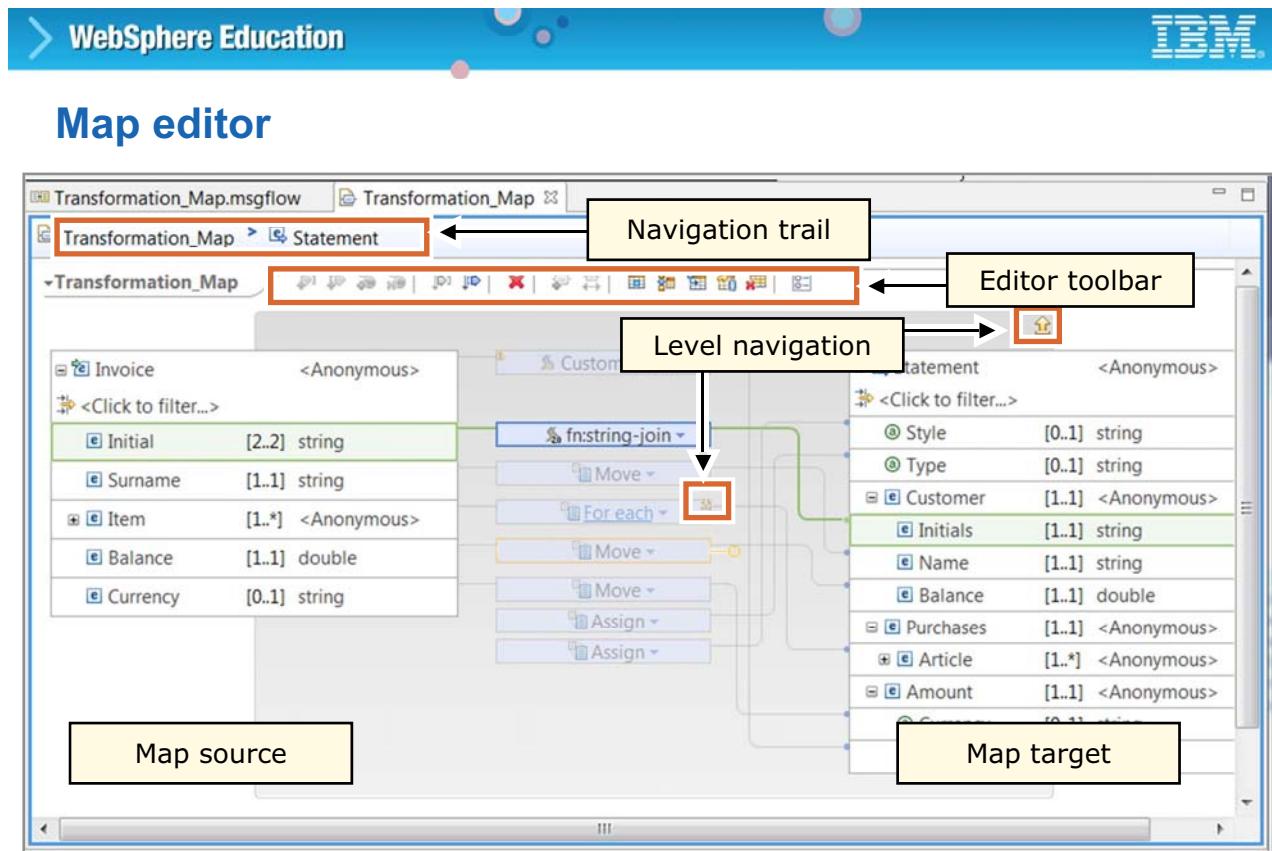
Figure 9-7. Map navigation

WM666 / ZM6661.0

### Notes:

The Graphical Mapping Editor has three main views.

- The **Map** view contains the map sources, map targets, and transformations.
- The **Outline** view is a navigation aid that shows the structure of the map. When you click a transform in the **Outline** view, the **Map** view updates to show the selected transform. By default **Outline** view appears in same view with **Properties**. You can drag the tab so that the view is displayed with Integration nodes.
- Each transform contains a set of properties, which are maintained in the **Properties** view.



© Copyright IBM Corporation 2015

Figure 9-8. Map editor

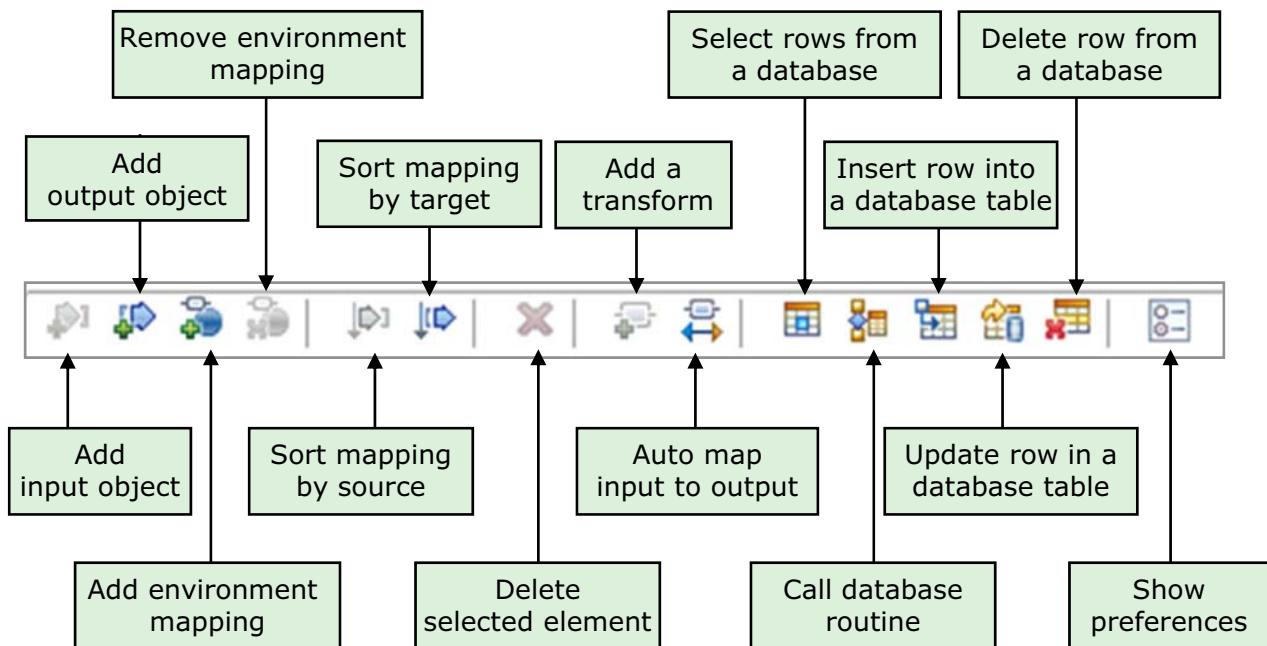
WM666 / ZM6661.0

## Notes:

The **Map** view contains the model of the map source data on the left and the model of the map target data on the right. The editor toolbar helps you define transforms.

If the source and target data contain groups within groups, you might need a submap to map objects in a subgroup. The **Map** view contains level navigation icons and a navigation trail to help you move from the main map to a submap and back up to main map.

## Graphical Data Mapping editor toolbar



© Copyright IBM Corporation 2015

Figure 9-9. Graphical Data Mapping editor toolbar

WM666 / ZM6661.0

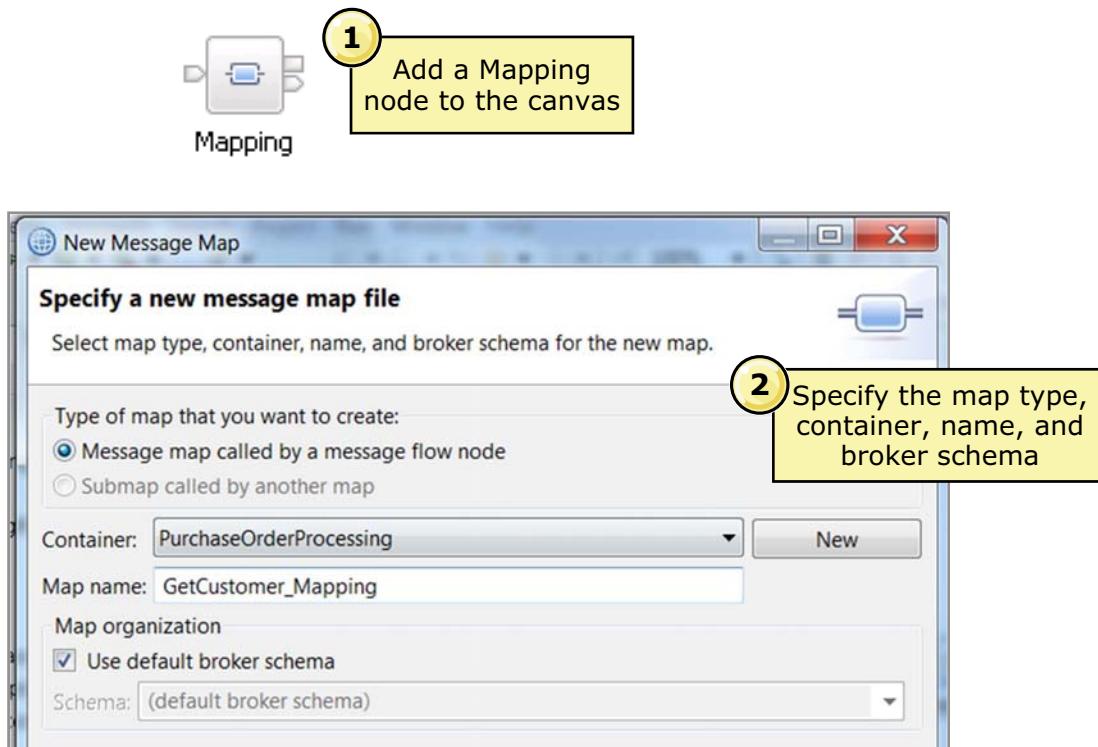
### Notes:

The Graphical Data Mapping editor toolbar can be used to complete many actions, including:

- Adding fields to or deleting fields from a source or target message
- Sorting the fields in the source or target message
- Automatic mapping between the source and target messages. You can control how similar the source and target fields must be (in terms of field names and data types)
- Retrieving rows from a database at run time to insert into the target message



## Creating a map in the IBM Integration Toolkit (1 of 3)



© Copyright IBM Corporation 2015

Figure 9-10. Creating a map in the IBM Integration Toolkit (1 of 3)

WM666 / ZM6661.0

### Notes:

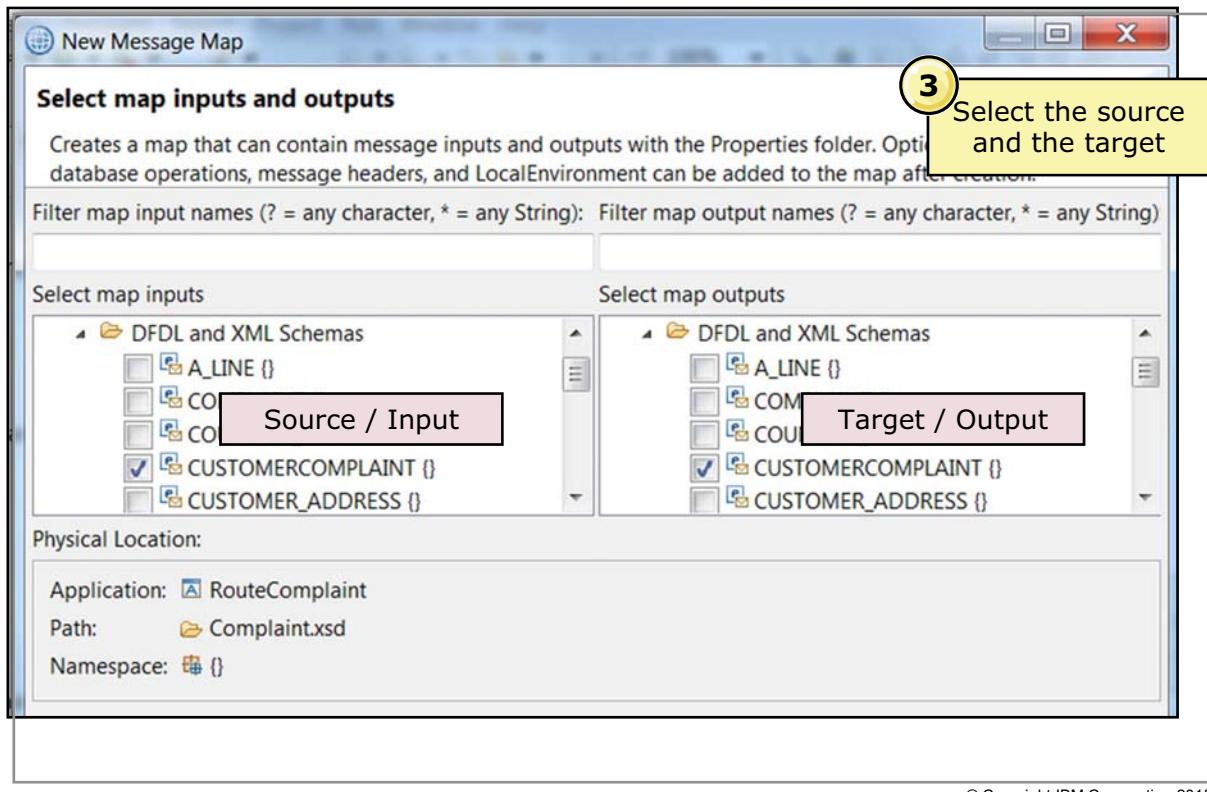
To create a map with the Graphical Mapping editor:

1. In the Message Flow editor, add a Mapping node to the canvas.
2. Double-click the Mapping node. The mapping wizard starts. Set the properties, including the Container and Map name.

You can also create a map from the Integration Development perspective by clicking **File > New > Message Map**. If you use this method, you must select the type of map you want to create:

- If you select **Message map called by a message flow**, a message map is created that can be accessed from a node.
- If you select **Submap called by another map**, a message map is created that can be referenced from another message map.

## Creating a map in the IBM Integration Toolkit (2 of 3)



© Copyright IBM Corporation 2015

Figure 9-11. Creating a map in the IBM Integration Toolkit (2 of 3)

WM666 / ZM6661.0

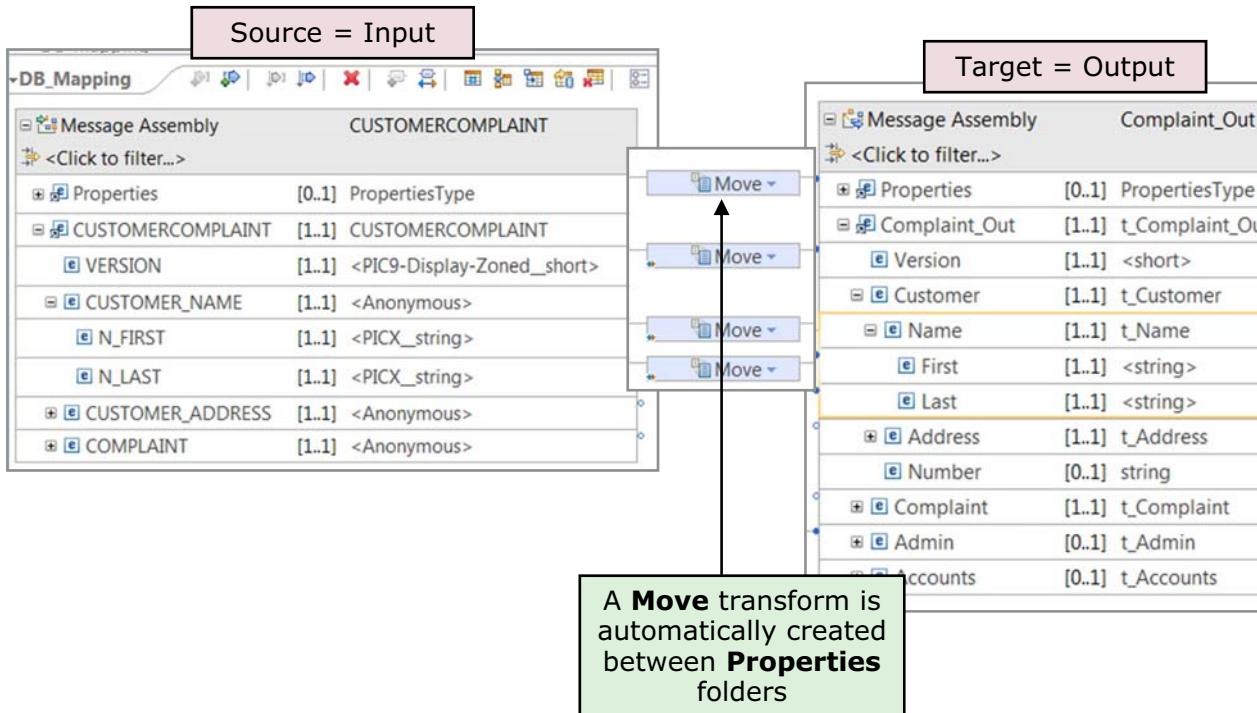
### Notes:

- After you set the properties for the map, you select the source (input) and the target (output) for the map. You can select more than one source and target for a map.

On the **Select map inputs and outputs** pane, expand the list of available input objects and output objects, and then select the objects that you want to use in the map. If necessary, use the **Filter map** fields to filter what is shown in the list of available objects.



## Creating a map in the IBM Integration Toolkit (3 of 3)



© Copyright IBM Corporation 2015

Figure 9-12. Creating a map in the IBM Integration Toolkit (3 of 3)

WM666 / ZM6661.0

### Notes:

After you select the sources and targets, the Graphical Data Mapping editor opens and with the source and target messages.

To create a transform between a source element and the target element:

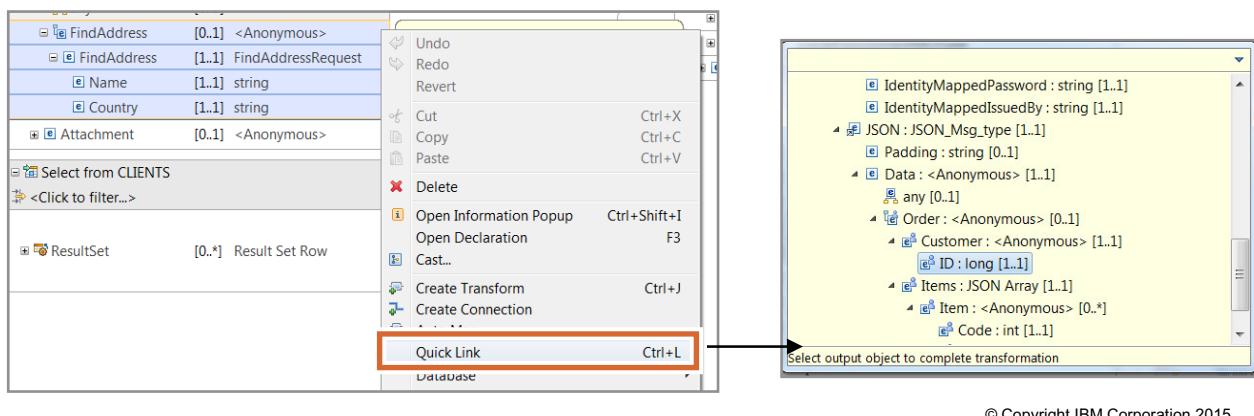
1. Hover the mouse cursor over the object you want in the source. A yellow wiring handle is displayed.
2. Drag the wiring handle to the object in the target you want to use. The connection is made and a default transform is displayed.

To change the transform, click the small downward-pointing arrow in the transform name, and select a new transform.



## Creating transforms in a map

- Option 1: Drag from source to target
- Option 2: Create connection or transform by using menu action on the selected source or target
- Option 3: When you cannot locate the source and target concurrently select **Quick Link** use the quick outline view and its built-in filter to find and select the required element
- After you select the source and target elements, the most applicable transform is created in the Graphical Data Mapping editor



© Copyright IBM Corporation 2015

Figure 9-13. Creating transforms in a map

WM666 / ZM6661.0

### Notes:

The Map editor supports multiple methods for creating a transform in a map.

- You can create a transform in a map by clicking the map source element and dragging to the map target element and then selecting the transform.
- You can create a connection or transform by right-clicking the map source element and then clicking an action from a menu.
- You can click **Quick Link** from a menu to find the source and target elements and create a transform.

## Core mapping transforms

- **Assign** Sets a value in the output element
- **Cast** Sets a specific value type in the output element. Cast can also move and convert an input element to become a specific value type in the output element
- **Concat** Creates a string concatenation to retrieve data from two or more entities and link them into a single result
- **Convert** Copies the input element to the output element and changes the type
- **Create** Creates an empty element, a nil element, or a simple type element by using a default value that is based on the element's type
- **Custom ESQL** Enter user-defined ESQL code to use in the transform
- **Custom Java** Enter user-defined Java code to use in the transform
- **Custom XPath** Enter user-defined XPath expressions or XSLT functions to use in the transform
- **Move** Copies data from the input element to the output element
- **Normalize** Removes white space such as spaces, and tabs from input and moves the resulting normalized string to the output element
- **Substring** Extracts information as required, and moves the extracted string to the output element
- **Task** Describes a manual task or point of concern that might require review or resolution before a graphical data map can be used in a messaging solution
- **XPath** All XPath 2.0 functions are supported, in the form `fn:<function_name>`

© Copyright IBM Corporation 2015

Figure 9-14. Core mapping transforms

WM666 / ZM6661.0

### Notes:

The Graphical Data Mapping editor provides many transformations. A subset of those transforms is listed here.

- String functions include string length, padding, retrieve system property string, convert case, compare, concatenate, substring, translate, and others.
- Custom transformations provide the capability for you to write custom Java or XPath code that you can start at run time.



## Database transforms

- **Database Routine** Calls a stored procedure or user-defined function from a database
- **Delete** Deletes one or more rows in a database table that have a matching Where clause
- **Failure** A map handles errors for any exceptions that the database server raises in a database transform, instead of having such exceptions stop the map and be reported
- **Insert** Inserts a row into a database table
- **Return** Enables more processing after a successful Insert, Update, or Delete database operation
- **Select** Retrieves data from rows in a database table so that the data can be used as input in a graphical data map
- **Update** Updates one or more rows in a database table that have a matching Where clause with a single set of data values

© Copyright IBM Corporation 2015

Figure 9-15. Database transforms

WM666 / ZM6661.0

### Notes:

You can use the database transforms in the Graphical Data Mapping editor to insert new rows of data, or to update or delete existing rows of data, in your database tables.

For each database transform in your graphical data map, the Graphical Data Mapping editor uses a database definition file (.dbm file) to determine the name and structure of the database that you want to access. You can start the wizard to create a database definition file when you create a database transform in a graphical data map.

## Structural mapping transforms

- **Append** Iterates over multiple inputs in the order that is specified to append data
- **Remove** Removes an element
- **ForEach** Iterates over an input array element (either a simple type or a complex type)
- **Group** Takes a single input array and produces a set of nested output arrays that collate elements of the input array
- **If, Else if, Else** Controls the flow of the mapping by setting conditions
- **Join** Joins elements from two or more inputs
- **Local map** Provides a hierarchical view of element transforms in the graphical data map
- **Submap** References another map from this map or another map file, which can be stored in a library, an application, or an Integration project

© Copyright IBM Corporation 2015

Figure 9-16. Structural mapping transforms

WM666 / ZM6661.0

### Notes:

In addition to the mapping transformations that were described previously, several structural transforms are provided in the Graphical Data Mapping editor. The structural transforms control how nested elements are displayed in the Graphical Data Mapping editor, but they have no effect on the data itself.

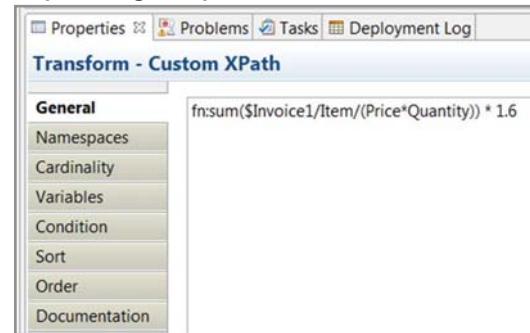
The structural transforms include:

- **If, Else if, Else:** Allows control of flow of the mapping by setting conditions. If, Else if, and Else operate as a group of conditional transforms. The condition is applied to the input element of the conditional transform. If the condition is satisfied, the transform that is nested within the conditional transform is run.
- **Local map:** A transform with only one element as input (simple type or complex type) that can contain nested maps. Target can be either a single element or an array element, but must be complex type.



## Transform properties

- Each transform is configured in the **Properties** view
  - **General** tab specifies the value or expression that determines the value of the output element
  - **Variables** tab shows variable names for input connections that are used in expressions and to set the value of an output
  - **Condition** tab defines the XPath expression that determines whether to apply a transform
  - **Order** tab reorders the input connections for transforms such as Append
  - **Documentation** for documenting the transform
- Extra **Properties** that determine the value of a repeating output element
  - **Cardinality** tab specifies the elements in an array to transform
  - **Filter Inputs** tab specifies the expression that determines whether each repeating element is applied or not
  - **Sort** tab reorders the input array connections
- Press Alt+Enter to open **Properties** view when Map Editor is full screen



© Copyright IBM Corporation 2015

Figure 9-17. Transform properties

WM666 / ZM6661.0

### Notes:

Configure the properties of a transform to:

- Set the value of an output element
- Define a conditional expression that determines whether the transform is applied
- Define the indexes to use when the input element is a repeating structure
- Reorder how the Graphical Data Mapping editor handles inputs to a transform

The transform type determines the property tabs that are available in the **Properties** view.



## Automatically mapping input to output elements

- Use the **Auto map** wizard to automatically create **Move** transforms from input to output elements based on some correlation of the names of input and output elements to create mappings
  - Map all simple descendants of the selected elements
  - Map the immediate child elements of the selected element
  - Specify how names are matched in the **Name Matching Options** section
- In the **Mapping Criteria** section, specify the mapping criteria for the matches between the input and output element names
  - Create transforms when the names of inputs and outputs are the same
  - Create transforms when the names of inputs and outputs are similar
  - Create transforms when the input and output names are matched to synonyms defined in a file

© Copyright IBM Corporation 2015

Figure 9-18. Automatically mapping input to output elements

WM666 / ZM6661.0

### Notes:

You can use the **Auto map** wizard to automatically create **Move** transforms from input to output elements based on some correlation of the names of input and output elements to create mappings.

You can choose to map all simple descendants of the selected elements or map the immediate child elements of the selected elements.

- Selecting **Map all simple descendants of the selected elements** maps the descendants of the input element to the descendants of the output element that match each other; this option is selected by default.
- Selecting **Map the immediate children of the selected elements** maps only the immediate child elements of the input element to the immediate child elements of the output element that match each other.

You can also specify how names are matched in the **Name Matching Options** section. The two options are independent of each other, and you can select their values separately:

- Use **Case sensitive** to select whether you want to match the case sensitivity of the name.

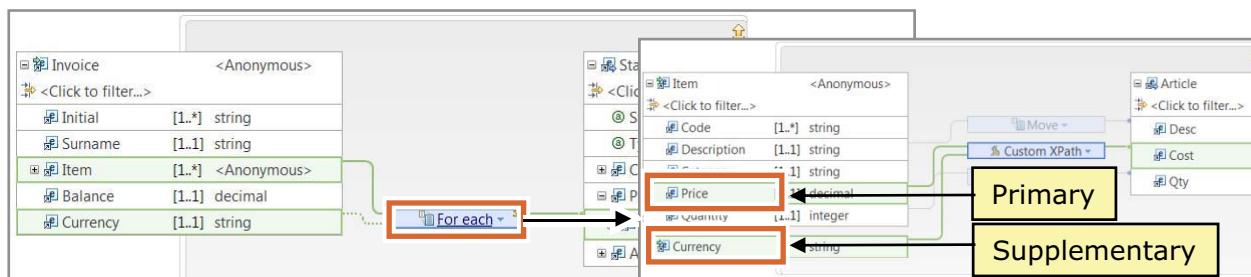
- Use **Alphanumeric characters (Letters and digits only)** to exclude special characters from the name.

In the **Mapping Criteria** section, specify the mapping criteria for the matches between the input and output element names:

- Selecting **Create transforms when the names of inputs and outputs are the same** matches items of the same name, and is selected by default.
- Selecting **Create transforms when the names of inputs and outputs are more similar than** specifies how similar two names must be to create a mapping between them.
- Selecting **Create transforms when the input and output names are matched to synonyms defined in a file** creates mappings for word pairs that are defined in a synonym file. A synonym file is a flat text file with the file extension .txt or .csv.

## Primary and supplementary inputs and connections

- Primary inputs are directly involved in the transformation:
  - Contain the data to produce the output
  - Can also be used to define properties of the transform
- Supplementary inputs are used to pass elements into a transform without affecting the primary purpose of that mapping:
  - NOT used directly in the transformation
  - Can be used to define transform properties
  - Can have as many supplementary inputs as required
  - Can be used to pass extra data into a nested map



© Copyright IBM Corporation 2015

Figure 9-19. Primary and supplementary inputs and connections

WM666 / ZM6661.0

### Notes:

A local map can have one primary input and multiple supplementary inputs, which can be either a simple type or a complex type. You must map the elements in the nested map; otherwise, no action is taken when the transform runs.

In the example, the **For Each** transform creates a nested local map. The **For Each** transform can have one primary input that contains the data that is used to produce the output and multiple supplementary inputs. You can use the supplementary inputs in any of the following ways:

- To create a **Filter Inputs** expression based on the value of the input. You can use it to determine which indexes to apply as part of the transformation.
- To pass an extra element into the nested transform. This input is available in the mapping of each iteration that the **For Each** transform runs.
- To configure the **Cardinality** property page on the **For Each** transform to indicate which index of an input array to iterate over.

In the example, the **For Each** transform uses **Price** as a primary input and **Currency** as a supplementary input to calculate the **Cost**.



## Using XPath 2.0 in the Graphical Data Mapping editor

- Use XPath expressions to define conditional logic
  1. Define an XPath expression in the **Condition** property tab of the transform
  2. If the expression evaluates to 'true', then the transform is evaluated and the target is produced
- Use XPath expressions to define numeric calculations
  - Most common mathematical operators such as sum, count, round, maximum, and minimum are built into the XPath syntax
- Use XPath expressions to manipulate strings
  - Extensive function library for operations such as string splitting, joining, and comparing
- Use the **Custom XPath** transform to define more complex combinations of functions and expressions
- Graphical Data Mapping editor provides content assist

© Copyright IBM Corporation 2015

Figure 9-20. Using XPath 2.0 in the Graphical Data Mapping editor

WM666 / ZM6661.0

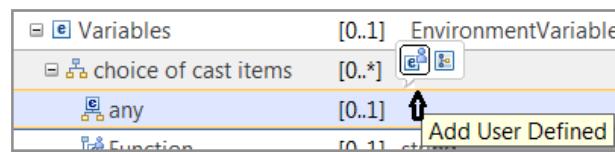
### Notes:

XPath expressions are used in a map to define conditional logic, numeric calculations, and manipulate strings. In the Graphical Data Mapping editor, you can use XPath 2.0 functions and expressions.

## User-defined elements

- An element that is added to a message map to define any of the following message assembly components:

- JSON messages
- Global or Local Environment
- **xsd:any** elements in message body or headers



- You can add any of the following types of user-defined elements:
  - Simple type
  - Complex type elements
  - Repeatable

© Copyright IBM Corporation 2015

Figure 9-21. User-defined elements

WM666 / ZM6661.0

### Notes:

A user-defined element is an element that you can add directly into a message map to define the data of an extension point, which is shown as `any` or `anyAttribute` in the map.

You can add, reuse, rename, transform, and delete a user-defined element directly into your message map during the development phase.

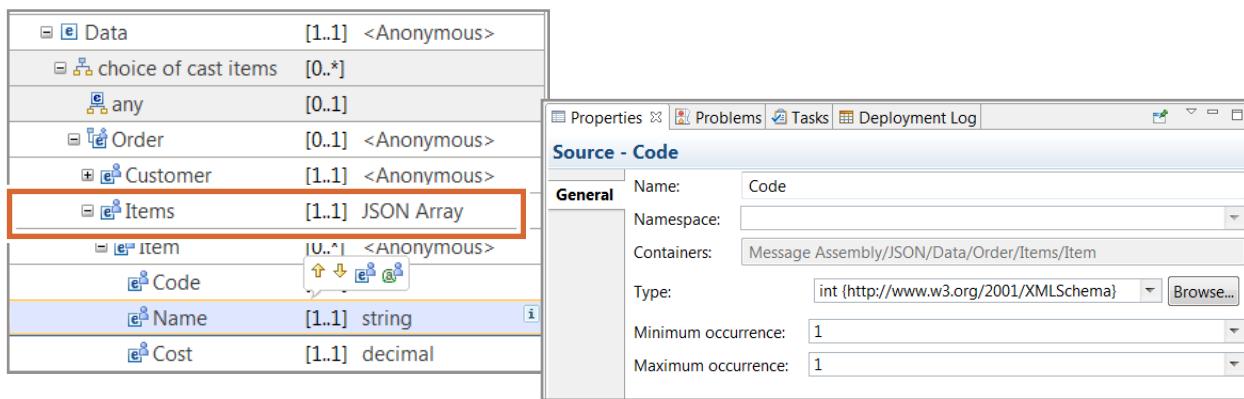
You can add user-defined elements to define extension points in any of the following message assembly components:

- The local environment tree variables folder
- The message body
- The `xsd:any` elements in your message body
- The Environment tree
- The transport headers that include extension points such as MQRFH2

You can also use user-defined elements to define a JSON message, and a SOAP or XML message that has an `xsd:any`.

## Considerations when adding user-defined elements

- Valid options:
  - Defining a simple type or a complex user-defined element
  - Defining user-defined attributes within a complex user-defined element
  - Adding a repeatable simple type element within a complex type
  - Reordering user-defined elements
  - Copying and pasting user-defined elements
  - Adding the JSON-specific array type



© Copyright IBM Corporation 2015

Figure 9-22. Considerations when adding user-defined elements

WM666 / ZM6661.0

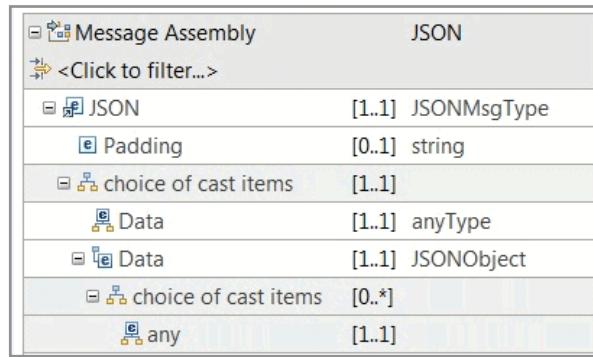
### Notes:

When you add a user-defined element, you must consider the following behavior:

- You can define a simple type user-defined element or a complex user-defined element in the input message assembly or in the output message assembly.
- You can include simple type attributes, simple type elements, complex type elements, and repeatable elements in a user-defined complex type element.
- You can define user-defined attributes within a complex user-defined element.
- You can set the type of a user-defined element or attribute from an existing global type definition. You can use any available message models, XML schemas, or DFDL schemas.
- JSON elements are defined as nullable by default to allow the JSON null value. The Graphical Data Mapping editor creates all JSON user-defined elements as nullable.

## Mapping JSON object messages

- Can select JSON message map for source or target
- JSON message consists of a single element of type **JSONMsgType** that contains child elements
  - **Padding** element (optional) is used to pass a user-defined JavaScript function call used by JSONP services
  - **Data** element to model JSON message
- **Data anyType:** Define this element when you have a schema model that defines your JSON message
- **Data JSONObject:** Define this element by using the **Add User-Defined** function when you want to transform some elements of the JSON object message



© Copyright IBM Corporation 2015

Figure 9-23. Mapping JSON object messages

WM666 / ZM6661.0

### Notes:

When you create a message map, you can choose to create a JSON object message or a JSON array message. Predefined models are available under the IBM supplied message models category in the map input and output selection tree. When you select any of the JSON message models as the map output, the output domain in the output message assembly is automatically set to **JSON**.

You select **JSON (JSON object message model supplied by IBM)** to create a JSON object message. The predefined JSON message contains a single element value of type **JSONMsgType**. This element contains the following child elements:

- A **Padding** element: This element is optional. You define its value to pass a user-defined JavaScript function call used by JSONP services.
- Two **Data** elements: You define only one of the **Data** elements to model your JSON message.

You define the element **Data**, of type **anyType**, by using the Cast function. You define this element when you have a schema model that defines your JSON message. If you create your map programmatically, you can cast **Data** to some type in an external schema file, which can define either a JSON object or a JSON array message.

You define the element **Data**, of type **JSONObject**, by using the **Add User-Defined** function. You define this element when you want to transform some elements of your JSON object message. You can also use the Cast function to define the JSON message, or a part of it, with a schema model.



## Tips for choosing a Transform type for JSON messages

- Use **Assign** to set a fixed value without the use of input data
- Use **Create** to create an empty value, or the JSON NULL value (nil) without the use of input data
- Use **Move** to copy an input element value to the output element
- Use **If**, **XPath**, or a custom transform to define the condition that determines the value of an output element
- Use **Append**, **Group**, **For each**, **Join**, **XPath**, or a custom transform to set the value of a JSON array
- Use **Select** or **Database Routine** to set the value with data that is available in a database
- Use **xs:type** to cast the value of a simple input element to a specific data type
- Use **fn:type** to set the value by using XPath 2.0 functions

© Copyright IBM Corporation 2015

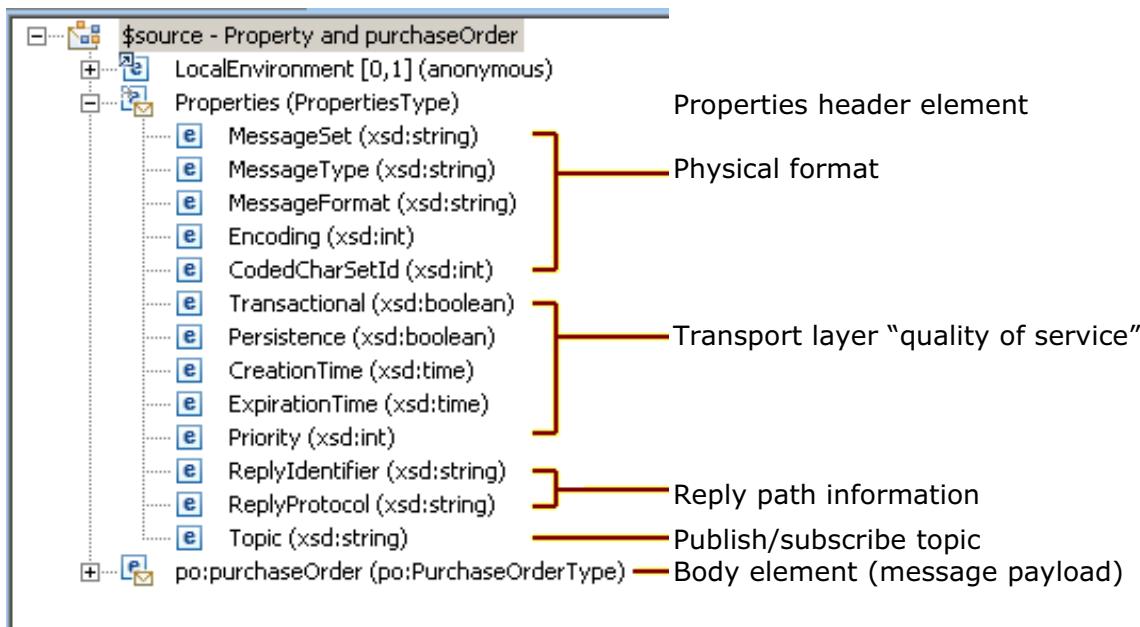
Figure 9-24. Tips for choosing a Transform type for JSON messages

WM666 / ZM6661.0

### Notes:

This figure lists some tips for defining transforms between the input elements and the output elements for JSON messages.

## Properties mapping



© Copyright IBM Corporation 2015

Figure 9-25. Properties mapping

WM666 / ZM6661.0

### Notes:

The message assembly can also contain:

- The **Properties** tree, which the integration node predefines
- Protocol-specific transport headers, which might have to be manipulated as part of the mapping transformation
- **Local environment**, which is used for functions such as dynamic setting of a message destination (routing)

By default, the Mapping node copies the properties and headers from input to output, unchanged.

The figure shows the simplified message assembly within the **Properties** folder in the traditional mapping editor, with some comments that describe the content of the virtual header. The **Properties** folder must be mapped whenever an output message based on the MRM parse tree is required. If the mapping is not done, the message is not fully propagated. Mapping is not required for other parser domains.

## Mapping the Environment tree

- You can add Environment tree to the message map
  - Model the content that is required in a map by using User-Defined elements or casts
  - Update, create, or remove elements by using transformations between elements in the Environment tree and any elements in the message assembly
- All Environment tree content is always transferred from input to output

© Copyright IBM Corporation 2015

Figure 9-26. Mapping the Environment tree

WM666 / ZM6661.0

### Notes:

The LocalEnvironment tree is a part of the logical message tree in which you can store information while the message flow processes the message. You use the LocalEnvironment tree to store variables that can be referred to and updated by message processing nodes that occur later in the message flow. You can also use the LocalEnvironment tree to define destinations (that are internal or external to the message flow) to which a message is sent.

When you add the LocalEnvironment tree to a message map, you must provide transforms for all of its elements so that the input values of each element are not lost. You can copy the input field unchanged or modified by a transform. Many IBM Integration Bus nodes depend on information in the local environment tree that is copied during the message flow.

The Mapping node copies the LocalEnvironment trees from input to output, unchanged, when they are not included in the message map.

## Deploying a message map

- By default, message map files are deployed in BAR files as individual resources
  - If you change a message map, you must redeploy the integration solution
- IBM Integration Bus prepares message maps for run time at deployment instead of when the first message is flowed through the Mapping node
  - There is no drop in performance from initializing a message map when the first message is flowed through the node after a deployment or restart
  - Message map and its dependencies, such as any referenced message models, are resolved and validated during deployment to ensure that the message map runs successfully on first message
  - Message map syntax is validated during deployment to ensure that the message map runs successfully on first message
  - When IBM Integration Bus is restarted, the message map syntax and its dependencies are validated before the message flow can be restored

© Copyright IBM Corporation 2015

Figure 9-27. Deploying a message map

WM666 / ZM6661.0

### Notes:

By default, message map files are deployed in BAR files as a part of an application, integration service, or library that provides an integration solution. You can also deploy a map as an independent resource if you are managing your message flows that way. If you change a message map, you must redeploy your integration solution, or independent message flows.

When you deploy message maps, all message maps are validated to ensure that all the map dependencies can be resolved at run time. This validation step checks that the referenced message models such as XML schema files, DFDL schema files, message set files, and the referenced submaps can be resolved.

Next, message maps and their dependencies are generated to an executable form. This step also checks that the contents of the map and submaps are valid, and that they are error-free. If the message maps and their dependencies are valid and can be successfully generated, they are persisted in both the deployed and generated forms to the configuration store. Otherwise, the deployment is stopped and you receive a BIP message that reports the map generation failure.

## Message maps and shared libraries

- When a logical message is validated against deployed XML schema files in shared libraries, you must specify the name of the shared library that contains the appropriate compiled model
  - **Properties.MessageSet** property contains the name of the shared library in which the schemas used in a map are available
- Store map files in a shared library when:
  - You need to use the same map files in multiple solutions
  - You deployed a map, and you want to update it without redeploying the application or integration service



Changes to the data model can affect the map.

© Copyright IBM Corporation 2015

Figure 9-28. Message maps and shared libraries

WM666 / ZM6661.0

### Notes:

The message map can be in the same container as the Mapping node (such as an application), or it can be in a referenced shared library. If multiple solutions are likely to use the same message map, store the map in a shared library.

## Choosing a graphical data map type

Map type	Use	Type of resource
Message map	Graphical data mapping	.map file
Submap	Reuse of common data transformations	.map file
Local map	Reduced complexity reading and managing a Message map	No file. A local map is embedded within a Message map
IBM WebSphere Message Broker message map*	Solutions that are migrated from earlier versions of IBM Integration Bus  Use Conversion tool to move to new technology	.msgmap file

**\*Note:** You can use an IBM WebSphere Message Broker message map in IBM Integration Bus V10, but you cannot modify it. Support for message maps is maintained for compatibility with earlier versions of IBM Integration Bus.

© Copyright IBM Corporation 2015

Figure 9-29. Choosing a graphical data map type

WM666 / ZM6661.0

### Notes:

This figure summarizes the map types that a graphical data map supports.



## Troubleshooting maps

- A mapping node reports the running of graphical data map scripts as detailed user trace events
  - Reports the entry and completion of each transform in a map
  - Reports the setting of values in the map output

© Copyright IBM Corporation 2015

Figure 9-30. Troubleshooting maps

WM666 / ZM6661.0

### Notes:

The Mapping node reports the running of graphical data map scripts as detailed user trace events. The user trace events report the entry and completion of each transform in a map, and the setting of values in the map output. You can use the IBM Integration Bus user trace to troubleshoot transformation logic that you build in your graphical data maps.

## Unit summary

Having completed this unit, you should be able to:

- Use the Graphical Data Mapping editor to map logical messages
- Run message maps within message flows

© Copyright IBM Corporation 2015

Figure 9-31. Unit summary

WM666 / ZM6661.0

### Notes:



## Checkpoint questions

1. True or false: The Graphical Mapping editor can automatically create transforms when the names of inputs and outputs are similar.
2. True or false: Maps concentrate on sequencing steps, not on structural transformation of the message.

© Copyright IBM Corporation 2015

Figure 9-32. Checkpoint questions

WM666 / ZM6661.0

### Notes:

Write your answers here:

- 1.
- 2.



## Checkpoint answers

1. True or false: The Graphical Data Mapping editor can automatically create transforms when the names of inputs and outputs are similar.

**Answer: True.**

2. True or false: Maps concentrate on sequencing steps, not on structural transformation of the message.

**Answer: False.** Maps concentrate on the structural transformation of the message.

© Copyright IBM Corporation 2015

Figure 9-33. Checkpoint answers

WM666 / ZM6661.0

### Notes:



# Unit 10. Referencing a database in a message flow application

## What this unit is about

In this unit, you learn about the database functions and nodes in IBM Integration Bus. You also learn about defining a database service for database assets.

## What you should be able to do

After completing this unit, you should be able to:

- Use database message processing nodes to modify messages and control message processing
- Configure database nodes to access user databases
- Describe the differences between ESQL and SQL SELECT
- Create a database definition file

## How you will check your progress

- Checkpoint questions
- Lab exercises

## References

IBM Knowledge Center

## Unit objectives

After completing this unit, you should be able to:

- Use database message processing nodes to modify messages and control message processing
- Configure database nodes to access user databases
- Describe the differences between ESQL and SQL SELECT
- Create a database definition file

© Copyright IBM Corporation 2015

Figure 10-1. Unit objectives

WM666 / ZM6661.0

### Notes:

## Using databases in message flows

- Connection to application databases on DB2, MS SQLServer, Oracle, Sybase, Informix, or SolidDB
- IBM Integration Bus ODBC automatically manages transactions
  - ESQL, maps, and JavaCompute (MbSQLStatement)
  - Integration node managed JDBC type 4: JavaCompute, DatabaseRoute, and DatabaseRetrieve nodes
- Transactions must be handled manually in code
  - Standard JDBC API type 2 and 4: JavaCompute
  - SQLJ: JavaCompute node
- Use mapping or compute nodes to SELECT, INSERT, UPDATE, or DELETE
- Database definitions can be imported or discovered
  - Created by the Relational Database Definition wizard in IBM Integration Toolkit
  - Used for maps and ESQL content assist

© Copyright IBM Corporation 2015

Figure 10-2. Using databases in message flows

WM666 / ZM6661.0

### Notes:

You can create and configure message flows to access databases from a message flow. A message flow can also modify the contents of a database by adding new information or removing or replacing existing information.

The integration node automatically handles connection pooling and manages the message flow as a single transaction. When a message flow is idle for approximately 1 minute, or if the message flow completes, the integration node closes the connection.

Accessing a user database in a map requires the database definitions in the workspace. The IBM Integration Toolkit can connect to the database, discover the definition, and then store it in the workspace. If you do not have the correct permissions, have your database administrator export the database definition files.

When using the ESQL Editor with database definitions available in the project (or a referenced project), Content Assist can be used to construct schema and column references.

## Configure database access for the integration node

- On Windows, configure an existing database as a System ODBC data source
- On Linux and UNIX, customize `odbc.ini` file
- Integration node managed JDBC Type 4
  - Create a JDBCProvider configurable service for each (existing) database that is connected to Java applications

```
mqsicreateconfigurableservice IBNODE -c JDBCProviders -o DSN1
  -n databaseName -v SAMPLE
mqsicangeproperties IBNODE -c JDBCProviders -o DSN1
  -n databaseType -v DB2
mqsicangeproperties IBNODE -c JDBCProviders -o DSN1
  -n serverName -v localhost
  ....
mqsireportproperties IBNODE -c JDBCProviders -o DSN1 -r
mqsistop IBNODE
mqsistart IBNODE
```

Configuring local Windows  
DB2 database SAMPLE as  
JDBCProvider DSN1

© Copyright IBM Corporation 2015

Figure 10-3. Configure database access for the integration node

WM666 / ZM6661.0

### Notes:

IBM Integration Bus provides different connection types to application databases, depending on the nodes. Use ODBC and integration node managed JDBC type 4 connections when possible to save on manual programming.

When you include a Mapping, DatabaseRetrieve, DatabaseRoute, JavaCompute, or Java user-defined node in a message flow, and interact with a database in that node, the integration node must establish connections with the database. Define a JDBCProvider configurable service to provide the integration node with the information that it requires to complete the connection.

The example on the figure shows how to configure a local Windows DB2 database SAMPLE as JDBCProvider DSN1. The properties are created with the `mqsicreateconfigurableservice` command modified with the `mqsicangeproperties` command. The `mqsireportproperties` command is used to examine the values of the properties.

It might be necessary to change more properties specific to your database and environment. Some values depend on how and where you installed the database product; for example, the property `jarsURL` identifies the location of the JAR files that the database provider supplies and provides.

## Verifying ODBC connections

- Use the `mqsicvp` command to run verification tests on an integration node, or to verify ODBC connections
- On Linux and UNIX systems only, verify that the ODBC environment is configured correctly:

```
mqsicvp IntNodeName
```

- To display useful information about a user data source:

```
mqsicvp -n Datasource -u DatasourceUserId -p DatasourcePassword
```

- To compare two user data sources:

```
mqsicvp -n Datasource -u primaryDatasourceUserId  
-p primaryDatasourcePassword -c secondaryDatasource  
-i secondaryDatasourceUserId -a secondaryDatasourcePassword
```

**Note:** The user ID and password parameters are required if you did not previously associate the data source name with the integration node

© Copyright IBM Corporation 2015

Figure 10-4. Verifying ODBC connections

WM666 / ZM6661.0

### Notes:

When you use the `mqsicvp` command as an ODBC test tool, the command sends an informational message for a successful connection, providing the name of the data source, database type, and version. If a secondary data source is supplied, the `mqsicvp` command sends a second information message for a successful connection to that data source, with the same information about the secondary data source, and stating that a comparison is made.

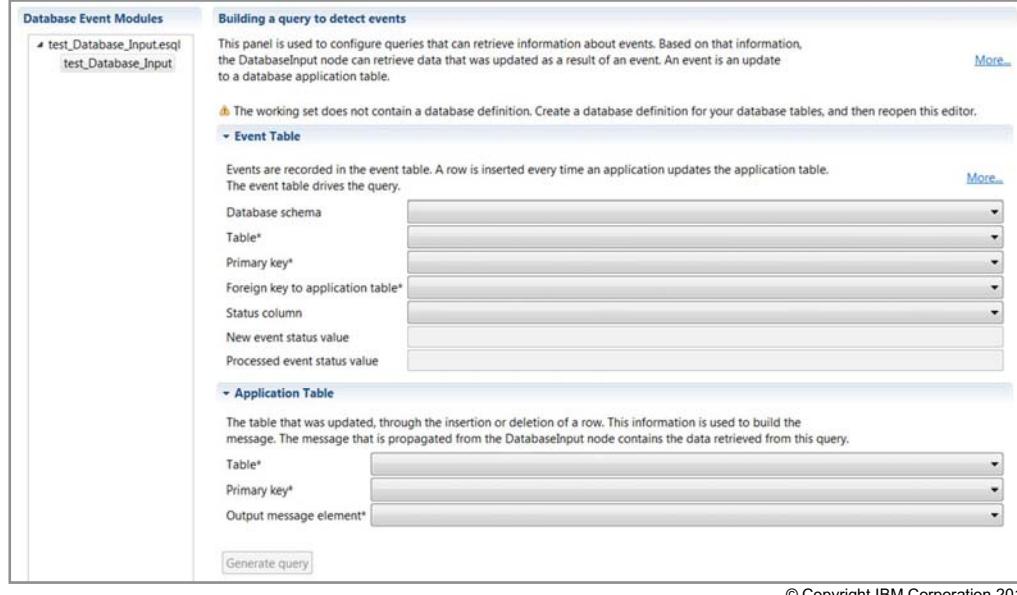
When the tool is run against one data source, it completes several checks against the ODBC interface. These checks determine which data types and functions are supported, with information about the names and sizes of those data types. If any data types or functions are not supported, they are summarized in a final informational message.

 WebSphere Education 

## DatabaselInput node



- Use the DatabaselInput node to detect events that are recorded in a database, and to retrieve the data that is affected by those events
- Use Database Event Design tab to generate the ESQL module with the query



© Copyright IBM Corporation 2015

Figure 10-5. DatabaseInput node

WM666 / ZM6661.0

### Notes:

You can use the DatabaseInput node to create message flows that respond to events in a database.

When you drag a DatabaseInput node onto the Message Flow editor canvas, the Integration Toolkit creates an ESQL module that contains template text. To configure the DatabaseInput node, modify the statements in that module to fit your requirements.

When you double-click the node to modify the ESQL code, the editor displays the **Database Event Design** tab for the module. Complete the mandatory fields and then click **Generate query**.

## Database node



- Allows a transaction in the form of an ESQL expression
- An input message is sent unchanged to the output terminal
- UPDATE, INSERT, DELETE, and SELECT
- SELECT cannot directly update a message

Example:

```

INSERT INTO Database.DSN2.Schema2.PRICEDATA
  (PRODUCT_NAME, ITEM_PRICE, STATUS)
VALUES (Body.Message.ProductID,
        Body.Message.UnitPrice, 'OPEN') ;
  
```

© Copyright IBM Corporation 2015

Figure 10-6. Database node

WM666 / ZM6661.0

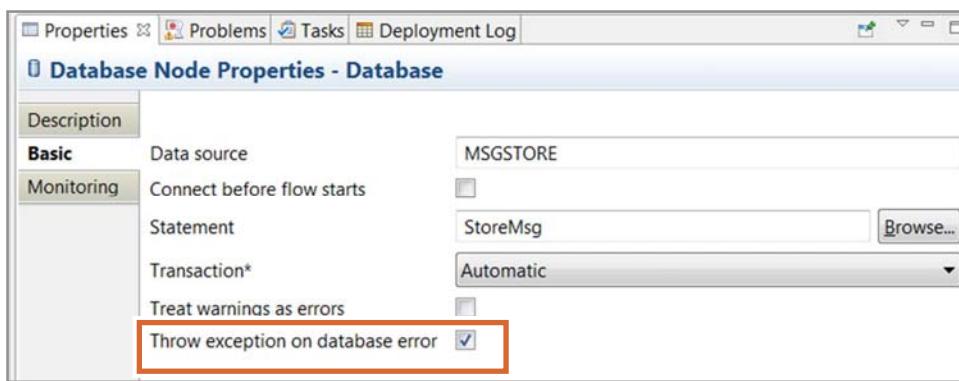
### Notes:

The Compute node can be used to extract and update databases, and create an output message. In a Database node, you can do all database operations without needing to build a new output message.



## Node properties for database access

- Set the **Data source** to use as a default DSN (ODBC)
- When **Transaction = Automatic**, database updates are either committed or rolled back with the entire message flow
- When **Transaction = Commit**:
  - Updates to the default DSN are immediate, even if the message flow fails later
  - All noncommitted updates are made in this message flow to the default DSN
- To handle SQL errors in ESQL code by using SQLNATIVEERROR functions, clear the **Throw exception on database error** check box



© Copyright IBM Corporation 2015

Figure 10-7. Node properties for database access

WM666 / ZM6661.0

### Notes:

Every node that accesses databases must identify the default data source name (DSN) in the node properties.

If there is a database mapping in the message flow, all ESQL nodes in the flow must specify a data source (it can be any data source).

The integration node can coordinate the transaction and commit the database update after completion of the flow (**Transaction = Automatic**), or commit all updates on this database, even from upstream nodes, immediately (**Transaction = Commit**).

In any database node, you can set the properties **Treat warnings as errors** (default = NO), and **Throw exception on database error** (default = YES).

## Transaction support

- By default, a message flow is a logical unit of work
  - Commit or rollback resources
  - Transaction property of input node; the default is YES
- Some processing nodes can be explicitly excluded from a logical unit of work
  - It is a partially coordinated transaction, if **TransactionMode** of node = **Commit**
- If multiple resource managers in a flow (IBM MQ and DBMS), choose the transaction coordinator for a deployed message flow in BAR

© Copyright IBM Corporation 2015

Figure 10-8. Transaction support

WM666 / ZM6661.0

### Notes:

IBM MQ can act as a transaction coordinator.

- It allows updates with external resource manager to be atomic.
- It is provided through an XA connection between IBM MQ and the resource manager

For example, with XA the application issues one MQCMIT call to commit both IBM MQ and DB2 updates. Without XA, the application must send an MQCMIT call to commit IBM MQ processing and an EXEC SQL SYNCPOINT to commit DB2 updates.

There is a potential for failure between commits, which can compromise data integrity.

For maximum performance, ensure that both the queue manager and the resource manager logs are on low latency disks.

An XA-coordinated transaction is enabled in the BAR file.

## Transaction coordinator options

- Uncoordinated transaction (default)
  - Integration node handles the transaction
  - A database commit, then MQCMIT
  - It is a problem if MQCMIT fails after a successful database commit
- XA-coordinated transaction
  - Integration node queue manager acts as an XA transaction manager
  - Two-phase commit includes database and IBM MQ resources (MQBEGIN)
  - Special setup is necessary
- To enable XA:
  1. Set up XA for the integration node queue manager
  2. In the BAR file, display the message flow properties and select **Coordinated transaction**
  3. Change the transaction to **Automatic** on the database nodes
  4. Enable XA in the application database, if necessary

© Copyright IBM Corporation 2015

Figure 10-9. Transaction coordinator options

WM666 / ZM6661.0

### Notes:

Uncoordinated flows are flows for which the **Coordinated** property is not set in the BAR file. Separate resource managers manage updates to resources used by an uncoordinated flow.

Some resource managers, such as IBM MQ, allow updates to be made nontransactionally, or as part of a resource-specific transaction. Other resource managers, such as database managers, always use a resource-specific transaction. A resource-specific transaction is a transaction whose scope is limited to the resources owned by a single resource manager, such as a database or queue manager.

To reduce the window of doubt when multiple separate resource managers are synchronized, transaction coordination by using the XA protocol is provided on distributed systems by IBM MQ and on z/OS systems by RRS. Message flows are always globally coordinated on z/OS, regardless of whether the **Coordinated** property of the message flow is specified as coordinated or not.

## ESQL SELECT compared with SQL SELECT

- ESQL limitations on SQL SELECT
- SELECT DISTINCT
  - SELECT ALL
  - GROUP BY or HAVING
  - AVG is a column function
- Use PASSTHRU to bypass the IBM Integration Bus parser
  - For special SELECTs
  - For special database syntax
  - Call stored procedures
  - Run a coded statement
- ESQL additions to SQL SELECT
  - Produces a tree-structured result
  - Accepts arrays in SELECT clauses
  - THE(SELECT)
  - SELECT ITEM

© Copyright IBM Corporation 2015

Figure 10-10. ESQL SELECT compared with SQL SELECT

WM666 / ZM6661.0

### Notes:

The ESQL SELECT statement is similar to the SQL SELECT statement, with limitations as listed in the figure.

ESQL SELECT differs from database SQL SELECT in the following ways:

- ESQL can produce tree-structured result data
- ESQL can accept arrays in SELECT clauses
- ESQL has the THE function and the ITEM and INSERT parameters
- ESQL has no SELECT ALL function
- ESQL has no ORDER BY function
- ESQL has no SELECT DISTINCT function
- ESQL has no GROUP BY or HAVING parameters
- ESQL has no AVG column function in this release

You can use the PASSTHRU function is to send complex SELECTs, not currently supported by IBM Integration Bus, to databases.

## PASSTHRU example

- PASSTHRU function issues complex SELECTS

Example:

```
SET OutputRoot.XMLNSC.Msg.Results[ ]= PASSTHRU(  

  'SELECT WORKDEPT, MAX(SALARY) AS MaxSalary  

   FROM schema1.EMPLOYEE  

   GROUP BY WORKDEPT  

   HAVING MAX(SALARY) > ?'  

  TO Database.SAMPLE  

  VALUES (InputBody.Msg.Salary));
```

- PASSTHRU statement issues administrative commands to a database
- Use parameter markers for performance

© Copyright IBM Corporation 2015

Figure 10-11. PASSTHRU example

WM666 / ZM6661.0

### Notes:

If you must use the full SQL SELECT function, use a SQL SELECT GROUP BY as a text string in an ESQL PASSTHRU statement. The database must be defined as an input or output data source in the typical way.

The question mark character (?) is processed as a substitution variable in a prepared SQL statement by the DBMS. The ESQL PASSTHRU statement allows a message flow to run a dynamic SQL statement. Because these statements are expensive to prepare, aim to achieve maximum reuse; use substitution variables (?), not literal values, in the PASSTHRU statement. The values in the variables are substituted just before the call is made.

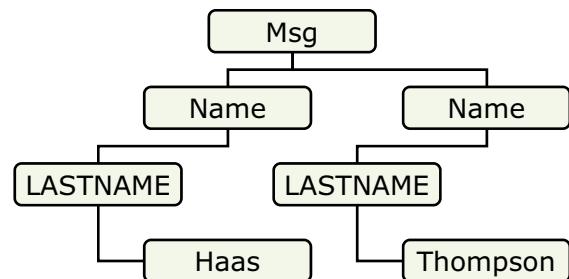
In the example in the figure, this method allows the statement to be reused for multiple MAX(SALARY) values.

PASSTHRU has no performance gains when compared to using the same SQL statement from ESQL. So, use PASSTHRU only if the SQL statement is not valid in ESQL; for example, to provide database-specific parameters.

## THE (SELECT ...)

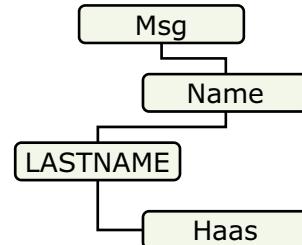
- SELECT returns a list

```
SET OutputRoot.XML.Msg.Name []=
(SELECT E.LASTNAME
from Database.EMPLOYEE as E);
```



- THE (SELECT ...) returns the first element of a list

```
SET OutputRoot.XML.Msg.Name =
THE (SELECT E.LASTNAME
from Database.EMPLOYEE as E);
```



© Copyright IBM Corporation 2015

Figure 10-12. THE (SELECT ...)

WM666 / ZM6661.0

### Notes:

This figure shows the difference between a SELECT statement and a THE(SELECT) statement.

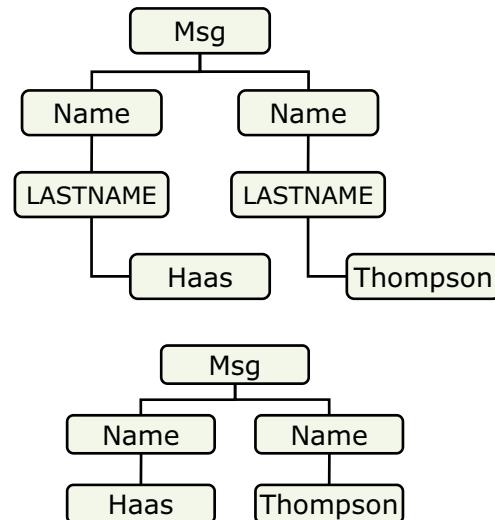
SELECT returns a list. THE (SELECT) returns a non-list. This function is necessary because ESQL accepts only both lists or both non-lists on either side of the equation.

## SELECT (ITEM ...)

- SELECT (ITEM...) gets the value only
  - Without the ITEM keyword, the database table column name or XML tag is used as a message tag
  - Can be overridden with an AS keyword in a SELECT statement

```
SET
OutputRoot.XML.Msg.Name [ ] =
(SELECT E.LASTNAME from
Database.EMPLOYEE as E);
```

```
SET
OutputRoot.XML.Msg.Name [ ] =
(SELECT ITEM E.LASTNAME from
Database.EMPLOYEE as E);
```



© Copyright IBM Corporation 2015

Figure 10-13. SELECT (ITEM ...)

WM666 / ZM6661.0

### Notes:

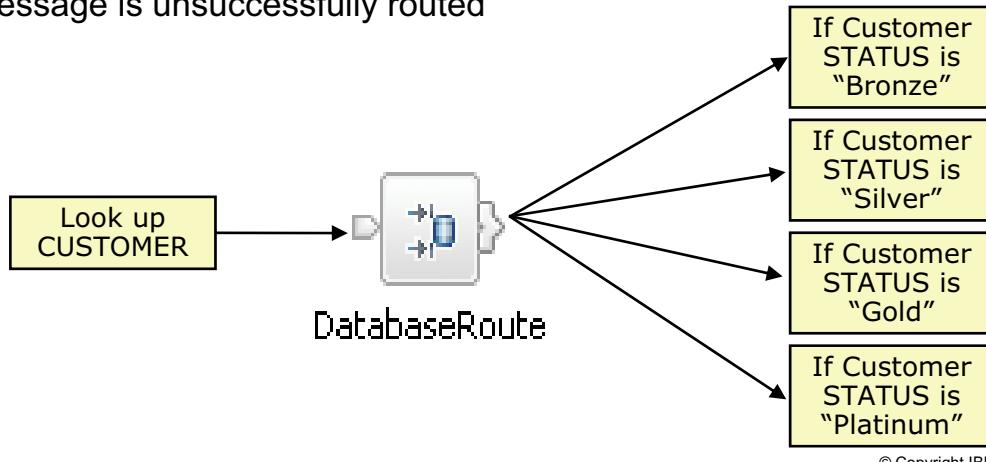
The SELECT clause can consist of the keyword ITEM and a single expression. The effect of SELECT(ITEM) is to make the results nameless. That is, the result is a list of values of the type that the expression returns, rather than a row.

The SELECT(ITEM) option has several uses:

- With a scalar expression and the THE function, it can be used to create a SELECT query that returns a single scalar value (for example, the price of a particular item from a table).
- With a CASE expression and ROW constructors, it can be used to create a SELECT query that creates or handles messages in which the structure of some rows is different from others. This use can handle messages that have a repeating structure but in which the repeats do not all have the same structure.
- With a ROW constructor, it can be used to create a SELECT query that collapses levels of repetition in the input message.

## DatabaseRoute node

- Directs messages that meet certain criteria down different parts of a flow
  - Matches on contents of a database (JDBC connection)
  - Uses XPath or ESQL(\*) expressions to identify match criteria
- Define the custom output terminals to specify multiple expressions
- **Default** terminal if no matches are found
- **KeyNotFound** terminal if the query is unsuccessful or the outgoing message is unsuccessfully routed



© Copyright IBM Corporation 2015

Figure 10-14. DatabaseRoute node

WM666 / ZM6661.0

### Notes:

The DatabaseRoute node provides a graphical technique for enabling incoming requests to be examined, and sent down the appropriate part of the message flow, depending on the specified criteria. Based on this criteria, the message can be routed to output terminals on the DatabaseRoute node, which the application designer can specify. The routing decision is based on the contents of a relational database.

In the example in the figure, the incoming message is a customer record that contains a customer name. This request must be routed down a certain path of the flow, depending on the STATUS of the customer. In this example, the customer status is stored in a relational database, which the DatabaseRoute node must read.

When accessing a database, the requested record might not be found. In this case, or if the message fails to be updated correctly, control is passed to the **keyNotFound** terminal. If a more general error occurs, control is passed to the **Failure** terminal, as normal.

## Configuring the DatabaseRoute node

1. The workspace must have:
  - A database definition
  - A message model
2. Add custom output terminals to the DatabaseRoute node
3. Add query elements
4. Add filter expressions
5. Set the **Distribution mode** to propagate messages to the FIRST terminal that matches, or to ALL that match
6. Wire the node terminals
7. Configure JDBC for the integration node
8. Deploy

© Copyright IBM Corporation 2015

Figure 10-15. Configuring the DatabaseRoute node

WM666 / ZM6661.0

### Notes:

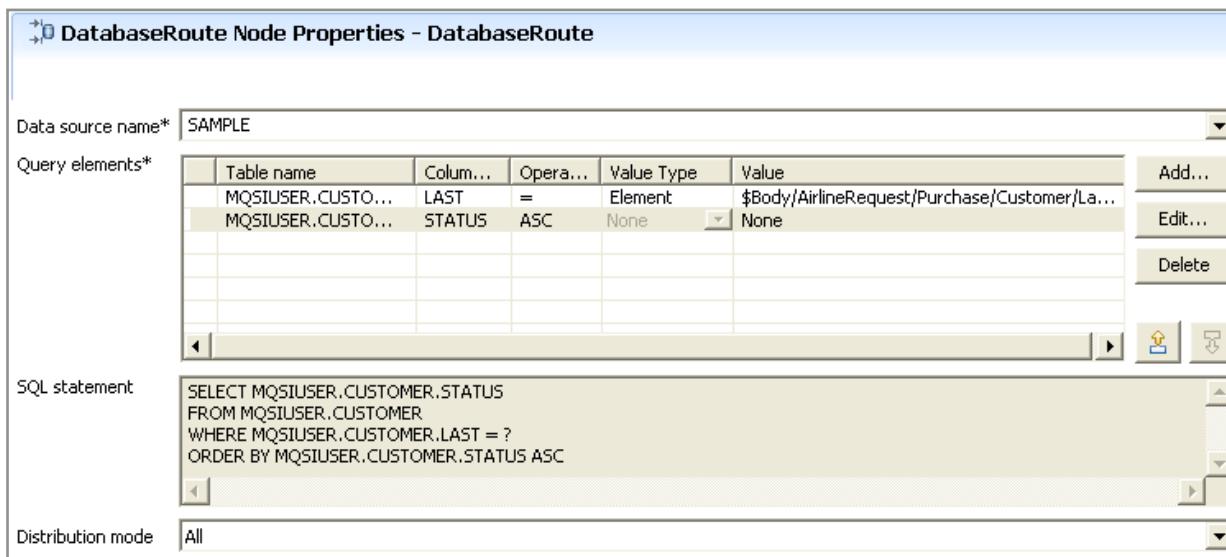
This figure lists the steps for configuring the DatabaseRoute node.

Both the database definition and a message model must be available in your workspace to successfully configure the DatabaseRoute node.

The profile of your message flow determines the number of output terminals. You can add more output terminals to the DatabaseRoute node. Right-click the DatabaseRoute node, click **Add Output Terminal**, and define as many named output terminals as needed.

An example of the DatabaseRoute node is shown on the next figure.

## DatabaseRoute node query elements



© Copyright IBM Corporation 2015

Figure 10-16. DatabaseRoute node query elements

WM666 / ZM6661.0

### Notes:

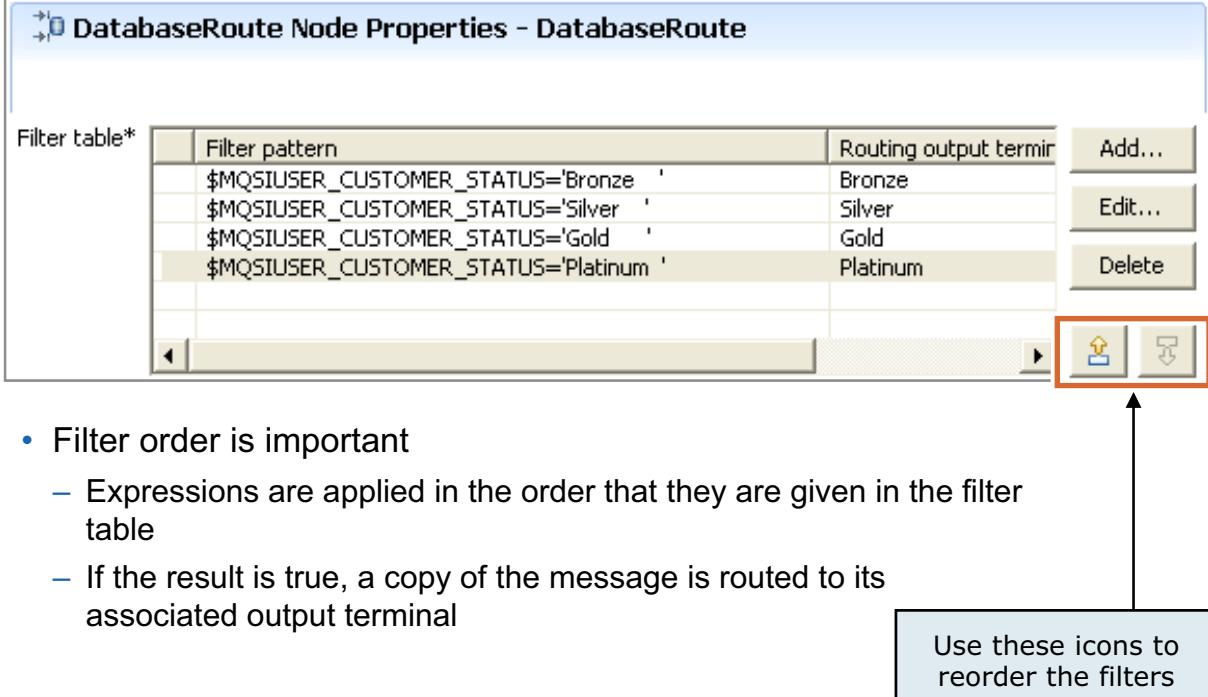
The figure shows some example DatabaseRoute node query elements:

- **Table name** identifies the table. In the example, MQSIUSER is the database schema, and CUSTOMER is the database table name.
- **Column** identifies the database column name. In this example, LAST is the column of the database table that the database route node reads.
- **Operator** is a Boolean value that specifies how to retrieve the required row. The Operator in this example is “=”, which is requesting a direct match on the element value.
- **Value** contains the XPath expression that identifies the data in the incoming message. This data is used to read the required database row. This value can be typed in manually, or automatically included by clicking the icon to the right of this field and following the information that is displayed in the dialog box.

The generated SQL statement is shown at the bottom of these properties. To ensure that the query statement retrieves at least one valid row, there must be a query element with a null (None) value type. If you do not include a query element with a null value type, the SQL statement is incorrect.

WebSphere Education 

## DatabaseRoute node filter expressions



The screenshot shows the 'DatabaseRoute Node Properties - DatabaseRoute' dialog box. It features a 'Filter table\*' section with four rows of data:

	Filter pattern	Routing output terminal
	\$MQSIUSER_CUSTOMER_STATUS='Bronze'	Bronze
	\$MQSIUSER_CUSTOMER_STATUS='Silver'	Silver
	\$MQSIUSER_CUSTOMER_STATUS='Gold'	Gold
	\$MQSIUSER_CUSTOMER_STATUS='Platinum'	Platinum

On the right side of the table are three buttons: 'Add...', 'Edit...', and 'Delete...'. Below the table are four small icons: a left arrow, a right arrow, an upward arrow, and a downward arrow. The upward and downward arrows are highlighted with a red box and connected by a vertical arrow pointing upwards to a callout box.

Use these icons to  
reorder the filters

- Filter order is important
  - Expressions are applied in the order that they are given in the filter table
  - If the result is true, a copy of the message is routed to its associated output terminal

© Copyright IBM Corporation 2015

Figure 10-17. DatabaseRoute node filter expressions

WM666 / ZM6661.0

### Notes:

The next step for configuring the DatabaseRoute node is to specify how to process the filter patterns statements. The default value for **Distribution Mode** is **All**, which means that all filter patterns are checked. Every filter condition that is true causes a message to be passed to the specified **Routing output terminal**. In this case, multiple paths in the message flow are run.

If the **Distribution Mode** is changed to **First**, each filter pattern is evaluated in turn, starting from the top of the list. When the first matching filter is encountered, control is passed to the specified terminal.

To improve performance, specify the expressions that are satisfied most frequently at the top of the filter table. The order in which filter patterns are evaluated can be changed by using the UP and DOWN icons.

The figure shows an example of filter patterns that filter based on the customer status.

## DatabaseRetrieve node



- Populates fields by using data in a database record
- Similar to the Compute node with database SELECT
- Output terminals
  - **Out:** Outgoing message is routed here when it is modified successfully
  - **KeyNotFound:** Terminal to which the original message is routed, unchanged, when the result set is empty
  - **Failure:** Terminal to which the message is routed if a failure is detected during processing
- Database Retrieve and Route nodes provide similar functions to the DatabaseRoute node

© Copyright IBM Corporation 2015

Figure 10-18. DatabaseRetrieve node

WM666 / ZM6661.0

### Notes:

The DatabaseRetrieve node looks up values from a database and stores them as elements in the outgoing message assembly trees. The lookup process returns column values, which are acquired and passed back in the result set from SQL queries. The values are converted first into a matching Java type and then into an internal message element value type when it is stored in an outgoing message assembly tree. If a message element exists in the outgoing message tree, the new value overwrites the old value. If the target element does not exist, it is created, and the value is stored.

The node needs query information that is used to form an SQL select query, which can access multiple tables in a database by using multiple test conditions. Sometimes, not all the information that you want to retrieve in a result set is in a single database table. To get the column values that you want, it might be necessary to retrieve them from two or more tables. This node supports the use of SELECT statements that facilitate getting columns from one or more tables in a single result set.

The application of the expression must return a single element, double element, Boolean, or string. If the query returns multiple rows, the first row is chosen and the rest are ignored, unless the **Multiple rows** option is selected. In this case, all rows are processed, and values in those rows are used to update the outgoing message assembly trees.



## Using databases in a Mapping node

- IBM Integration Toolkit requires a database definition (.dbm) file, contained in a data design project to create database mappings by using the Mapping node
  - Database definition file holds the physical data model that details all the database resources, such as the schema, the tables, and other resources
  - Can also use database definitions in other nodes, such as the Compute node, to validate references to database sources and tables
- If you can connect to the database from the IBM Integration Toolkit, use the New Database Definition File wizard to add database definitions

© Copyright IBM Corporation 2015

Figure 10-19. Using databases in a Mapping node

WM666 / ZM6661.0

### Notes:

You can use database transforms in a Mapping node to insert new rows of data, or to update or delete existing rows of data, in your database tables.

For each database transform in your graphical data map, the Graphical Data Mapping editor uses a database definition (.dbm) file to determine the name and structure of the database. You can use the **New Database Definition File** wizard to create a database definition file when you create a database transform in a graphical data map.

When you add a database Insert, Update or Delete transform to a graphical data map, the transform is displayed as an output target to which you can connect input objects.



## Using New Database Definition File wizard (1 of 2)

**1.** Click **File > New > Database Definition**

**2.** Define the **Data design project**, select the database and version

**3.** Define the database connection by providing the database name, user name, and password

© Copyright IBM Corporation 2015

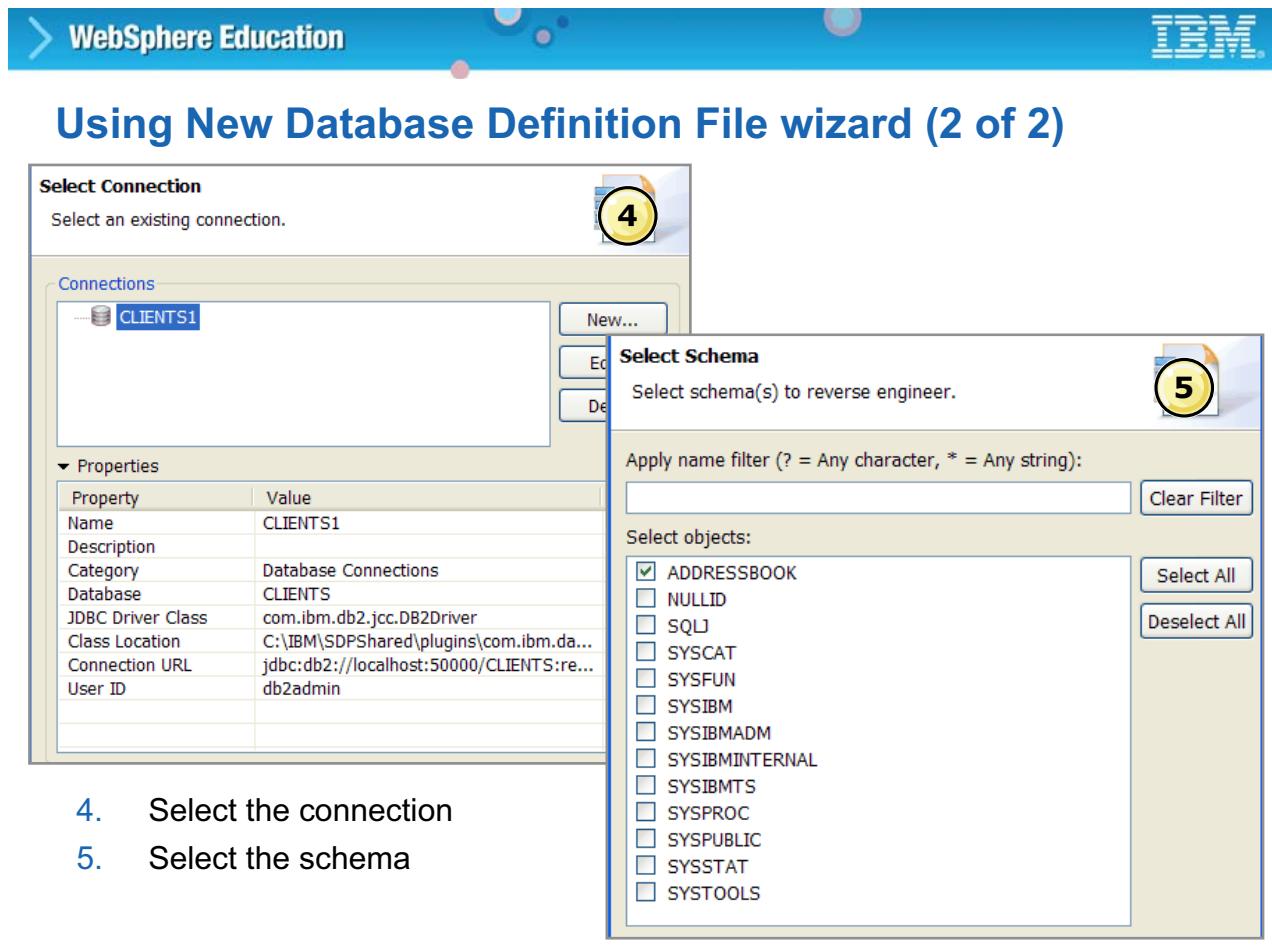
Figure 10-20. Using New Database Definition File wizard (1 of 2)

WM666 / ZM6661.0

### Notes:

To create database mappings by using the Mapping node, you must have a database definition file (.dbm) that is contained in a data design project.

This figure and the next summarize the steps for creating a database definition file.



© Copyright IBM Corporation 2015

Figure 10-21. Using New Database Definition File wizard (2 of 2)

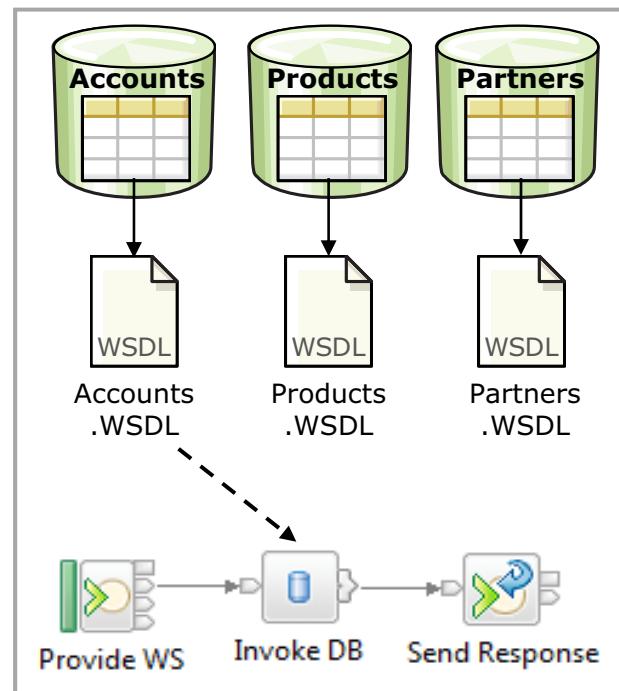
WM666 / ZM6661.0

**Notes:**

The final steps for creating a database definition file are to select the connection and then select the schema.

## Database service discovery

- DBMS represents a system of record for key business entities such as customers, accounts, partners, and products
- IBM Integration Bus tools discover and represent these key data
- Integration services extend access to user applications
- Many uses for a database service definition
  - WSDL contains both logical and physical database information
  - Drag WSDL to automatically create SQL access methods
  - Create an integration service to use customized database access



© Copyright IBM Corporation 2015

Figure 10-22. Database service discovery

WM666 / ZM6661.0

### Notes:

You can define a database service to make database operations accessible in a message flow and to external applications that call a message flow.

A database service contains:

- Operation-oriented interface definitions about the database interaction. For example, an operation can reflect the changes to make to a particular table. This information is contained in a WSDL file.
- A database connection binding that captures the semantics of the operations on the database. This information is contained in a WSDL file.
- XSD files that describe the inputs and outputs of the database operations.
- A .service file that captures metadata that is used during the discovery process. The IBM Integration Toolkit uses this file to store state information from the Database Service editor, and do iterative discovery of previously discovered artifacts.

Database service discovery is taught in detail in IBM course WM676, *IBM Integration Bus V10 Application Development II*.



## Unit summary

Having completed this unit, you should be able to:

- Use database message processing nodes to modify messages and control message processing
- Configure database nodes to access user databases
- Describe the differences between ESQL and SQL SELECT
- Create a database definition file

© Copyright IBM Corporation 2015

Figure 10-23. Unit summary

WM666 / ZM6661.0

### Notes:



## Checkpoint questions

1. True or False: Database updates are only rolled back in case of an error, if you configured flow, DBMS, and queue manager for a coordinated transaction.

© Copyright IBM Corporation 2015

Figure 10-24. Checkpoint questions

WM666 / ZM6661.0

### Notes:

Write your answers here:

1.



## Checkpoint answers

1. True or False: Database updates are only rolled back in case of an error, if you configured flow, DBMS and queue manager for a coordinated transaction.

**Answer: False.** The default is a transactional flow with a one-phase commit, that is, an uncoordinated (integration node coordinated) transaction.

© Copyright IBM Corporation 2015

Figure 10-25. Checkpoint answers

WM666 / ZM6661.0

### Notes:

## Exercise 9



Referencing a database in a map

© Copyright IBM Corporation 2015

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

10.1

Figure 10-26. Exercise 9

WM666 / ZM6661.0

### Notes:



## Exercise objectives

After completing this exercise, you should be able to:

- Create a shared library that contains data models that describe the input and output data
- Import a COBOL Copybook to create a DFDL schema file
- Reference a shared library in a message flow application
- Discover database definitions
- Add a Database node to a message flow
- Create the logic to store a message in a database
- Use the Graphical Data Mapping editor to map message elements
- Reference an external database when mapping message elements

© Copyright IBM Corporation 2015

Figure 10-27. Exercise objectives

WM666 / ZM6661.0

### Notes:

See the *Student Exercises Guide* for detailed instructions.

# Unit 11. Using Compute nodes to transform messages

## What this unit is about

This unit introduces programming options that are available in IBM Integration Bus for transforming and enriching data. It concentrates on the use of Extended Structured Query Language (ESQL) and Java.

## What you should be able to do

After completing this unit, you should be able to:

- Use the Compute node and ESQL to transform messages
- Use the JavaCompute node and Java to transform messages

## How you will check your progress

- Checkpoint questions
- Lab exercise

## References

IBM Knowledge Center

Appendix A: ESQL examples



## Unit objectives

After completing this unit, you should be able to:

- Use the Compute node and ESQL to transform messages
- Use the JavaCompute node and Java to transform messages

© Copyright IBM Corporation 2015

Figure 11-1. Unit objectives

WM666 / ZM6661.0

### Notes:

## Transformation options in IBM Integration Bus

- C#, F#, Visual Basic, C++/CLI (.NETCompute node)
- Graphical Data Mapping (Mapping node)
- XSLT (XSLTransform node)
- ESQL (Compute node)
- Java (JavaCompute node)
- IBM WebSphere Transformation Extender add-on
- IBM WebSphere adapter nodes
- Database stored procedures
- C or Java plug-in nodes
- A combination of the preceding options
  
- Data, model, and parser affects the transformation technology choice
  - ESQL and Java provide the greatest coverage of message models
  - Other technologies work on some, but not all of the models

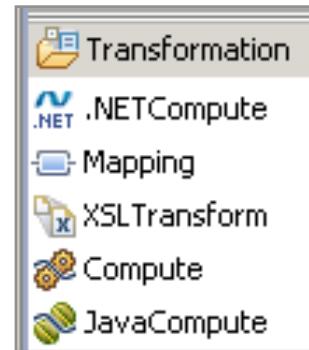


Figure 11-2. Transformation options in IBM Integration Bus

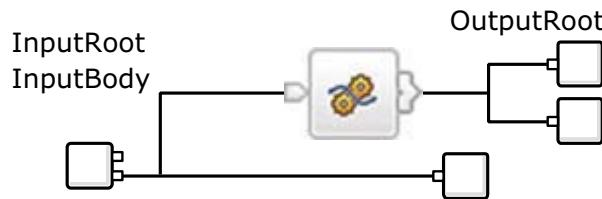
WM666 / ZM6661.0

### Notes:

You can use many types of message processing nodes to modify the contents of the message tree. The compute nodes (Compute, JavaCompute, and .NETCompute), the XSLTransform node, and the Mapping node are most often used for modifying the message tree. Database stored procedures, WebSphere Transformation Extender, and WebSphere adapter nodes are used to update the message tree with technologies external to IBM Integration Bus.

The compute nodes require you to write code to transform messages. Each of the nodes can do the same types of message transformation and enrichment but they each use a different programming language.

## Compute node



- Data is copied from the input to the output area
  - Entire input or just message headers
  - Other parts of the message assembly
- Create a message or messages by coding ESQL in the message flow ESQL resource file
- Supports manipulation of output message structure such as data retrieval from an external database, and reordering fields
- ESQL PROPAGATE statement can be used to:
  - Generate multiple output messages
  - Route a message to different output terminals: Out, Out1, Out2, Out3, and Out4
- **Compute Mode** property specifies which message trees to propagate

© Copyright IBM Corporation 2015

Figure 11-3. Compute node

WM666 / ZM6661.0

### Notes:

The Compute node ESQL code can access and modify the message tree. It can build multiple output messages by using fields from a single input message, calculated fields, and fields that are retrieved from database tables.

In previous units, you saw that `Root` was the highest level identifier in the logical message tree when you referenced the message tree in a Filter node. In the Compute node, you must prefix `Root` with `Input` or `Output` to differentiate between the incoming message and the outgoing message. However, when using some ESQL statements `OutputBody` is NOT a valid correlation name (like `InputBody`) because the integration node cannot determine the message domain (parser) to use to build the body. So, you must always explicitly identify the parser with `OutputRoot`. For example:

```

SET OutputRoot.XMLNSC.a.Field
SET OutputRoot.DFDL.b.Field
  
```

You can also use the Compute node to build a new `OutputRoot` tree by copying the entire `InputRoot` or just parts of it, such as the `InputBody` or the headers. The ESQL code template includes logic to copy the message from an input tree to an output tree. You can delete this code if your application does not need it.

Only one output message is built at a time. You can build a sequence of different messages by using the ESQL PROPAGATE statement. So it is possible, for example, to build multiple output messages from one incoming message.

The **Compute Mode** property controls the components that are used by default in the output message. You can modify the property to specify whether the Message, LocalEnvironment, and Exception List components that are either generated in the node or contained in the incoming message are used. The value of the **Compute Mode** property is used when the transformed message is routed to the **Out** terminal when processing in the node is completed. It is also used whenever a PROPAGATE statement does not specify the composition of its output message.



## ESQL editor

- Statement syntax validation
  - Missing punctuation, incorrectly used functions and keywords, and more
  - Project can be deployed with warnings, but not errors
- Semantic validation
  - Unresolved identifiers
  - Message or database reference mismatch
  - Deprecated keywords
- Can set validation preferences
- Content assist
  - To start, press **Ctrl+Space** or click **File > Edit**
  - Right-click in editor to access more editor options

© Copyright IBM Corporation 2015

Figure 11-4. ESQL editor

WM666 / ZM6661.0

### Notes:

The ESQL editor provides statement syntax and semantic validation.

The ESQL editor is described in detail in Unit 5, “Controlling the flow of messages”.

## ESQL content assist and validation

- Create a module for a node: flowName\_nodeName
- Associate a node with an existing module

Right-click in the editor view for a menu with more options

Error "decorator"

Category	Description	Resource	In Folder	Location
Warning	Identifier myNumber cannot be resolved.	File1.e...	EsqlEditorDemo/e...	line 8

© Copyright IBM Corporation 2015

Figure 11-5. ESQL content assist and validation

WM666 / ZM6661.0

### Notes:

The figure shows an example of the ESQL for a Compute node. IBM Integration Bus contains ESQL content assist to help you create syntactically valid ESQL statements.

The error decorator indicates that a warning (yellow triangle) or an error (red exclamation point) was detected on the marked line.

Review the **Problems** tab for details about the issues the decorator represents. An ESQL module can be deployed if it contains warnings, but not if it contains errors.

## Compute node ESQL

- ESQL file
  - Contains ESQL for every node in the message flow that requires ESQL
  - Is named *MessageFlowName\_ComputeNodeName.esql* by default
- Compute node is associated with one ESQL module in the message flow ESQL file
  - ESQL module name must match **ESQL Module** node property
- Each module contains a function that is called `Main()` which is the entry point at which the processing of the node starts
- When used in the `Main()` function, RETURN value controls message propagation to Compute node output terminals

`RETURN TRUE;`      Message is propagated to the **Out** terminal

`RETURN FALSE;`      Message is not propagated

`RETURN UNKNOWN;`      Message is not propagated

© Copyright IBM Corporation 2015

Figure 11-6. Compute node ESQL

WM666 / ZM6661.0

### Notes:

ESQL is written in stand-alone modules. Each ESQL resource (file) can contain many modules. Nodes that use ESQL are configured in their properties to call an ESQL module.

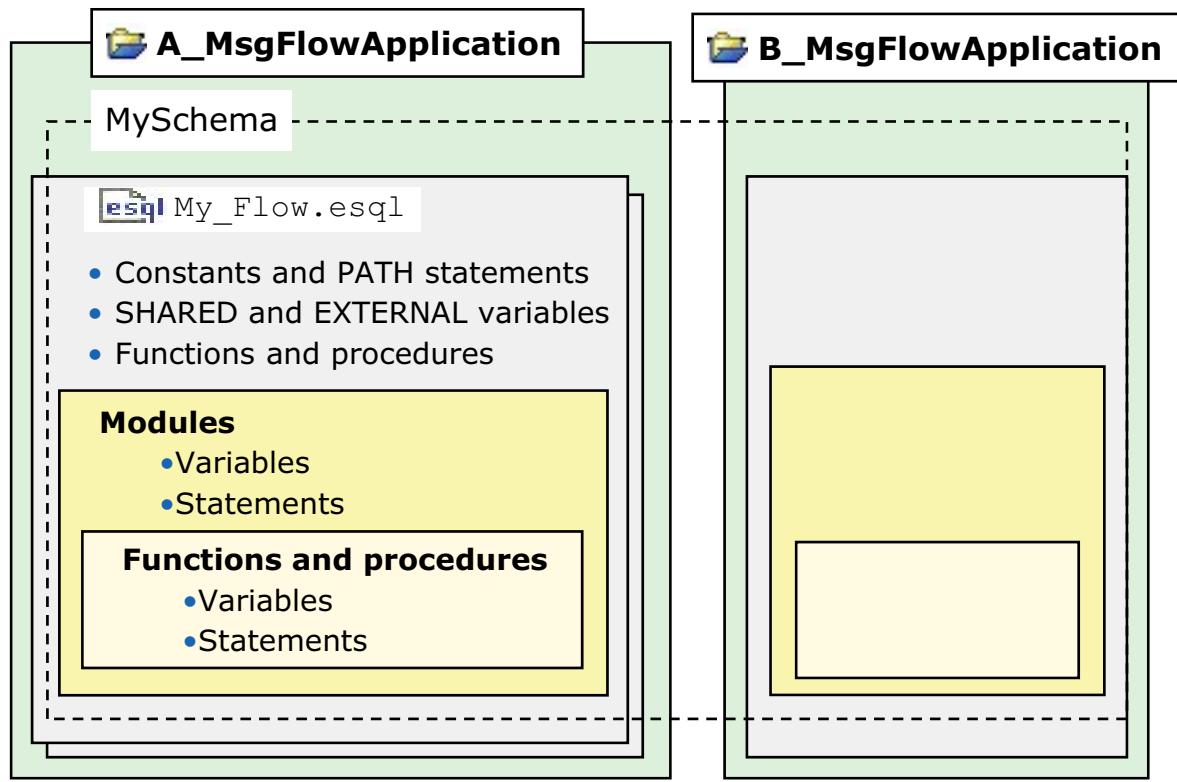
Every ESQL module must have a function `main()` that has no arguments and returns type Boolean. An ESQL code template, is created automatically when a new module is generated from a node.

Be sure that you understand that the Boolean value returned by the MAIN function controls which terminal is used to propagate the output message.

The RETURN statement ends the program; subsequent action occurs depending on where the RETURN statement is used.

The function of the RETURN statement is different when used in a Filter or Database node. RETURN is especially useful when used in Filter nodes because it allows the filter decision to be made by using the full power of ESQL statements, rather than just an expression.

## ESQL module and integration node schema



© Copyright IBM Corporation 2015

Figure 11-7. ESQL module and integration node schema

WM666 / ZM6661.0

### Notes:

A name identifies each ESQL module that is defined in an ESQL file in the same integration node schema as the node that references it. There must be one module of the correct type for each corresponding node.

If you want to reuse ESQL constants, functions, or procedures, you must declare them at the integration node schema level. You can then reference the ESQL objects from any resource within that integration node schema, in either the same or in another project. If you want to use this technique, you must either fully qualify the resource or include a PATH statement that sets the qualifier. The PATH statement must be coded in the same ESQL file, but not within any module.

```
PATH MySchema2;
CALL MySchema2.ProcX(p);
```

## Compute node ESQL module skeleton

```

CREATE COMPUTE MODULE MessageFlowName_ComputeNodeName
  CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
      -- CALL CopyMessageHeaders();
      -- CALL CopyEntireMessage();
      RETURN TRUE;
    END;
  CREATE PROCEDURE CopyMessageHeaders() BEGIN
    DECLARE I INTEGER 1;
    DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
    WHILE I < J DO
      SET OutputRoot.*[I] = InputRoot.*[I];
      SET I = I + 1;
    END WHILE;
  END;
  CREATE PROCEDURE CopyEntireMessage() BEGIN
    SET OutputRoot = InputRoot;
  END;
END MODULE;

```

Uncomment CALL statement to either copy message headers only or copy entire message

© Copyright IBM Corporation 2015

Figure 11-8. Compute node ESQL module skeleton

WM666 / ZM6661.0

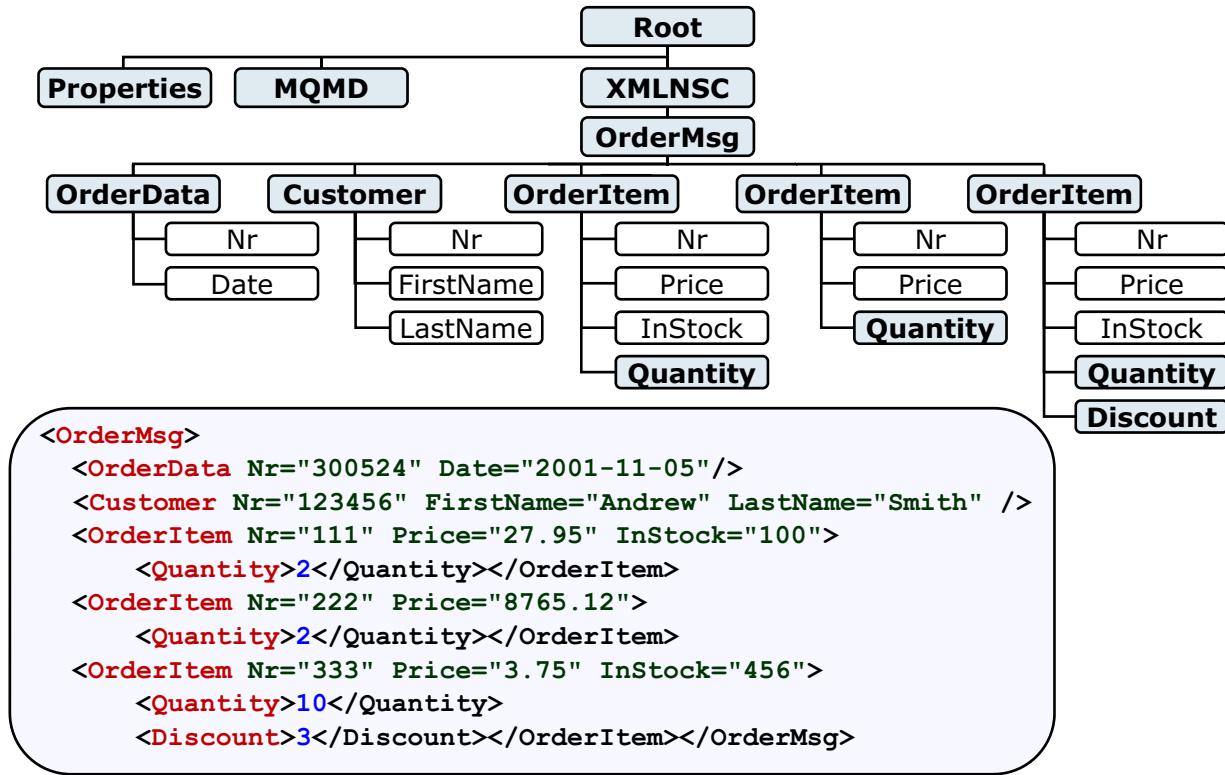
### Notes:

The ESQL module for a Compute node must contain the function `Main()`, which is the entry point for the module. The `Main()` function is included automatically if the module is created for you.

Within `Main()`, you can code ESQL to configure the behavior of the node. If you include ESQL within the module that declares variables, constants, functions, and procedures, they are of local scope only and can be used within this single module.

The ESQL editor automatically creates the code that is shown on the figure. The code in this template includes statements that you can use to copy the message headers or the entire message as part of your Compute node activities.

## OrderMsg sample message



© Copyright IBM Corporation 2015

Figure 11-9. OrderMsg sample message

WM666 / ZM6661.0

### Notes:

This sample message and tree of an order message, which is shown in the figure, are used to demonstrate the ESQL functions and statements in the next series of figures.

The order message (OrderMsg) contains order data (OrderData), customer information (Customer), and order item information (OrderItem).



#### Information

Appendix A of this Student Guide contains more examples of ESQL statements.

## Sample ESQL syntax to transform OrderMsg

### PRODUCTS table:

PRODNO	DESCR
111	Apples
222	Oranges
333	Grapes

### ESQL:

```
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[] =
(SELECT I.Nr, D.DESCRIPTION AS Description
 FROM InputBody.OrderMsg.OrderItem[] as I,
      Database.PRODUCTS as D
 WHERE I.Nr = D.PRODNO) ;
```

### Output Message:

```
<OrderMsg>
  <OrderItem>
    <Nr>111</Nr>
    <Description>Apples</Description>
  </OrderItem>
  <OrderItem>
    <Nr>333</Nr>
    <Description>Grapes</Description>
  </OrderItem>
</OrderMsg>
```

### Input Message:

```
<OrderMsg>
  <OrderItem>
    <Nr>111</Nr>
    <Price>27.95</Price>
    <Quantity>2</Quantity>
  </OrderItem>
  <OrderItem Discount="3">
    <Nr>333</Nr>
    <Price>100.75</Price>
    <Quantity>1</Quantity>
  </OrderItem>
</OrderMsg>
```

© Copyright IBM Corporation 2015

Figure 11-10. Sample ESQL syntax to transform OrderMsg

WM666 / ZM6661.0

### Notes:

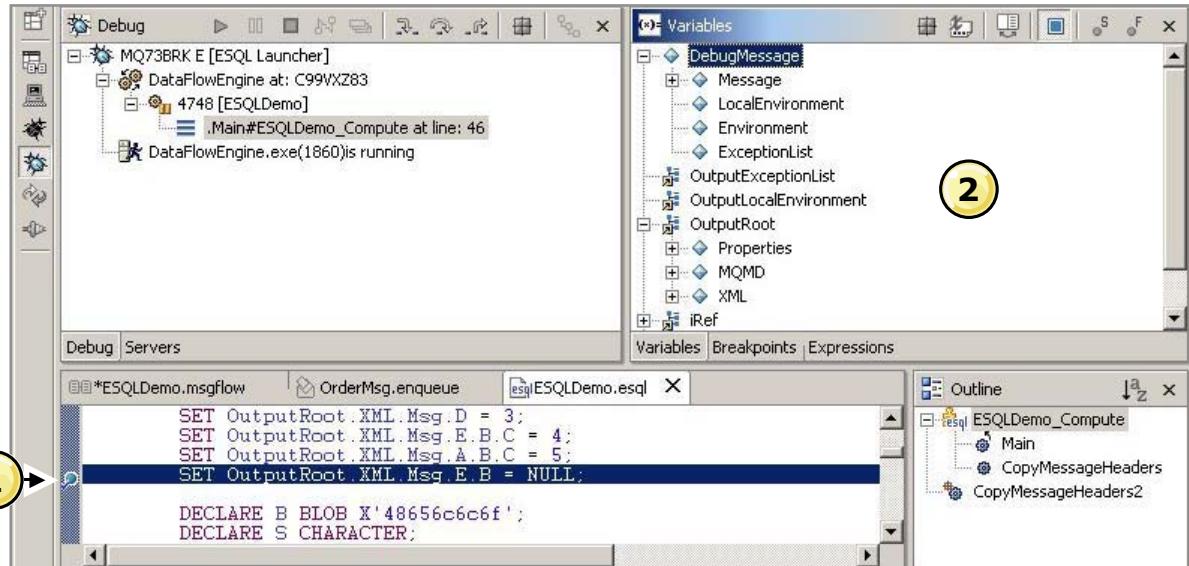
By using the ESQL syntax, you can complete complex transformations in a single statement. ESQL code is the most concise of all transformation options, and it generally has the best raw performance.

The sample code in the figure demonstrates a join between an XML message (Input Message) and a database table PRODUCTS. A new output message is built by looking up the description (DESCRIPTION) for each OrderItem in the column PRODNO.

## ESQL debugger

From the Debugger, step into the source code

1. Set breakpoints in ESQL
2. Display the message assembly



© Copyright IBM Corporation 2015

Figure 11-11. ESQL debugger

WM666 / ZM6661.0

### Notes:

The debugger in the IBM Integration Toolkit can step into the ESQL source code on a statement by statement basis.

As the figure shows, the first step is to set breakpoints in the ESQL code. The second step is to display the message assembly.

The debugger is described in detail in Unit 8, “Using problem determination tools and help resources.”

## Some ESQL syntax

- ESQL keywords are not case-sensitive  
Examples: `SET set SeT . . .`
- Correlation names and field references **are** case-sensitive  
Examples: `OutputRoot, InputBody.Msg.field`
- “Character constants” and literals are enclosed with apostrophes
- “Field references” are enclosed with quotation marks if:
  - Special characters exist in a name, such as a blank
  - The name is an ESQL reserved word
- A statement delimiter is ; (a semicolon)
- Comments
  - One line: `-- (two dashes)`
  - Block: `/* . . . comment here . . . */`

© Copyright IBM Corporation 2015

Figure 11-12. Some ESQL syntax

WM666 / ZM6661.0

### Notes:

ESQL statements can be used in an ESQL program in uppercase, lowercase, or mixed case. From a programming convention standpoint, uppercase is suggested for ease of identification. This suggestion also holds true for ESQL functions.

Correlation names, field references (also termed field names), and program variables (after they are defined) must be referenced by using the correct case. The syntax checker can assist you in avoiding typographical errors.

The use of double quotation marks (" ") for a field reference is required only when the actual value of the field happens to be the same as one of the ESQL reserved words. While it is suggested that you avoid such words, but it is not always possible with some data structures. Furthermore, some field names can contain special characters, such as a blank space, which explains the need for double quotation marks. Another example of the need for double quotation marks is when using either dots (.) or blank spaces within a field name.

A list of reserved words is available in the IBM Knowledge Center.

ESQL syntax is described in more detail in Appendix A of this guide.

## Inserting, updating, and deleting fields

- SET *modifies* a field or a structure if it exists
- SET *appends* a field or a structure (and the entire path) if it does not exist
- SET...=NULL *deletes* a field or a structure
- The order of ESQL statements determines the field order

Example:

```
CALL CopyMessageHeaders();

SET OutputRoot.XMLNSC.Msg.A.B.C = 1;

SET OutputRoot.XMLNSC.Msg.E = 2;

SET OutputRoot.XMLNSC.Msg.D = 3;

SET OutputRoot.XMLNSC.Msg.E.B.C = 4;

SET OutputRoot.XMLNSC.Msg.A.B.C = 5;

SET OutputRoot.XMLNSC.Msg.E.B = NULL;
```

© Copyright IBM Corporation 2015

Figure 11-13. Inserting, updating, and deleting fields

WM666 / ZM6661.0

### Notes:

When a SET statement runs and it sets a field that does not exist. That field and all necessary parent and sibling fields are created as the next sibling of the lowest-level ancestor that exists. So, the order of ESQL statements is important.

If the SET statement points to an existing field, that field is updated.

After applying the ESQL in the example on the figure, the message body would look as follows:

```
<Msg>
<A>
<B>
<C>5</C></B></A>
<E>2</E>
<D>3</D>
</Msg>
```

## Special ESQL data types for tree structures

- ROW: A named tree with a dynamic data type for leaf nodes
  - Copies or subsets of other ROWs > Like Root
  - SELECT the results
  - Built by using the ROW() function

```
DECLARE rowCust ROW InputRoot.XMLNSC.OrderMsg.Customer;
SET rowCust.Address = ROW('Hursley' AS Town, 'UK' AS Country);
```

- REFERENCE: Tree cursor

**Warning:** Reference to a non-existing field sets the pointer to Root, so check the success of a declaration with FIELDNAME or LASTMOVE statement

```
DECLARE refCust REFERENCE TO InputBody.OrderMsg.Customer;
IF LASTMOVE(refCust) THEN.....
IF FIELDNAME(refCust) <> 'Customer' THEN...
```

- SET refCust is not allowed

© Copyright IBM Corporation 2015

Figure 11-14. Special ESQL data types for tree structures

WM666 / ZM6661.0

### Notes:

The field references seen so far (such as correlation names like Root, and ROW variables) are static references. By declaring a REFERENCE variable, you get a dynamic reference or message cursor that can be moved in the message tree. The REFERENCE data type can hold the location of a field in a message or the location of a declared scalar variable. It cannot hold the location of a constant, a database table, a database column, or another reference.

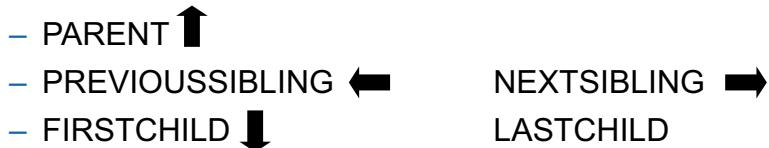
You can use dynamic references everywhere a static reference can be used. The value of the dynamic reference is that of the field to which it points.

If you declare a REFERENCE variable to a nonexistent field, the message cursor always points to Root. Use the field name of the reference (as shown on the figure) or check existence of the referred field first (as shown in the following line of code):

```
IF InputBody.Order.NonExistentField IS NOT NULL ...
```

## Moving REFERENCE variables

- Move the cursor



```
DECLARE myRef REFERENCE TO InputBody.OrderMsg;
MOVE myRef FIRSTCHILD;
MOVE myRef NEXTSIBLING NAME 'OrderItem';
MOVE myRef LASTCHILD TYPE XMLNSC.Attribute;
MOVE myRef PREVIOUSSIBLING REPEAT TYPE;
```

- Check the success of MOVE or DECLARE...REFERENCE

```
WHILE LASTMOVE(myRef) DO...
```

© Copyright IBM Corporation 2015

Figure 11-15. Moving REFERENCE variables

WM666 / ZM6661.0

### Notes:

The MOVE statement moves a dynamic variable either to a specific target field reference, or in a specific direction relative to its current position. You can further qualify the MOVE statement with a TYPE or NAME expression.

If a MOVE is not successful, the message cursor stays fixed. You must check the success of the MOVE operation with the LASTMOVE function. The LASTMOVE function is typically used in WHILE loops that iterate through a structure.

## Field references in tree structures

- Paths are dot-separated

[i]	Index (Origin 1; <1 is LAST)
[]	Array of (a list)
()	Search for type (Name, XMLNSC.Attribute, and other types)
:	A namespace followed by a name
*	Any element of any type, name, or value (or namespace)
{}	Evaluate this expression (must yield a CHAR string)

Examples:

```
DECLARE i INTEGER CARDINALITY(InputRoot.*[]);
SET rowCust.'Customer-'||C = InputBody.Customer[<2];
SET OutputRoot.MRM.addr:addressDetails.addr:postcode = 'ZZ01 4WW';
SET OutputRoot.XMLNSC.A.(XMLNSC.Attribute)B = 3;
```

© Copyright IBM Corporation 2015

Figure 11-16. Field references in tree structures

WM666 / ZM6661.0

### Notes:

You reference a field by using its correlation name, followed by zero or more path fields that are separated by periods (.). The correlation name identifies a well-known starting point and must be the name of a constant, a declared variable, or one of the predefined start points, for example, InputRoot. The path fields define a path from the start point to the required location.

In the example:

- The first statement counts the number of child elements of InputRoot (which is usually three: Properties, MQMD, and Body).
- The second statement demonstrates how to build a dynamic path. The runtime value of variable C is part of the field name. Customer[<2] means the second but last instance of repeating field Customer.
- The third statement demonstrates the usage of namespaces in ESQL.
- The last statement creates (or updates) an XML attribute. The XMLNSC parser renders each field as an XML element by default. So, if you want to address any other XML syntax element, you must use special correlation names in brackets, for example, (XMLNSC.Attribute) or (XMLNSC.XmlDeclaration).

## Deleting and reordering fields

- Delete a field or a structure

- Delete and free memory; useful for large messages

```
DELETE FIELD OutputRoot.XMLNSC.OrderMsg.Customer;
```

```
DECLARE myRef REFERENCE TO
OutputRoot.XMLNSC.OrderMsg.OrderItem[1];
DELETE NEXTSIBLING OF myRef;
```

- Set to NULL

```
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[2]=NULL;
```

- Detach (cut and paste)

```
DETACH myRef;
```

- Reorder: paste a detached structure in a new position in a message tree

```
ATTACH myRef TO OutputRoot.XMLNSC.OrderMsg
AS FIRSTCHILD;
```

© Copyright IBM Corporation 2015

Figure 11-17. Deleting and reordering fields

WM666 / ZM6661.0

### Notes:

You can use dynamic field references to cut and paste parts of the logical message. So, it is possible to copy the entire message and then to change the order of field elements or the whole tree structure by detaching first and then attaching at another place. If the order of fields in the output message is different from the fields in the input message, it might be easier to build the output message step-by-step.

After applying the ESQL on the figure to the OrderMsg example, the file looks as follows:

```
<OrderMsg>
<OrderItem Nr="111" Price="27.95" InStock="100">
<Quantity>2</Quantity></OrderItem>
<OrderData Nr="300524" Date="2001-11-05"/>
<Customer Nr="123456" FirstName="Andrew" LastName="Smith"/></OrderMsg>
```

## CREATE statement

- Create new fields (tags) or attributes
  - At any point in a message
  - A scalar field

```
SET OutputRoot=InputRoot;
CREATE FIELD OutputRoot.XMLNSC.OrderMsg.Customer.AnAttr
TYPE XMLNSC.Attribute VALUE 'C';
```

- Or a structure (per duplication)

```
DECLARE myRef REFERENCE TO OutputRoot.XMLNSC;
CREATE PREVIOUSSIBLING OF myRef DOMAIN 'MQRFH2';
```

```
DECLARE myRef REFERENCE TO OutputRoot.XMLNSC.OrderMsg;
CREATE FIRSTCHILD OF myRef
FROM InputBody.OrderMsg.OrderItem[3];
```

© Copyright IBM Corporation 2015

Figure 11-18. CREATE statement

WM666 / ZM6661.0

### Notes:

To bring the tree navigation completely under your control, it is necessary to manually create new fields. The CREATE function is provided to give you this ability.

The CREATE function is a flexible function with many alternative forms.

By using the CREATE function with a dynamic field reference, you can create fields relative to existing fields anywhere in the message tree. You can specify TYPE, NAME, and VALUE explicitly. This action results in a single field (a scalar) being created. Here are more samples:

```
CREATE NEXTSIBLING OF myRef TYPE Name NAME 'anEmptyTag';
CREATE PREVIOUSSIBLING OF myRef TYPE Name NAME 'TagNoContent' VALUE NULL;
```

You can duplicate a field (including its subtypes) by using CREATE with a FROM clause and a field reference.

## NULL value

- NULL is a distinct state
  - Not 0 or ''
  - SQL\_NULL\_DATA
  - All ESQL data types (except REFERENCE)
- If value is
  - Unknown
  - Undefined
  - Uninitialized
- General rule: If any operand is NULL, then value is NULL
- Use NULL to delete a field or a structure

Example:

```
SET OutputRoot.XMLNSC.Msg.Field=NULL;
SET OutputRoot.XMLNSC.Msg.F2= InputBody.Msg.Unknown;
```

© Copyright IBM Corporation 2015

Figure 11-19. NULL value

WM666 / ZM6661.0

### Notes:

Always consider what would happen if a message field is not there (UNKNOWN or NULL).

A common runtime problem is shown in the last line of the ESQL example. Field F2 is deleted because field Unknown is not present in InputBody. This behavior might not be the expected behavior. You can discover this type of problem in a debug level User Trace.

Setting a field to NULL deletes it only if NULL is not allowed as a value (which is the default).

## Procedures

- Subroutines with IN, OUT, and INOUT parameters and an optional RETURNS value
  - Can be used recursively
  - Language can be ESQL, Java, or a database (stored procedure)

```
CREATE PROCEDURE navigate (IN root REFERENCE, INOUT answer CHARACTER)
LANGUAGE ESQL
BEGIN
  SET answer = answer || 'Reached Field...Name:' || FIELDNAME(root);
  DECLARE cursor REFERENCE TO root;
  MOVE cursor FIRSTCHILD;
  IF LASTMOVE(cursor) THEN
    SET answer = answer || 'Field has children... drilling down ';
  ELSE
    SET answer = answer || 'Listing siblings... ';
  END IF;
  WHILE LASTMOVE(cursor) DO
    CALL navigate(cursor, answer);
    MOVE cursor NEXTSIBLING;
  END WHILE;
  SET answer = answer || 'Finished siblings... Popping up ';
END;
```

© Copyright IBM Corporation 2015

Figure 11-20. Procedures

WM666 / ZM6661.0

### Notes:

The CREATE PROCEDURE statement defines a procedure. A procedure is a subroutine that can have IN, OUT, and INOUT parameters. Parameters allow the procedure to return values to the caller. Procedures can also have a RETURN value and be used as functions.

The CALL statement calls a named procedure that was defined by using the CREATE PROCEDURE statement. The CALL statement passes the parameters into the procedure in the declared order. Parameters that are defined as IN or INOUT are evaluated before the CALL is made, but parameters that are defined as OUT are always passed in as NULL parameters of the correct type.

When the procedure is complete, any parameters that are declared as OUT or INOUT are updated to reflect any changes that were made to them when the procedure is run. The procedure never changes parameters that are defined as IN.

The number and type of parameters that is passed to the procedure must match the number and type of the parameters as declared in the CREATE PROCEDURE definition.

Output message trees cannot be accessed directly within the body; they must be passed in as a REFERENCE.

Procedures and functions can be called from within each other, so recursion is supported. The procedure that is shown in the figure recursively traverses the message tree (or whatever tree structure it is pointed to) and prints the tree contents in infix notation.

## Variables

- Scalar and fixed data type
- Tree structures (ROW and REFERENCE)

```
>-DECLARE-----Name-----+-----+---+---+---+---+---+---+>
    +--<<---,--<<---+   +-CONSTANT-+           |+-InitialValueExpr-+
        +-NAMESPACE-----+
        +-NAME-----+
```

```
DECLARE var, I INTEGER 1;
```

```
DECLARE addr NAMESPACE 'http://www.ibm.com/address' ;
```

```
DECLARE course CONSTANT CHAR 'WM665' ;
```

© Copyright IBM Corporation 2015

Figure 11-21. Variables

WM666 / ZM6661.0

### Notes:

An ESQL variable is a data field that is used to help process a message. You must declare a variable and state its type before you can use it.

The data type of a variable is fixed. If the ESQL code assigns a value of a different type, either an implicit cast to the data type of the target is implemented or an exception is raised (if the implicit cast is not supported).

You can assign an initial value to the variable on the DECLARE statement. If an initial value is not specified, scalar variables are initialized with the special value NULL, and ROW variables are initialized to an empty state.

To define a variable and give it a name, use the DECLARE statement. The names of ESQL variables are case-sensitive.

This figure shows the syntax and some examples of declaring variables in ESQL.

## Variable scope, lifetime, and sharing (1 of 2)

- **Scope:** Where the variable is used  
Examples: Node, flow, integration server, and integration node
- **Lifetime:** Variable duration  
Examples: Code block, module, and thread
- **Sharing:** Thread visibility
  - When the flow or integration server stops, **SHARED** variable lifetime ends
  - **ATOMIC** blocks can serialize access to **SHARED** variables

Example:

```
DECLARE s_counter SHARED INT 1;
.....
BEGIN ATOMIC
SET s_counter = s_counter+1;
END;
```

© Copyright IBM Corporation 2015

Figure 11-22. Variable scope, lifetime, and sharing (1 of 2)

WM666 / ZM6661.0

### Notes:

It is sometimes necessary to store data for longer than the lifetime of a single message that is passing through a flow. An option is to store the data in a database. Storing the data in a database is good for long-term persistence and maintaining the transaction, but access (particularly write access) is slow.

Alternatively, you can use appropriate “long-lived” ESQL data types to cache data for a certain period. This option makes access much faster than if it was accessed from a database, but at the expense of shorter persistence and no way to maintain the transaction.

Long-lifetime variables are created by using the **SHARED** keyword on the **DECLARE** statement.

Only one instance of a message flow (that is, one thread) is allowed to run the statements of a specific **BEGIN ATOMIC... END** statement (identified by its schema and label), at any one time.

It is not necessary to use the **BEGIN ATOMIC** construct in flows that are never deployed with more than one instance. It is also unnecessary to use the **BEGIN ATOMIC** construct on reads and writes to shared variables. The integration node always safely writes a new value to, and safely reads the latest value from a shared variable. **ATOMIC** is only required when the application is sensitive to seeing intermediate results.

## Variable scope, lifetime, and sharing (2 of 2)

### Short life variables

Variable type	Scope	Life	Shared
Schema and module	Flow node	Thread within a flow node	No
Routine local	Flow node	Thread within a routine	No
Block local	Flow node	Thread within a block	No

### Long life variables

Variable type	Scope	Life	Shared
Flow Node SHARED	Flow node	Life of flow node	All threads of flow
Flow SHARED	Flow	Life of flow	All threads of flow

© Copyright IBM Corporation 2015

Figure 11-23. Variable scope, lifetime, and sharing (2 of 2)

WM666 / ZM6661.0

### Notes:

Long-lived data types have an extended lifetime beyond the lifetime of a single message that is passing through a node. They are shared between threads and exist for the life of a message flow. Strictly speaking, the lifetime is the time between configuration changes to a message flow.

A typical use of these data types might be in a flow in which data tables are “read-only” as far as the flow is concerned. Although the table data is not static, the flow does not change it, and thousands of messages pass-through the flow before there is any change to the table data.

An example is a table that contains the credit card transactions for a day. The table is created each day and the messages for that day are run against it. Then, the flow is stopped. The table is updated, and the messages for the next day are run. It is likely that such flows would do better if they cached the table data rather than reading it from a database for each message.

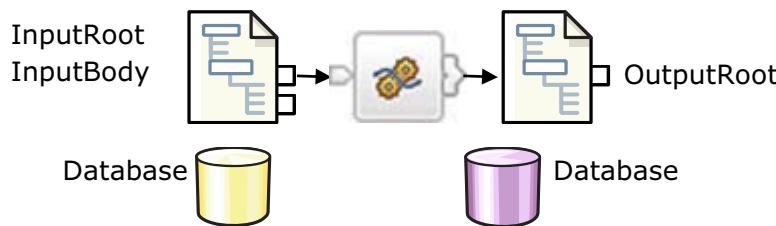
Another use of these data types might be the accumulation and integration of data from multiple messages.

There are a number of advantages to using long-life variables:

- The ability to handle large amounts of long-lifetime data
- The joining of data to messages quickly

- On multiple processor servers, multiple threads can access the same data simultaneously
- Subsequent messages can access the data that remains from a previous message
- Long lifetime read/write data can be shared between threads because there is no long-term association between threads and messages
- In contrast to data stored in database tables in the environment, this type of data is stored privately; that is, within the integration node
- ROW variables can be used to create a modifiable copy of the input message; see ESQL **ROW** data type
- It is possible to create shared constants, as stated previously

## Using a Compute node to access a database



- Retrieve the data from an application database and include it in an output message
- Set DSN in node properties and write or generate ESQL

Example:

```

SET OutputRoot.XML.Message.Price =
THE (SELECT ITEM A.PRICE
FROM Database.PRICEDATA AS A
WHERE A.PRODUCT = 'Widget');
    
```

- Compute node can also UPDATE, INSERT, or DELETE a database

© Copyright IBM Corporation 2015

Figure 11-24. Using a Compute node to access a database

WM666 / ZM6661.0

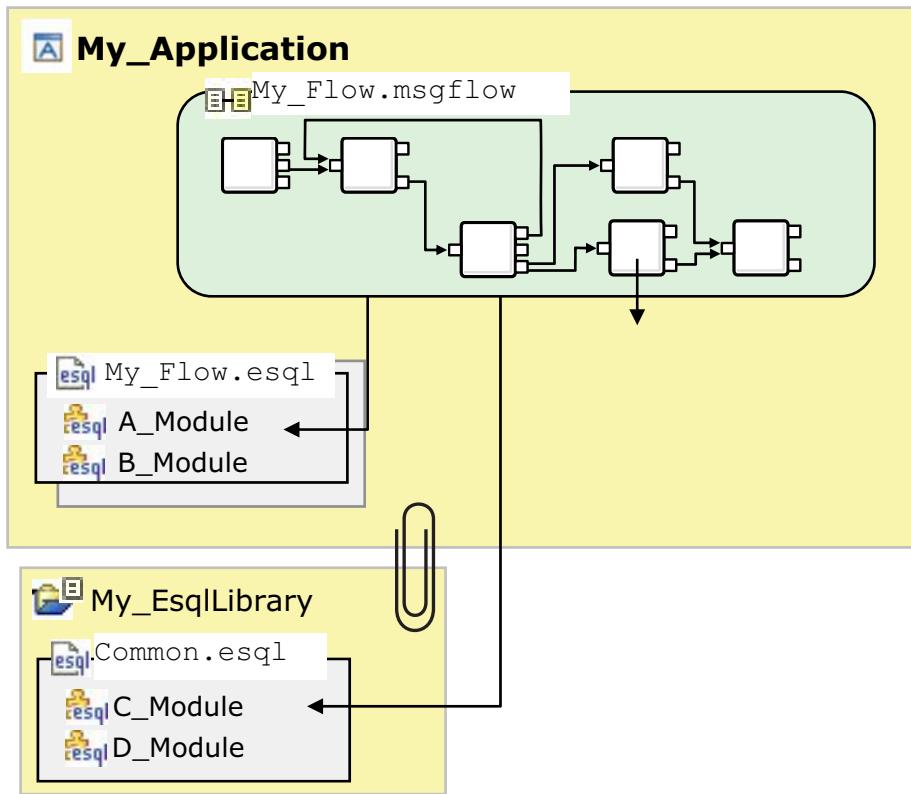
### Notes:

One way to extract data from database tables and pass it in a field in a message or use it to enrich a field (like a calculation) is to use a Compute node.

The ODBC DSN to reference is set in the Compute node properties; the ESQL code refers to it with the correlation name of **Database**. The DSN property can also be configured in the deployment descriptor of the BAR file. The result is that the ESQL code (or the mappings) is not bound to a special database.

A simple example of the ESQL that is required to set the value of an output field in a message to a value retrieved from within a database row is shown in the figure. If the PRODUCT column in the table PRICEDATA has a value of “Widget,” then the **Price** field in the output message is set to the value taken from the column PRICE of that row.

## Reusable ESQL



© Copyright IBM Corporation 2015

Figure 11-25. Reusable ESQL

WM666 / ZM6661.0

### Notes:

Some process nodes, such as the Compute node, refer to ESQL modules that are contained in \*.esql text files or a referenced message flow project. An ESQL file can contain multiple modules, and multiple nodes in various flows can reference each module.

The ESQL files can be contained in the application or in a library. You can reuse ESQL code and build common code libraries, provided the message flow application has appropriate references to libraries.



## JavaCompute node

- Use Java to complete the following tasks:
  - Examine an incoming message and depending on its content, propagate it unchanged to one of the two output terminals of the node (similar to Filter node)
  - Change part of an incoming message and propagate the changed message to one of the output terminals
  - Create and build an output message that is independent of the input message
  - Create a map in a global cache, and add and retrieve data from that map
- Java code can be coded by using any of the following programming styles:
  - Java plug-in API
  - Java Architecture for XML Binding (JAXB)
  - Document Object Model (DOM)
- IBM Integration Toolkit wizard guides you through the creation of a Java class that contains template code

© Copyright IBM Corporation 2015

Figure 11-26. JavaCompute node

WM666 / ZM6661.0

### Notes:

You can create a Java class file for a JavaCompute node and code Java functions to tailor the behavior of the node. You can add any valid Java code to your JavaCompute nodes and use the Java user-defined node API to process messages.

By storing data in the global cache, that data is available to other JavaCompute nodes or message flows.

When you add a JavaCompute node to a message flow, you can start a wizard to guide you through the creation of a code template that is based on your requirements.

## Getting started with the JavaCompute node

- Open the Java wizard
  - Guides the user and creates a correct code template
  - Java package and class identified
  - Places the user in an Eclipse Java perspective
- Java perspective: Full Eclipse experience for Java development
  - Javadoc for content assists
  - Incremental compilation
- Deployment of Java
  - Automatic when the Java Compute node flow is added to a BAR file
  - Workspace searched and a .jar file automatically added to BAR
  - External .jar files can be added to a project class path
  - Stored on the integration node file system



© Copyright IBM Corporation 2015

Figure 11-27. Getting started with the JavaCompute node

WM666 / ZM6661.0

### Notes:

When a message flow that contains a JavaCompute node is added to a BAR file, the JAR file that contains the Java code is also added to the BAR file automatically. It is also possible for the Java code to reference code in other Java projects in the Eclipse workspace (internal dependency) and external JAR files on the file system (external dependency).

To reference code in other Java projects:

1. Right-click the project folder and click **Properties**.
2. Select **Java Build Path** on the left pane.
3. Select the **Libraries** tag.
4. Select **Add JARs** to select internal dependencies or **Add External JARs** to select external dependencies.

When the message flow is added to the BAR file, all internal dependencies get automatically added too. External dependencies do not get added automatically.

You must create a Java class file for each node in which you code Java functions to tailor the behavior of the node. Java files are managed through the Java perspective in the Integration Toolkit.

The screenshot shows the WebSphere Education interface. At the top left is the "WebSphere Education" logo. At the top right is the "IBM" logo. The main title "Using the Java wizard (1 of 2)" is centered above a window titled "New Java Compute Node Class". The window contains fields for "Source folder" (set to "PurchaseOrderProcessingJava"), "Name" (set to "Test\_JavaCompute"), and "Modifiers" (set to "public"). A yellow callout box labeled "1" provides instructions: "Double-click the JavaCompute node or Right-click the JavaCompute node and click Open Java". Another yellow callout box labeled "2" points to the "Source folder" field with the instruction: "Set Source folder to the application or library that contains the Java classes". Buttons for "Finish", "Next >", "Back <", and "Cancel" are at the bottom.

© Copyright IBM Corporation 2015

Figure 11-28. Using the Java wizard (1 of 2)

WM666 / ZM6661.0

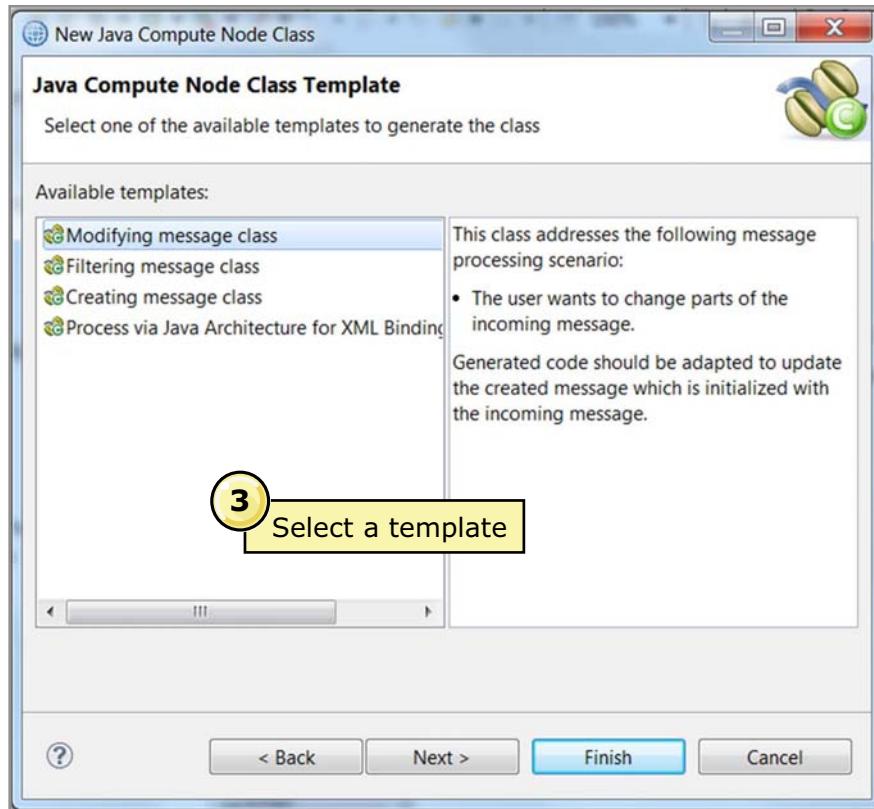
### Notes:

The figure shows the first two steps for creating Java for the Java Compute node by using the wizard:

1. You can either double-click the JavaCompute node on the message flow canvas or right-click the JavaCompute node and then click **Open Java** from the menu.
2. Set the source folder.



## Using the Java wizard (2 of 2)



© Copyright IBM Corporation 2015

Figure 11-29. Using the Java wizard (2 of 2)

WM666 / ZM6661.0

### Notes:

The final step in the New Java Compute Node Class wizard is to select the template.

Similar to the Compute node, a block of standard text is included in the JavaCompute node to help you get started. The template that you select in the **New Java Compute Node Class** wizard determines the code template.



## JavaCompute templates

- Only one template per JavaCompute node
  - Modifying: Read/write; copies entire message
  - Filtering: Read-only
  - Creating: Empty `outMessage` and `CopyMessageHeaders` method
  - Processing messages by using JAXB of Java class objects
- `Class MessageFlowName_NodeName extends com.ibm.broker.javacompute.MbJavaComputeNode`

© Copyright IBM Corporation 2015

Figure 11-30. JavaCompute templates

WM666 / ZM6661.0

### Notes:

When you run the **New Java Compute Node Class** wizard, you must select one of the available templates to generate the code template for the node.

On the Java Compute Node Class Template page, choose one of the following options:

- For a filter node template code, select **Filtering message class**.
- To change an incoming message, select **Modifying message class**.
- To create a message, select **Creating message class**.
- To process messages by using the JAXB template, select **Process via JAXB class**.

## JavaCompute template example

```

import com.ibm.broker.javacompute.MbJavaComputeNode;
...Other import statements for com.ibm.broker.plugin

public class Test_JavaCompute extends MbJavaComputeNode {

    public void evaluate(MbMessageAssembly inAssembly) throws MbException {
        MbOutputTerminal out = getOutputTerminal("out");
        MbOutputTerminal alt = getOutputTerminal("alternate");

        MbMessage inMessage = inAssembly.getMessage();
        MbMessageAssembly outAssembly = null;
        try {
            // create new message as a copy of the input
            MbMessage outMessage = new MbMessage(inMessage);
            outAssembly = new MbMessageAssembly(inAssembly, outMessage);
            // -----
            // Add user code
            // -----
        }
        ...Code for catching Exceptions
        // The following should only be changed
        // if not propagating message to the 'out' terminal
        out.propagate(outAssembly);
    }
}

```

© Copyright IBM Corporation 2015

Figure 11-31. JavaCompute template example

WM666 / ZM6661.0

### Notes:

The figure shows an example of a JavaCompute template for modifying a message.

The incoming message and message assembly are read-only. So, before you can modify a message, you must make a copy of the incoming message. The “modifying” class template creates a copy of the incoming message by calling the copy constructors:

```

MbMessage outMessage = new MbMessage(inAssembly.getMessage());
MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly, outMessage);

```

The new `outAssembly` object gets propagated to the next node.

User code is inserted between the comment lines:

```

// Add user code below
// End of user code

```



## JavaCompute node basics (1 of 2)

- Incoming message is passed into the `evaluate` method as part of a message assembly
  - `MbMessageAssembly` encapsulates four `MbMessage` objects
- Must implement the `evaluate()` method, which the integration node calls after each message is processed
- `MbElement` represents a “parsed” logical part of the message
- Message assembly is propagated to an output terminal:  
`out.propagate(outAssembly)`

© Copyright IBM Corporation 2015

Figure 11-32. JavaCompute node basics (1 of 2)

WM666 / ZM6661.0

### Notes:

You can add any valid Java code to a JavaCompute node, making full use of the existing Java user-defined node API to process an incoming message. You can use the Java editing functions in Eclipse to develop your Java code. Java editing functions include code completion, integrated Javadoc documentation, and automatic compilation.

JavaCompute node can be used to examine the content of an input message, transform a message, and build new messages.

You can create a Java class file for a JavaCompute node and code Java functions to tailor the behavior of the node.

## JavaCompute node basics (2 of 2)

- Each element has:

`getObjectType()` A generic type that returns one of either NAME, VALUE, or NAME/VALUE  
`getName()` A name that returns a string  
`getValue()` A value that returns an object of a parser-determined type  
Other properties such as a namespace URI, a parser-specific type

- Access user-defined attributes with:

`getUserDefinedAttribute(String name)`

- Utility methods are provided to show users how to do common tasks such as `copyMessageHeaders()`

© Copyright IBM Corporation 2015

Figure 11-33. JavaCompute node basics (2 of 2)

WM666 / ZM6661.0

### Notes:

The figure lists some of the methods available to return information about the referenced element. You can customize a JavaCompute node to access properties that you associated with the message flow in which the node is included. To access these properties from a JavaCompute node, use the `getUserDefinedAttribute(name)` method, where `name` is the name of the property that you are accessing. The type of the object that is returned depends on the type of the property that you are accessing.

## Message assembly and standard trees

- Standard trees are contained in `MbMessageAssembly`
  - `Message` (read-only)
  - `LocalEnvironment` (read-only)
  - `GlobalEnvironment`
  - `ExceptionList` (read-only)
- Trees start at Root
  - `MbElement` is an object that represents an element

Java accessor from <code>MbMessageAssembly</code>	Equivalent ESQL Correlation name
<code>getMessage().getRootElement()</code>	<code>InputRoot</code>
<code>getMessage().getRootElement().getLastChild()</code>	<code>InputBody</code>
<code>getLocalEnvironment().getRootElement()</code>	<code>InputLocalEnvironment</code>
<code>getGlobalEnvironment().getRootElement()</code>	<code>Environment</code>
<code>getExceptionList().getRootElement()</code>	<code>InputExceptionList</code>

© Copyright IBM Corporation 2015

Figure 11-34. Message assembly and standard trees

WM666 / ZM6661.0

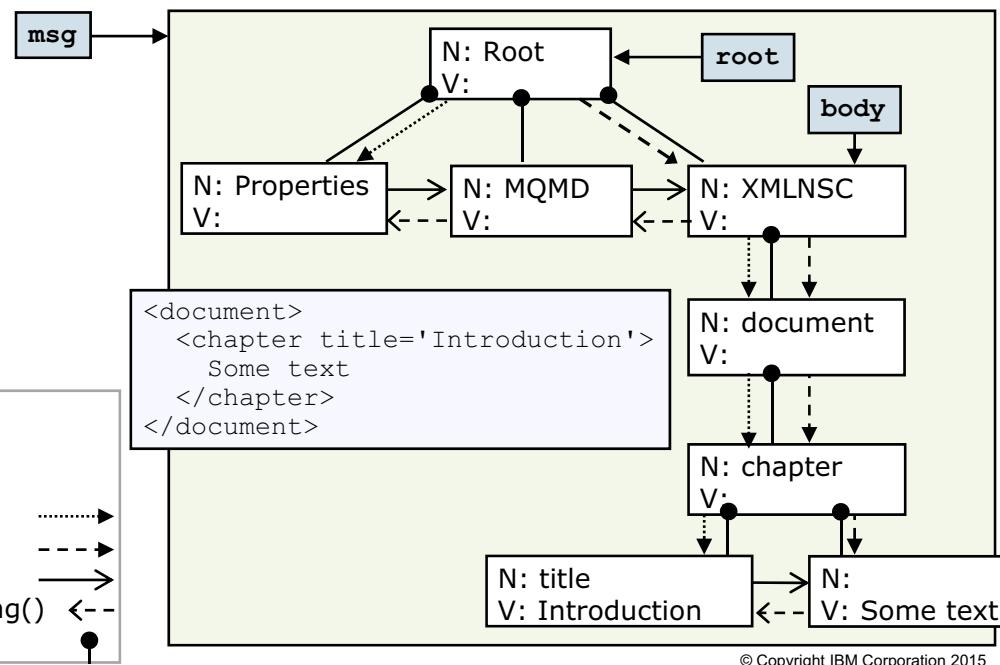
### Notes:

The Java user-defined node API includes some extra methods that simplify tasks that involve message routing and transformation. These tasks include accessing named elements in a message tree, setting their values, and creating elements without browsing the tree explicitly.

`MbMessageAssembly` wraps the four message objects. Messages and assembly are read-only except the global environment. If you try to write to them, an `MbReadOnlyException` is reported.

## Using traversal methods to browse message trees

```
MbMessage msg = assembly.getMessage();
MbElement root = msg.getRootElement();
MbElement body = root.getLastChild();
MbElement chapter = body.getFirstChild().getFirstChild();
```



© Copyright IBM Corporation 2015

Figure 11-35. Using traversal methods to browse message trees

WM666 / ZM6661.0

### Notes:

The figure shows an example of the message tree. To move the pointer in the tree, start at the root and use the `MbMessageAssembly` traversal methods.

After you move to an element in the tree, you can use methods like `getValue()` or `getName()` to return information about the referenced element.

Also, see the Javadoc for the Java user-defined node API, which is accessible from the IBM Knowledge Center. It provides further explanation on the classes and methods that the JavaCompute node and custom-written Java nodes use.

## Using XPath in JavaCompute node

- Use the `MbMessage.evaluateXPath` method on an appropriate tree element
- Four return types
  - `java.lang.Boolean`, `Double`, and `String` for corresponding XPath values
  - `java.util.List` for XPath `Nodesets`; direct or iterated access

```
String title = (String) message
    .evaluateXPath("string(document/chapter/@title)");
List chapterList = (List)message.evaluateXPath("//chapter");
Iterator itr = chapterList.iterator();
while (itr.hasNext()) {
    MbElement chapter = (MbElement)itr.next();
    ...
}
```

- Variable binding allows a runtime expression evaluation

```
titleExtractor = new MbXPath(
    "string(/library/books/book[@isbn = $ISBN]/title)");
titleExtractor.assignVariable("ISBN", "0140620168");
(String)message.evaluateXPath(titleExtractor);
```

© Copyright IBM Corporation 2015

Figure 11-36. Using XPath in JavaCompute node

WM666 / ZM6661.0

### Notes:

The `evaluateXPath()` method can be called on a `MbMessage` object (for absolute paths) or a `MbElement` object (for relative paths). The XPath expression is passed as a string parameter. A second form of this method is provided, which takes an `MbXPath` object. This object encapsulates an XPath expression along with variable bindings and namespace mappings, if they are required.

XPath supports the ability to refer to variables inside an expression that are assigned before evaluation. The `MbXPath` class has three methods for assigning and removing these variable bindings from user Java code. The value must be one of the four XPath supported types (Boolean, string, number, or nodeset).

XPath performance tips:

- Use `/aaa[1]` if you want the first one. It stops searching when it finds it.
- Avoid descendant-or-self `(//)` if possible. It traverses (and parses) the whole message. Use `/descendant::aaa[1]` instead of `(//aaa)[1]`

## Updating the tree

- Requires a writable assembly
- Setting the element name/value/type:

```
setName()
setValue()
setNamespace()
setSpecificType()
```

- Creating elements:

```
createElementAsFirstChild()
createElementAsLastChild()
createElementBefore()
createElementAfter()
```

- Copying and moving elements: `detach()` and `copy()`

- Adding unattached elements:

```
addAsFirstChild(element)
addAsLastChild(element)
addBefore(element)
addAfter(element)
```

© Copyright IBM Corporation 2015

Figure 11-37. Updating the tree

WM666 / ZM6661.0

### Notes:

Many message transformation scenarios require a new outgoing message. The “creating” template in the wizard generates skeleton code for building a new message. The default constructor of `MbMessage` gets called to create a blank message.

```
MbMessage outMessage = new MbMessage();
```

Optionally, the headers can be copied from the incoming message by using the supplied utility method, `copyMessageHeaders()`. For example:

```
copyMessageHeaders(inMessage, outMessage);
```

The message body can now be created. First, the top-level parser element must be added. For XMLNSC, the syntax is:

```
MbElement outRoot = outMessage.getRootElement();
MbElement outBody = outRoot.createElementAsLastChild("XMLNSC");
```

The remainder of the message can then be built by using the `createElementXXX()` methods or the extended syntax of the integration node XPath implementation, as shown in the figure.



## JavaCompute node at run time

- The `MbService` class writes to the Windows Event Viewer, the UNIX syslog, and the z/OS job log
- Exception handling
  - Made available by using the `MbException` hierarchy
  - Usually not necessary to catch exceptions; the integration node propagates to a flow node that is wired to the **Failure** terminal
- You can enable and disable Java isolation for individual applications by setting the **Java Isolation** configuration property on the **Manage** tab of the BAR file editor

© Copyright IBM Corporation 2015

Figure 11-38. JavaCompute node at run time

WM666 / ZM6661.0

### Notes:

The `MbService` class contains some static methods for writing to the event log or syslog. You can define message catalogs that use Java resource bundles to store the message text. Three levels of severity are supported: information, warning, and error.

The integration node exception model is exposed by using the hierarchy of exception classes that extend `MbException`. In general, it is not necessary to catch these exceptions. However, if caught, these exceptions must be rethrown. The integration node then constructs the exception list and pass the incoming message to the failure terminal, if it is connected. Otherwise, it rethrows the exception back to a Catch or input node.

The only supported method of fully coordinated database access is by using the `MbSQLStatement` class.

## JavaCompute node database access example

### JDBCType4Connection

```
// Add user code below
Connection conn = getJDBCType4Connection("MSGSTORE",
    JDBC_TransactionType.MB_TRANSACTION_AUTO);
MbElement mqmd = message.getRootElement().get FirstElementByPath("MQMD");
try (
    PreparedStatement insertStmt = conn
        .preparedStatement("INSERT INTO MQSIUSER.COMPLAIN" +
            " (MSDID, RECEIVED, MESSAGE) VALUE(?, ?, ?)");
    insertStmt.setBytes(1, (byte[]) mqmd.getFirstElementByPath("MsgId");
        .getValue());
    Timestamp ts = new Timestamp(System.currentTimeMillis());
    insertStmt.setTimestamp(2,ts);
    insertStmt.setBytes(3, (byte[]) message.getBuffer());
    insertStmt.execute();
) catch (SQLException e) (
    e.printStackTrace();
)
// End of user code
```

© Copyright IBM Corporation 2015

Figure 11-39. JavaCompute node database access example

WM666 / ZM6661.0

### Notes:

The figure shows an example of how to use a JavaCompute node to insert a row into a database table.

## Combine ESQL and Java

- Run ESQL in a JavaCompute node with `MbSQLStatement`
- Call static Java methods from a Compute node ESQL module
  - Procedure is called as a Java method if it specifies a LANGUAGE clause of JAVA
  - An exact one-to-one matching of the data types and directions of each parameter between the definition and the CALL is required

© Copyright IBM Corporation 2015

Figure 11-40. Combine ESQL and Java

WM666 / ZM6661.0

### Notes:

In some cases, it is useful to combine ESQL and Java.

The strength of ESQL is concise code, especially when combining database, and logical messages. The strength of Java is the large code base that is available and having functions that are not available in ESQL.

For example, the only way to pause the message flow from within ESQL code is to call the Java `Thread.sleep` method. The figure shows how to call a Java method from ESQL, create a static Java method, wrap it into an ESQL procedure, and call it as you would any other procedure.

## Example: Calling Java method from ESQL

### 1. Develop Java code in the IBM Integration Toolkit Java perspective:

```
public static String getHostName()
{
    try
    {
        java.net.InetAddress localMachine = java.net.InetAddress.getLocalHost();
        return localMachine.getHostName();
    }
    catch(java.net.UnknownHostException uhe)
    {
        //handle exception
    }
}
```

### 2. Wrap the Java method as an ESQL procedure:

```
CREATE PROCEDURE getHostName( )
RETURNS CHARACTER
LANGUAGE JAVA
EXTERNAL NAME "com.<yourcompanyname>.middleware.common.util.
WmbUtils.getHostName";

SET hostname = getHostName();
```

© Copyright IBM Corporation 2015

Figure 11-41. Example: Calling Java method from ESQL

WM666 / ZM6661.0

### Notes:

The example in the figure shows how to include a Java method that gets a hostname in ESQL.

## Unit summary

Having completed this unit, you should be able to:

- Use the Compute node and ESQL to transform messages
- Use the JavaCompute node and Java to transform messages

© Copyright IBM Corporation 2015

Figure 11-42. Unit summary

WM666 / ZM6661.0

### Notes:



## Checkpoint questions (1 of 2)

1. True or False: You can reuse Java code that is written for IBM Integration Bus in any other JVM (not an IBM Integration Bus integration server).
  
2. True or False: An ESQL file in an Integration Bus application contains ESQL for every node in the message flow that requires ESQL.

© Copyright IBM Corporation 2015

Figure 11-43. Checkpoint questions (1 of 2)

WM666 / ZM6661.0

### Notes:

Write your answers here:

1.

2.

## Checkpoint answers (1 of 2)

1. True or False: You can reuse Java code that is written for IBM Integration Bus in any other JVM (not an IBM Integration Bus integration server).

**Answer: False.** Any JavaCompute node class extends `com.ibm.broker.javacompute.MbJavaComputeNode`, which sets up the special IBM Integration Bus environment.

2. True or False: An ESQL file in an Integration Bus application contains ESQL for every node in the message flow that requires ESQL.

**Answer: True.**

© Copyright IBM Corporation 2015

Figure 11-44. Checkpoint answers (1 of 2)

WM666 / ZM6661.0

### Notes:



## Checkpoint questions (2 of 2)

3. What are ESQL correlation names? Name some examples.
4. What is the difference between OutputRoot and Root?
5. How can you access the IBM Integration Bus message trees in JavaCompute node?

© Copyright IBM Corporation 2015

Figure 11-45. Checkpoint questions (2 of 2)

WM666 / ZM6661.0

### Notes:

Write your answers here:

3.

4.

5.

## Checkpoint answers (2 of 2)

3. What are ESQL correlation names? Name some examples.

**ESQL correlation names are pointers into the message tree. An ESQL correlation name is a starting point for the path to fields such as Root, ExceptionList, and InputBody.**

4. What is the difference between OutputRoot and Root?

**In the Compute node, you must distinguish between the input message structure and the new output data structure. In other ESQL nodes (such as Filter and Database), the logical message (Root tree) cannot be changed, so a distinction between InputRoot and OutputRoot is not necessary or allowed.**

5. How can you access the IBM Integration Bus message trees in the JavaCompute node?

**The message tree is passed to a JavaCompute node as an argument (MbMessageAssembly object) of the evaluate method.**

**First, get the relevant tree and its root element from the MbMessageAssembly object. Then, use the traversal of the MbElement or evaluateXPath methods.**

© Copyright IBM Corporation 2015

Figure 11-46. Checkpoint answers (2 of 2)

WM666 / ZM6661.0

### Notes:

## Exercise 10



Transforming data by using the  
Compute and JavaCompute nodes

© Copyright IBM Corporation 2015

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

10.1

WM666 / ZM6661.0

Figure 11-47. Exercise 10

### Notes:



## Exercise objectives

After completing this exercise, you should be able to:

- Use a Compute node or JavaCompute node in a message flow application to transform a message

© Copyright IBM Corporation 2015

Figure 11-48. Exercise objectives

WM666 / ZM6661.0

### Notes:

See the *Student Exercises Guide* for detailed instructions.



# Unit 12. Processing JMS, HTTP, and web service messages

## What this unit is about

This unit introduces IBM Integration Bus support for HTTP, JMS, and web services.

## What you should be able to do

After completing this unit, you should be able to:

- Describe how to use message flow applications with JMS
- Describe how message flow applications can support Hypertext Transfer Protocol (HTTP) and SOAP messages
- Explain how the Web Services Definition Language (WSDL) file is used to develop web services message flows

## How you will check your progress

- Checkpoint questions

## References

IBM Knowledge Center



## Unit objectives

After completing this unit, you should be able to:

- Describe how to use message flow applications with JMS
- Describe how message flow applications can support Hypertext Transfer Protocol (HTTP) and SOAP messages
- Explain how the Web Services Definition Language (WSDL) file is used to develop web services message flows

© Copyright IBM Corporation 2015

Figure 12-1. Unit objectives

WM666 / ZM6661.0

### Notes:



## Java Message Service (JMS)

- Vendor-independent messaging API in Java
  - Oracle owns the specification
  - Managed by the Java Community Process
  - Expert group includes IBM
- Ensures provider independence but not interoperability
- Supports both point-to-point and publish/subscribe
- Asynchronous message delivery
- API does not define a wire protocol
- No security
- Supports transactions

© Copyright IBM Corporation 2015

Figure 12-2. Java Message Service (JMS)

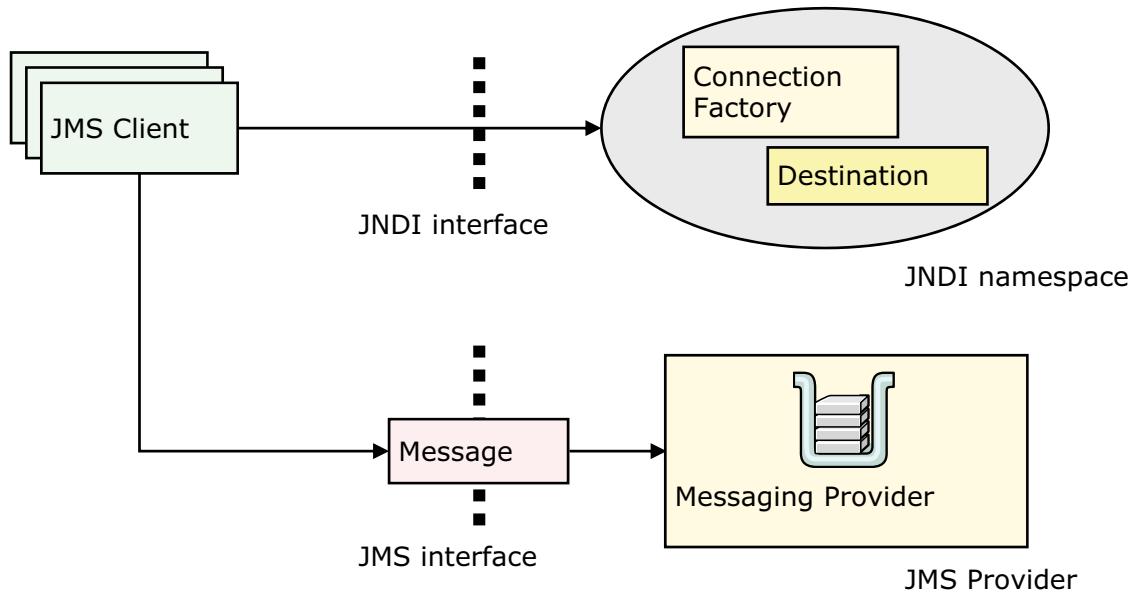
WM666 / ZM6661.0

### Notes:

JMS provides a standard API for Java applications to use for enterprise messaging. Java standards are controlled through the Java Community Process. It was developed with input from many messaging providers, including IBM.

JMS does not specify anything other than the programming interface. In particular, it does not specify the wire format that is used to communicate between remote applications. In other words, it does not specify how different JMS providers should interoperate (except at the API level).

## JMS architecture



© Copyright IBM Corporation 2015

Figure 12-3. JMS architecture

WM666 / ZM6661.0

### Notes:

This figure provides an overview of the JMS architecture. JMS messaging applications can consist of the following parts:

- JMS client: Java program that produces and uses messages by using JMS
- Messaging provider: The enterprise messaging system, for example, IBM MQ
- Message: The object that communicates information between the client and the provider
- Administered objects (Connection Factory and Destination): Preconfigured JMS objects that an administrator creates for the use of clients

Java Naming and Directory Interface (JNDI) is a standard Java extension that provides a uniform API for accessing various directory and naming services. JMS clients use JNDI to browse a naming service to obtain references to administered objects. Administered objects are the JMS connection factory and JMS destination objects, where JMS destination objects are topics and queues. A system administrator creates and configures administered objects.

A JMS client specifies a JNDI InitialContext to obtain a JNDI connection to the JMS messaging server. The InitialContext is the starting point in any JNDI lookup and acts like the root of a file

system. The JMS directory service that is being used determines the properties that are used to create an InitialContext.

## IBM Integration Bus support for JMS

- Supports the JMS API, which provides message processing nodes for writing message flows to interact with JMS
- Provides a JMS Transport service, which connects applications that:
  - Receive a message as input from a JMS destination, at the beginning of a message flow or in the middle of a message flow
  - Create a message for output to a JMS destination
- Supports any JMS provider that is JMS 1.1 or 2.0 compliant, including IBM MQ and WebSphere Application Server
- Includes the following features:
  - JMS input and output nodes
  - JMS to and from IBM MQ transform nodes
  - JMS domains for message parsing and serialization
  - JMS activity logs
  - Syncpoint coordinated JMS messages as part of a message flow XA coordinated transaction

© Copyright IBM Corporation 2015

Figure 12-4. IBM Integration Bus support for JMS

WM666 / ZM6661.0

### Notes:

IBM Integration Bus supports the JMS version 1.1 and 2.0 standard. It uses the IBM Integration Bus JMS Transport service to connect to JMS messaging providers.

IBM Integration Bus also provides message processing nodes that are designed to send messages to and receive messages from JMS providers.



## JMS nodes

- Allow IBM Integration Bus to be a JMS client to a JMS provider for point-to-point and publish/subscribe messages
- Add brokering value to the JMS network by providing routing, and transforming point-to-point and publish/subscribe messages
- Simplify JMS message processing
- Connect the JMS network to an existing IBM MQ network
  - Inbound and outbound scenarios
  - Includes IBM MQ publish/subscribe messages
- Use configurable services to connect to JMS providers
- Transform a JMS message tree to and from an IBM MQ message tree

© Copyright IBM Corporation 2015

Figure 12-5. JMS nodes

WM666 / ZM6661.0

### Notes:

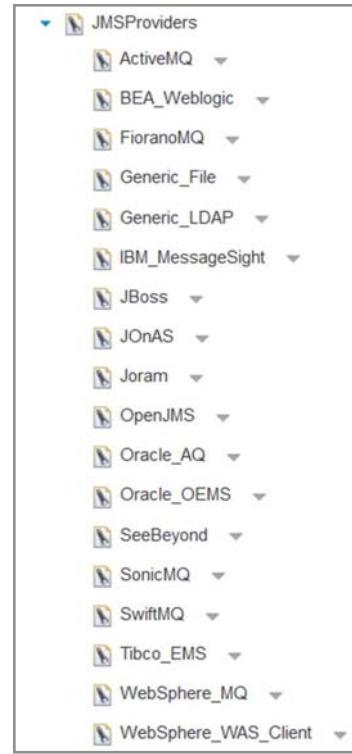
The JMS nodes can be used in applications in which messages are produced and used from various JMS destinations. In sending and receiving messages, the JMS nodes behave like JMS clients.

The JMS nodes work with the IBM MQ JMS provider, WebSphere Application Server, and any JMS provider that conforms to JMS V1.1 or 2.0.

The IBM Integration Bus configurable services define connections to the JMS providers.

## Making the JMS provider client available to the JMS nodes

- IBM Integration Bus configurable services are defined for many JMS providers
  - Choose one of the predefined services
  - Create a service for a new provider
- Use **`mqsicreateconfigurableservice`** command and IBM Integration web user interface to create a configurable service
  - **jarsURL** property identifies the location of the service JAR files on the integration node system



© Copyright IBM Corporation 2015

Figure 12-6. Making the JMS provider client available to the JMS nodes

WM666 / ZM6661.0

### Notes:

IBM Integration Bus contains predefined configurable services for many JMS providers that you can use as a template to create a configurable service for your JMS provider.

You can use the IBM Integration web user interface or the `mqsicreateconfigurableservice` command to create a configurable service.

## JMSInput node



- Receives messages from JMS destinations that are accessed through a connection to a JMS provider
- Receives and propagates messages with a JMS message tree
- When the JMSInput node receives publication topics, it internally restricts the message flow property **Additional Instances** to zero to prevent the receipt of duplicate publications

**JMS Input Node Properties - JMS Input**

Description	
Basic	JMS provider name*
<b>JMS Connection</b>	Client for WebSphere Application Server
Input Message Parsing	Initial context factory
Parser Options	com.ibm.websphere.naming.WsnInitialContextFactory
Message Selectors	Location JNDI bindings
Advanced	<The configurable service must provide a value if the nod
Validation	Connection factory name
Monitoring	<The configurable service must provide a value if the nod
	Backout destination
	Backout threshold
	0

© Copyright IBM Corporation 2015

Figure 12-7. JMSInput node

WM666 / ZM6661.0

### Notes:

Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

The JMSInput node acts as a JMS message consumer and can receive all six message types that are defined in the JMS specification. Messages are received by using method calls, which are described in the JMS specification.

The JMSInput node is contained in the **JMS** drawer of the palette.

The JMSInput node receives and propagates messages with a JMS message tree. You can set the properties of the JMSInput node to control how the JMS messages are received.

## Parsing the JMS message payload

- JMS domains can be used for modeling messages that are produced by the implementations of the JMS standard, version 1.1 or 2.0
  - Use the JMSMap domain when handling JMS messages of type MapMessage
  - Use the JMSStream domain when handling JMS messages of type StreamMessage
- How the message payload is extracted in JMS processing varies depending on the JMS message type
  - For message of JMS message type *BytesMessage*, *TextMessage*, and *ObjectMessage*, the payload is passed as a bit stream to an integration node parser
  - Payload for *MapMessage* and *StreamMessage* can be extracted only as individual elements

© Copyright IBM Corporation 2015

Figure 12-8. Parsing the JMS message payload

WM666 / ZM6661.0

### Notes:

The parsing process for the payload of JMS messages depends on the type of incoming JMS message.

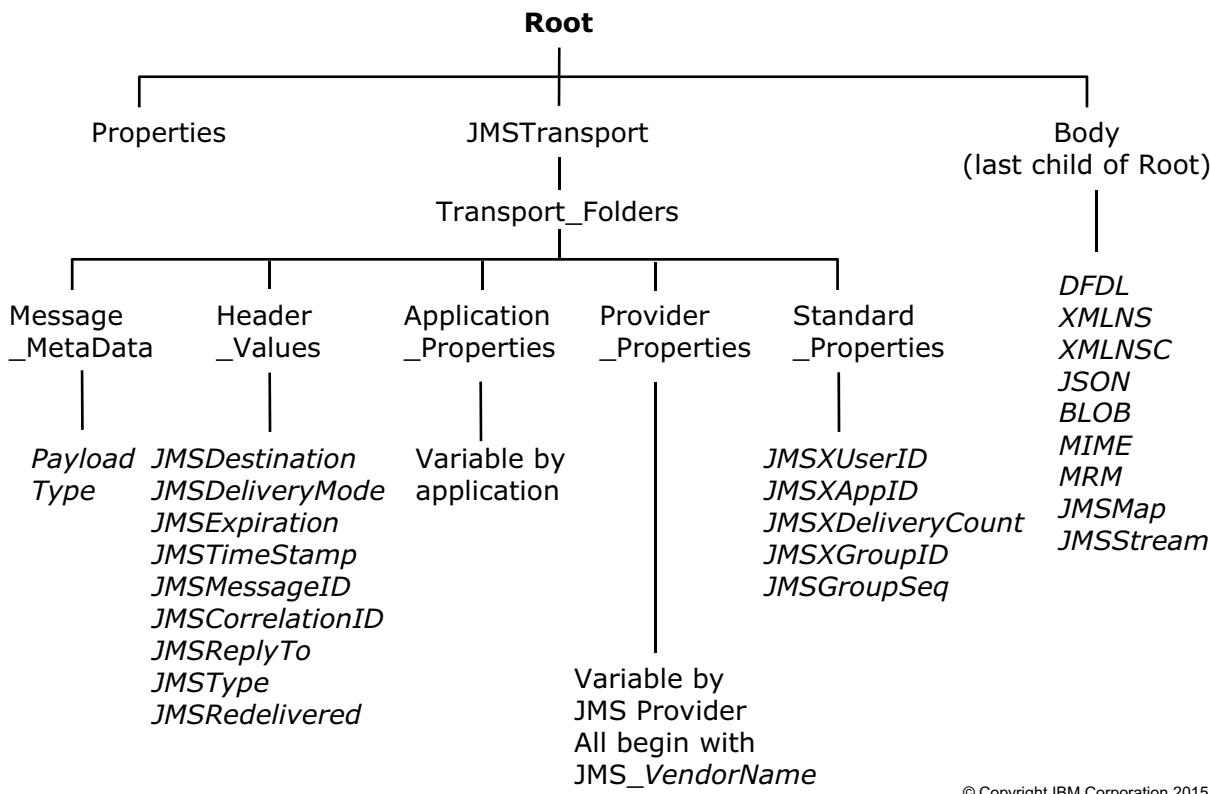
The payload for messages of the JMS message types *BytesMessage*, *TextMessage*, and *ObjectMessage* are extracted as a whole from the message object by using the JMS API. The payload is passed as a bit stream to an integration node parser.

For *ObjectMessage* types, the JMSInput node obtains the payload by calling `getObject()` on the message. The `getObject()` method returns a de-serialized object of the original class. This class definition must be made available to the JMSInput node, and you should ensure that it is accessible through the integration node Java class path. (For more information about how to specify the class path, see the IBM Knowledge Center.) The JMSInput node calls the BLOB parser, which creates the message body by using a bit stream that is created from the object. The Java Object can then be reserialized in a JavaCompute node (or a user-defined extension), and is updated by using its method calls.

The JMSInput node must reformat the payload for *MapMessage* and *StreamMessage* as follows:

- The JMSMap domain is a synonym for the integration node XML parser, which expects a stream of XML data. However, *MapMessage* payload data is extracted as sets of name-value pairs from the message object. The JMS API is used to obtain the name-value pairs. The JMSInput node appends each name-value pair to a bit stream as an XML element and value, and preserves the type of the value by using the `dt=` attribute. The map data is *not* returned from the JMS API in the order in which it was received.
- The *StreamMessage* payload data is a sequence of fields, where each field has a specific type. The fields do not have associated names and so a default element name `elt` is used to generate the XML elements. Like *MapMessage* messages, the JMS API allows for fields only to be retrieved individually. The JMSInput node extracts fields from the JMS message and appends each to a bit stream in XML format.

## JMS message tree



© Copyright IBM Corporation 2015

Figure 12-9. JMS message tree

WM666 / ZM6661.0

### Notes:

The JMS nodes use a special message tree that provides a JMS view of the message. With the standard *Properties* and *Body* folders, a JMS message contains a *JMSTransport* folder that contains information that is specific to the JMS message. It contains elements (tags) that are familiar to a JMS developer. Header values are stored as name-value pairs in the *Header\_Values* folder. The JMS message tree has the same form regardless of the JMS provider that the message flow uses.

The JMS message tree includes the following folders:

- *Message\_MetaData* contains information about the payload type of the JMS message. The *JMSOutput* node uses this folder.
- *Header\_Values* contains values for (or from) the JMS message headers.
- *Application\_Properties* is an optional folder that contains application-specific information. For example, it might message context information (for example, correlation ID and ReplyTo destination information) that is being preserved from the input message.
- *Provider\_Properties* can contain information that is specific to the JMS provider that is being used. The client can set these properties, or the provider can supply them automatically.

Provider-related properties are prefixed with *JMS\_* appended with the vendor name and the specific property name. For example, the IBM MQ JMS client sets the provider property to *JMS\_IBM\_MsgType*.

- The JMS provider sets *Standard\_Properties* when it sends a message. A JMS provider vendor can choose not to support any standard properties to support some standard properties, or to support all standard properties. Standard property names start with *JMSX*; for example: *JMSXUserid* or *JMSXDeliveryCount*.

The various JMS message processing nodes use some or all of these *JMSTransport* folders, depending on their operations.

## JMSOutput node



- Sends messages to JMS destinations
- Acts as a message producer
  - Sends a datagram message that carries sufficient information to be routed from the source to the destination computer
  - Sends a reply message that is routed according to the value in the **JMSReplyTo** property from the request message
  - Sends a request message to a JMS destination with the expectation of a response from the message consumer that processes the request
- If the message flow does not contain a JMSInput node, the MQJMSTransform node is included to transform the message to the format that the JMSOutput node expects

© Copyright IBM Corporation 2015

Figure 12-10. JMSOutput node

WM666 / ZM6661.0

### Notes:

The JMSOutput node acts as a JMS message producer, and can publish all six message types that are defined in the JMS specification. Messages are published by using method calls, which are described in the JMS specification.

The JMSOutput node is contained in the **JMS** drawer of the palette.

In the JMS message tree, the **PayloadType** field of the *Message\_MetaData* subfolder represents the JMS message type. To control the type of JMS message that the JMSOutput node creates, use ESQL code to set the **PayloadType** value, as shown in the following example:

```
SET OutputRoot.JMSTransport.Transport_Folders.Message_MetaData.PayloadType=
Payload value
```

## Serializing the JMS output message

- When a JMSOutput node receives a JMS message, it calls the JMSTransport parser to return an XML bit stream that contains the JMSTransport folder of the message assembly
- JMSOutput node obtains payload type information from the *Message\_MetaData* folder to determine which JMS message type to use for creating the output:
  - It populates the JMS header data from the JMS Header data in the XML string
  - It populates the output property values from the XML string, including the descriptors that define which Java Object type to create for each property value
- Message payload is obtained from the JMS message as a bit string, and calls the required JMS API methods to render the message fields

© Copyright IBM Corporation 2015

Figure 12-11. Serializing the JMS output message

WM666 / ZM6661.0

### Notes:

When a JMSOutput (or JMSReply) node receives a JMS message, it calls the JMS Transport parser to serialize the message assembly.

- **Header data:** The JMSOutput node extracts the JMS header data from the XML string, and uses this data to populate the values for the JMS header fields in the message.
- **Property data:** The JMSOutput node extracts the property values from the XML string. The XML elements contain type information that identifies which Java Object type to create for each property value.
- **Message payload data:** The message payload is obtained from the JMS message as a bit stream. For *TextMessage* and *BytesMessage* payloads, the bit stream can be passed to the JMS API directly to create the appropriate payload. For *MapMessage* and *StreamMessage* payloads, the individual elements must be extracted from the XML bit stream. The output node calls the appropriate JMS API method to create the map or stream fields in the message. For an *ObjectMessage* payload, the JMSOutput node reserializes the bit stream payload by using the object class.



## JMSReceive node

- Receive messages from JMS destinations
- Works like the JMSInput node, but can be used in the middle of a message flow
- Output message tree from a JMSReceive node combines the input tree (received on the **In** terminal) with the result tree from the received JMS message.
  - Set the **Result** properties on the JMSReceive node to specify the data from the input message to combine with the result tree
- Properties on the **Result** tab include:
  - **Output Data Location** specifies where in the output message tree to place the value that is specified in the *Result Data Location* message
  - **Result Data Location** specifies what part of the message tree to copy to *Output Data Location*

© Copyright IBM Corporation 2015

Figure 12-12. JMSReceive node

WM666 / ZM6661.0

### Notes:

The JMSReceive node works like the JMSInput node, but you can use the JMSReceive node in the middle of a message flow. The message assembly that comes into the node on the **In** terminal is merged with the message assembly that the node creates when it receives a message from the JMS destination. You control which data from the input message is combined with the result tree by configuring the **Result** property tab on the node.

The two main properties in the **Result** property tab that control the construction of the output tree are:

- **Output Data Location** specifies where in the output message tree to place the value that is specified in the **Result Data Location** message. The input root is first copied to the output root, and then the result data is copied to the location on the output tree that the **Output data location** property specifies. The default value, `$OutputRoot`, replaces the copied message tree with the data that is retrieved from the JMS provider; it does not propagate the input message. You can use an ESQL or XPath expression for this property.
- **Result Data Location** specifies what part of the message tree that was retrieved from the JMS provider to copy to the **Output Data Location**. The default, `$ResultRoot`, inserts the result

message into the output tree at the location that the **Output data location** property specifies. You can use an ESQL or XPath expression for this property.

For example, suppose that you set **Output Data Location** to \$OutputRoot/XMLNSC/JMSRecvMsg and you set **Result Data Location** to \$ResultRoot/JMSMap. After the JMSReceive node receives the message from the JMS provider, it copies the part of the message that \$ResultRoot/JMSMap defines to the \$OutputRoot/XMLNSC/JMSRecvMsg location in the message assembly.



## JMSReply node

- Sends a message to the JMS destination specified in the **JMSReplyTo** header field of the JMS message
- Similar to JMSOutput node, but send message only to **JMSReplyTo** destination
- Similar to the MQOutput node, output (**JMSReplyTo**) destination can be a list of destinations

© Copyright IBM Corporation 2015

Figure 12-13. JMSReply node

WM666 / ZM6661.0

### Notes:

The JMSReply node is similar to the JMSOutput node. The primary difference is that the JMSReply node sends a message only to the destination specified in the JMSReplyTo header. With the JMSOutput node, you can specify message destinations by using other values (including other locations that are specified in the message tree).

## JMS transformation nodes

- **JMSHeader** node sets or modifies the contents of the JMS *Header\_Values* and *Application\_properties* trees
- **JMSMQTransform** node transforms a message that contains a JMS message tree into a message that is compatible with the format of messages that the IBM MQ JMS provider produces
- **MQJMSTransform** node transforms a message that contains an IBM MQ JMS provider message tree into a format that can be used to send messages to JMS destinations



© Copyright IBM Corporation 2015

Figure 12-14. JMS transformation nodes

WM666 / ZM6661.0

### Notes:

You can use three "helper" nodes to accelerate development and reduce the amount of application code that you must write when working with JMS messages. By using these nodes, you can modify message header values without using a compute-type node to modify the values.

- With the **JMSHeader** node you can set, copy, or modify the *Header\_Values* properties in the JMS tree by using a set of fields. You can modify the properties by using predefined values, user-defined values, or XPath expressions. You can also set *Application\_Properties* in the JMS tree by adding, modify, or deleting name-value pairs of application properties.
- The **JMSMQTransform** node transforms a message that contains a JMS message tree to a message that is formatted to be compatible with the IBM MQ JMS provider.
- The **MQJMSTransform** node does the opposite. It transforms a message that is formatted for the IBM MQ JMS provider to a message that contains JMS headers that can be sent to JMS destinations.

For more information about the fields that the JMSMQTransform and MQJMSTransform nodes map, see the IBM Knowledge Center.

## JMS transactional processing

- You can use syncpoint coordination with message flows that receive JMS messages on a JMSInput node, or send messages to a JMS output with a JMSOutput node, as part of a message flow XA coordinated transaction
- JMS nodes must be connected to, and in a session with, a JMS provider
- Any message flow input node can inform the external syncpoint coordinator when a message flow transaction starts and ends, and whether to commit or roll back any resources that the message flow affects
- Syncpoint coordinator sends XA/Open compliant requests to all participating resource managers to inform them to prepare
  - Any changes are either committed or rolled back
  - Resource managers (such as IBM MQ, DB2, or any XA-compliant JMS provider) can participate in an XA coordinated transaction

© Copyright IBM Corporation 2015

Figure 12-15. JMS transactional processing

WM666 / ZM6661.0

### Notes:

By using of an XA-compliant transaction coordinator, you can enforce transactional control in a message flow that uses JMS.

The details of transactional control and how in-doubt transactions are recovered are specific to the transaction manager that is used. For more information, see the product documentation.

## Securing JMS message flows

- To enforce more security in JMS message flows, you can secure the JMS connections, the JNDI binding lookups, or both
- To secure JMS connections:
  - Specify the **Connection Factory Name** property on every node in the message flow that uses JMS transport
  - Use the `mqsisetdbparms` command to authorize the user ID and password for the specified connection factory

Example: `mqsisetdbparms Inodename -n jms::connectionFactoryName  
-u userid -p password`

- To secure JNDI connections:
  - Specify the **Initial Context Factory** property on every node in the message flow that uses JMS transport
  - Use the `mqsisetdbparms` command to authorize the user ID and password for the specified context factory

Example: `mqsisetdbparms Inodename -n jndi::initialContextFactoryName  
-u userid -p password`

© Copyright IBM Corporation 2015

Figure 12-16. Securing JMS message flows

WM666 / ZM6661.0

### Notes:

When you include JMS nodes in a message flow, you can secure the JMS connection resources and the JNDI binding lookups. To do so, you need to specify the appropriate properties on every JMS node in the message flow. You must also specify the user ID and password for the connection factory or initial context factory (or both) by using the `mqsisetdbparms` command.

## Web services overview

- A standard way to allow functions or methods to be called by using standard web protocols such as HTTP
- For interoperation between enterprises
- For interoperation within enterprises, for example, between channel servers and line-of-business servers
- Some transports can provide reliable messaging

© Copyright IBM Corporation 2015

Figure 12-17. Web services overview

WM666 / ZM6661.0

### Notes:

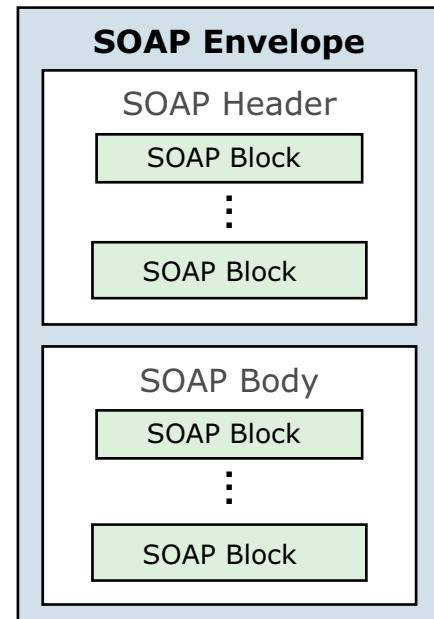
Web services provide a common program-to-program communications model, which is built on existing and emerging standards such as HTTP, XML, SOAP, WSDL, and Universal Description, Discovery, and Integration (UDDI).

A web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging. A web service is described by using a standard, formal XML notation, which is called its service description. It covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location. The interface hides the implementation details of the service. This interface allows it to be used independently of the hardware or software on which it is implemented, and also independently of the programming language in which it is written.

Independence allows and encourages web-services-based applications to be loosely coupled, component-oriented, cross-technology implementations. Web services fulfill a specific task or a set of tasks. They can be used alone or with other web services to do complex aggregation or a business transaction.

## SOAP

- XML-based protocol that consists of:
  - SOAP envelope that defines a framework for describing what is in a message and how to process it
  - SOAP header contains information for *any* SOAP receiver in the message path
  - SOAP body contains zero or more element information items that are targeted at the ultimate SOAP receiver
- SOAP requests can be sent by using many different transports such as HTTP and JMS



© Copyright IBM Corporation 2015

Figure 12-18. SOAP

WM666 / ZM6661.0

### Notes:

SOAP is an XML-based language for sending data between applications. SOAP is transport-neutral and platform-neutral. Not all web services use SOAP, but it is probably the most common message format for new web services.

A SOAP message contains an envelope with an optional header (containing one or more header blocks) and a mandatory body. A WSDL typically defines the content of the header and body.



## Web Services Description Language (WSDL)

- XML-based metadata document that defines a web service and how to access it
- Allows programs to discover services and call them without any administrative setup
- Provides functional and technical information
- Import WSDL files, and any referenced XML schema files, directly into the Integration Toolkit application or library
- Drag the WSDL to the Message Flow editor canvas to define and configure web services message flows and 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 2015

Figure 12-19. Web Services Description Language (WSDL)

WM666 / ZM6661.0

### Notes:

WSDL is an XML-based language for describing an interface between applications. A WSDL definition tells a client how to compose a web services request and describes the server-provided interface.

The WSDL document defines a *service* in terms of a number of *ports* (WSDL V1.1) or *endpoints* (WSDL V1.2). The ports define where the service is available.

Each named port also defines a mechanism (*binding*) for accessing it. There is a separate binding for each protocol that is supported, for example, SOAP over HTTP or SOAP over JMS. Each binding refers to a named *portType* (WSDL V1.1) or *interface* (WSDL V1.2).

A binding defines the message wire format and transport details. A *portType/interface* is the logical interface to the web service. Both binding and *portType/interface* define *operations*.

Each operation comprises *input* and *output* elements that are defined in terms of messages or message parts.

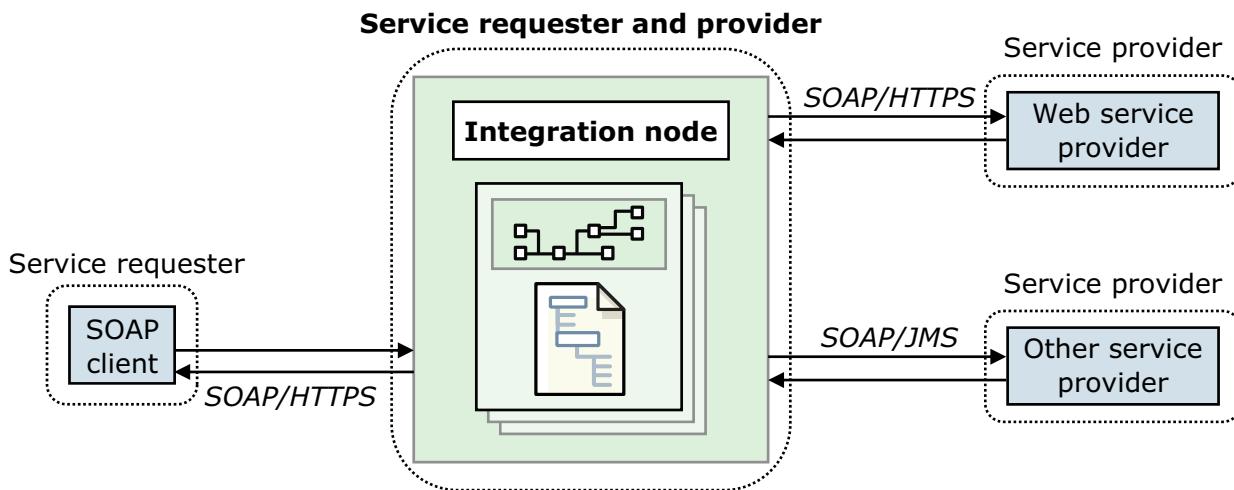
The *message* elements define a logical message in terms of one or more *parts*. (Each part might correspond to a parameter on a method call.)

A part is defined either as an XML schema element or as an XML schema type. Either or both of the following items fully define these elements and types:

- WSDL <types> section
- Imported schema (.xsd) files that provide the definitions for the WSDL part definitions

See the W3C at <http://www.w3.org> and in particular the WSDL V1.1 document at <http://www.w3.org/TR/wsdl>.

## IBM Integration Bus and web services



- IBM Integration Bus can:
  - Act as a front end to multiple service providers to allow for failover and provide a higher overall quality of service
  - Provide the logging of messages for an audit
  - Map SOAP messages to other industry standard formats
  - Select a service provider, basing the selection on the message context and content

© Copyright IBM Corporation 2015

Figure 12-20. IBM Integration Bus and web services

WM666 / ZM6661.0

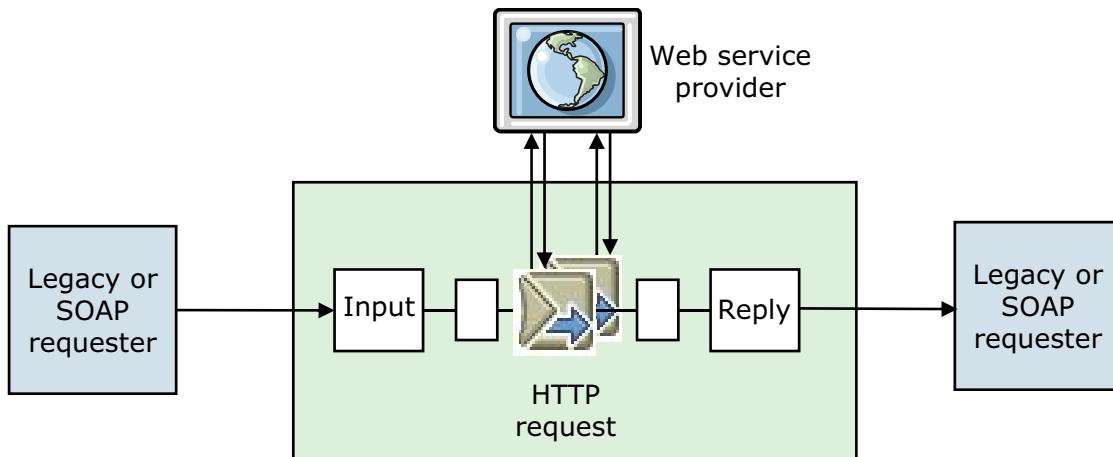
### Notes:

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

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

## Message flow as HTTP/HTTPS client

- Legacy clients that access web services
- IBM Integration Bus as a buffer between a changing set of service providers (web services and other)
- Aggregation of web services



© Copyright IBM Corporation 2015

Figure 12-21. Message flow as HTTP/HTTPS client

WM666 / ZM6661.0

### Notes:

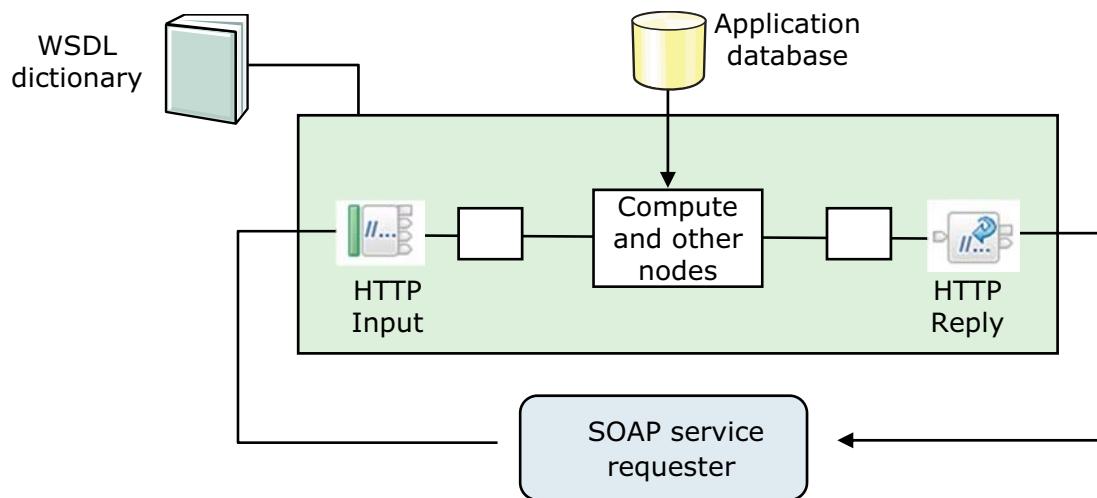
The **HTTPRequest** node can interact with a web service by using all or part of the input message as the request sent to that service. The node can also be configured to create an output message from the contents of the input message. You can augment the output message with the contents of the web service response before the message is propagated to subsequent nodes in the message flow.

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

The **Default web service URL** property can be set on the **HTTPRequest** node to determine the destination URL for a web service request. A **Compute** node can be included before the **HTTPRequest** node within the message flow to override the value that is set in the property.

## Message flow as service provider

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



© Copyright IBM Corporation 2015

Figure 12-22. Message flow as service provider

WM666 / ZM6661.0

### 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 retrieved. 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 after further possible transformation or aggregation.



## IBM Integration Bus and web services configuration

- Message flows
  - Can be exposed as web services
  - Can start web services
- WSDL
  - Can be generated from a message model with namespaces
  - Can be imported directly as an application or library
  - Can be used to automatically generate many artifacts that are used in a message flow that supports web services
- HTTP and SOAP nodes
- SOAP parser

© Copyright IBM Corporation 2015

Figure 12-23. IBM Integration Bus and web services configuration

WM666 / ZM6661.0

### Notes:

A message flow can be a requester or a client. An input node drives a flow that calls a web service (over HTTP or JMS bindings). The result of the call is used to augment the message that is sent back by using an output node.

A message flow can be a service provider, allowing web service clients to call an existing message flow.

The web service that the WSDL defines can be directly imported into the IBM Integration Toolkit and used to generate message flows.

IBM Integration Bus includes nodes for HTTP and SOAP. It also includes a parser for SOAP messages.

## HTTP nodes

- **HTTPInput node**
  - Web service URL
  - Uses HTTPS
  - Maximum time a client waits for an HTTPReply; on timeout send a SOAP fault message
  - HTTP Proxy location
  - Follow HTTP redirection
- **HTTPReply node**
  - Reply send timeout for an acknowledgment from the client
  - Generate default HTTP headers from the HTTPResponseHeader of an input message
- **HTTPRequest node**
  - Web service URL
  - Request a timeout for web services replies (default = 120 seconds)
  - Use `InputBody` as a request, or a path such as `InputRoot.x`
  - Use a web service reply as `OutputBody`, or a path such as `OutputRoot.x`
  - Generate default HTTP headers from the input

© Copyright IBM Corporation 2015

Figure 12-24. HTTP nodes

WM666 / ZM6661.0

### Notes:

When a message is received from a web service client or a web server, the HTTPInput or HTTPRequest node that receives that 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 bit stream.

The HTTPInput and HTTPRequest nodes normally create HTTPInput and HTTPResponse headers from the original client/server bit stream.

The developer normally sets the HTTPReply and HTTPRequest headers in a Compute node, or by setting a default on the properties of the HTTPReply or HTTPRequest node.

Remember to add appropriate MQMD headers when a message that originates from HTTPInput node is routed to the MQOutput node.



## SOAP nodes

- When a message flow is the web services provider
  - SOAP Input
  - SOAP Reply
- When a message flow is the web services consumer
  - SOAP Request
  - SOAP AsyncRequest and SOAP AsyncResponse
- SOAP Extract node
  - To remove SOAP envelopes, allowing just the body of a SOAP message to be processed
  - To route a SOAP message based on its operation name
- SOAP Envelope node
  - To add a SOAP envelope onto an existing message
  - Used with the SOAPExtract node
- Nodes support WS-Addressing and WS-Security

© Copyright IBM Corporation 2015

Figure 12-25. SOAP nodes

WM666 / ZM6661.0

### Notes:

SOAP nodes process SOAP messages directly. For the consumer scenario, both synchronous and asynchronous capability is supported allowing 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 over JMS

- JMS can be used as a transport for SOAP message
- SOAP over Java Message Service 1.0 specification describes how SOAP can bind to a messaging system that supports JMS
- SOAP is transport-independent and can be bound to any protocol
  - Both SOAP 1.1 and SOAP 1.2 can be bound in this way by using the SOAP 1.2 Protocol Binding Framework
- IBM Integration Bus supports the JMS transport on the SOAP nodes, which are configured by importing WSDL with SOAP/JMS bindings

© Copyright IBM Corporation 2015

Figure 12-26. SOAP over JMS

WM666 / ZM6661.0

### Notes:

You can use SOAP over JMS to use the JMS transport with SOAP messages, and is an alternative messaging mechanism to the standard SOAP over HTTP messaging.

JMS applications send messages to JMS destinations, which can be queues or topics. Queues are used in point-to-point messaging, and topics are used in publish/subscribe messaging.

## Unit summary

Having completed this unit, you should be able to:

- Describe how to use message flow applications with JMS
- Describe how message flow applications can support Hypertext Transfer Protocol (HTTP) and SOAP messages
- Explain how the Web Services Definition Language (WSDL) file is used to develop web services message flows

© Copyright IBM Corporation 2015

Figure 12-27. Unit summary

WM666 / ZM6661.0

### Notes:



## Checkpoint questions

1. True or False: If a message flow includes a JMSInput node, you must also include a JMSOutput node.
2. True or False: IBM MQ can act as a JMS provider.
3. True or False: A message flow can be both a provider of web services and a consumer of web services.

© Copyright IBM Corporation 2015

Figure 12-28. Checkpoint questions

WM666 / ZM6661.0

### Notes:

Write your answers here:

- 1.
- 2.
- 3.

## Checkpoint answers

1. True or False: If a message flow includes a JMSInput node, you must also include a JMSOutput node.

**Answer: False.** It is not necessary to have a JMS Output node with a message flow that contains a JMSInput node. For example, a message flow can convert a JMS message to an IBM MQ message. In this example, the message flow would contain an IBM MQOutput node.

2. True or False: IBM MQ can act as a JMS provider.

**Answer: True.**

3. True or False: A message flow can be both a provider of web services and a consumer of web services.

**Answer: True.**

© Copyright IBM Corporation 2015

Figure 12-29. Checkpoint answers

WM666 / ZM6661.0

### Notes:



# Unit 13. Preparing for production

## What this unit is about

In this unit, you learn how to expand the capabilities of message flow applications by making them aware of the runtime environments in which they operate. You also learn techniques for implementing dynamic message routing at run time, adding monitoring and auditing, and controlling processing of message flows with applications and shared libraries.

## What you should be able to do

After completing this unit, you should be able to:

- Deploy applications and shared libraries at run time to affect the visibility of resources
- Use promoted properties, user-defined properties, and operational policies to develop environment-aware message flows
- Dynamically route messages in a message flow by using external registries and registry lookup nodes to allow policy-driven message flows to meet governance requirements
- Add monitoring and auditing to a message flow
- Perform basic performance analysis on message flows

## How you will check your progress

- Checkpoint questions
- Lab exercise

## References

IBM Knowledge Center

## Unit objectives

After completing this unit, you should be able to:

- Deploy applications and shared libraries at run time to affect the visibility of resources
- Use promoted properties, user-defined properties, and operational policies to develop environment-aware message flows
- Dynamically route messages in a message flow by using external registries and registry lookup nodes to allow policy-driven message flows to meet governance requirements
- Add monitoring and auditing to a message flow
- Perform basic performance analysis on message flows

© Copyright IBM Corporation 2015

Figure 13-1. Unit objectives

WM666 / ZM6661.0

### Notes:

By now you have an understanding of the capabilities that message flows provide. In this unit, you learn how message flows are even more powerful and flexible at run time.

## Applications and libraries at run time

- How you deploy applications and libraries affects the visibility of those resources at run time
- Applications promote encapsulation and isolation
  - Multiple applications can be packaged into a single BAR
  - Multiple applications can be deployed to an integration server
  - Visibility of a resource is restricted to the containing application
  - Static libraries are deployed inside applications
- Shared and static libraries facilitate reuse and simplify resource management
  - Static libraries are packaged as part of referencing application in the BAR
  - Shared libraries are deployed separately before the referencing applications

© Copyright IBM Corporation 2015

Figure 13-2. Applications and libraries at run time

WM666 / ZM6661.0

### Notes:

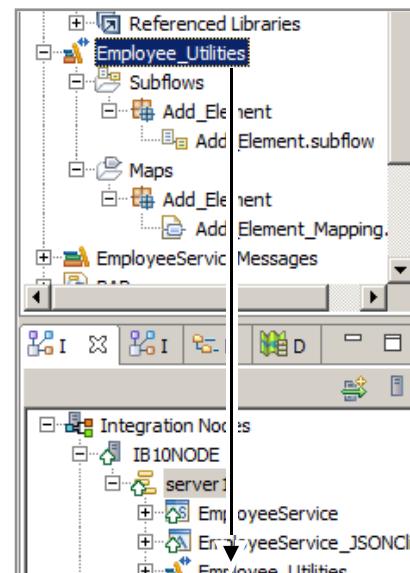
In IBM Integration Bus, you store development artifacts in containers that are known as *applications* and *libraries*.

An application provides runtime isolation so that the visibility of a resource is restricted to the application in which it is contained. These resources include libraries that an application references.

Libraries can be static or shared. Static libraries are packaged with the application. Shared libraries are deployed and maintained separately.

## Applications and libraries on the integration node

- Each application is an independent container within the integration server
- Static libraries are listed as a child of the referencing application container
- Shared libraries are deployed and maintained separately
- All applications and libraries are stored as unmodified files on the work path of the target integration node



© Copyright IBM Corporation 2015

Figure 13-3. Applications and libraries on the integration node

WM666 / ZM6661.0

### Notes:

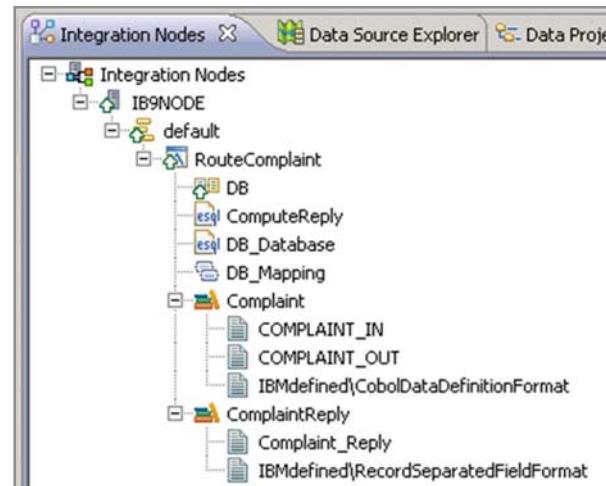
Applications and libraries are deployable containers of resources, such as message flows, subflows, message models (DFDL, XSD files), JAR files, and XSL stylesheets.

How you deploy applications and libraries affects the visibility of those resources at run time.

You can deploy multiple applications to an integration server. These applications can optionally contain static libraries.

## Resource isolation with applications

- Use applications to group flows and supporting resources so that updates to one group do not affect other groups
- When one or more applications references a static library, a copy of the library is deployed to each application container that references that library, which provides a certain level of isolation
- When an update to a static library is deployed to an application, it flows only inside the affected application area



© Copyright IBM Corporation 2015

Figure 13-4. Resource isolation with applications

WM666 / ZM6661.0

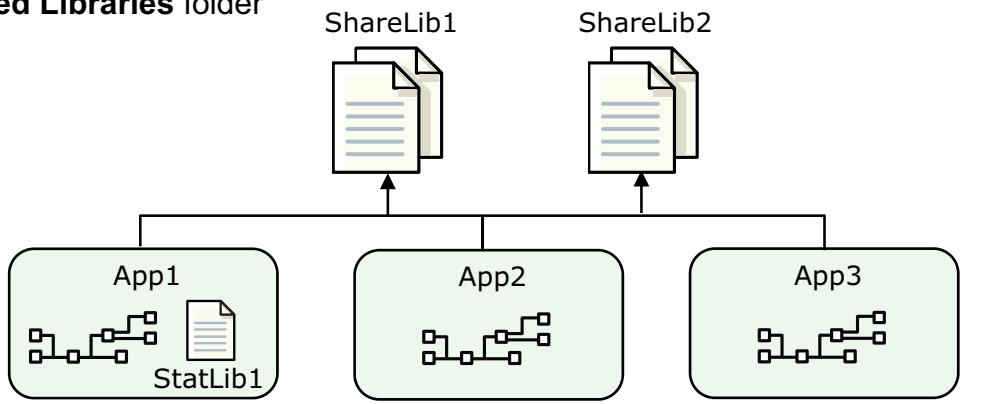
### Notes:

You use applications at design time and run time to provide isolation so that update to one group of flows does not affect another group.

If the application references a static library, a copy of the library is deployed with the application so that changes to the library are not propagated to the library that is referenced in the application.

## Shared libraries at run time

- Applications can reference one or more shared libraries
- When deploying an application, if a required shared library is not already deployed or currently being deployed then the deployment fails
- Applications that reference the shared library automatically recognize updates to the deployed shared library
- It is not possible to remove a shared library from the integration node when deployed applications are currently referencing it
- Shared libraries can be viewed in IBM Integration web user interface under the **Shared Libraries** folder



© Copyright IBM Corporation 2015

Figure 13-5. Shared libraries at run time

WM666 / ZM6661.0

### Notes:

A shared library can be deployed directly to an integration server. Any application can reference the resources in that deployed shared library. If that shared library is updated, all referencing applications automatically recognize the changes.

A shared library must be deployed with or before an application that references it.

- If you deploy resources by dragging them onto an integration server, deploy the shared library first.
- If you deploy resources by adding them to a BAR file, ensure that you include any shared libraries that your application references.

You cannot remove a deployed shared library while deployed applications are referencing it.

## Deploying updates to shared libraries

- When an update is deployed to a shared library that is in use by one or more applications, those applications are instantly affected
- 1. At the start of deployment processing, all flows in all applications that use the shared libraries are locked and unable to process messages
- 2. Shared library resources are updated on the integration node
- 3. Applications are validated to ensure that the shared library update did not break them
  - If applications fail, the deployment of the library is rolled back and rejected
  - If the applications pass validation, all previously locked flows are unlocked and they immediately use the updated resources from the updated shared libraries
- Use `mqsilist` command with `-y` option to list the applications that reference a library

Example: `mqsilist IBNODE -e default -y ShareLib1`

© Copyright IBM Corporation 2015

Figure 13-6. Deploying updates to shared libraries

WM666 / ZM6661.0

### Notes:

The figure describes the events that occur when a shared library is updated and deployed.

You can view the resources that are deployed in a shared library by using the `mqsilist` command.



## Creating BAR files

- With IBM Integration Toolkit
- With `mqsispackagebar` command
- With `mqsicreatebar` command

© Copyright IBM Corporation 2015

Figure 13-7. Creating BAR files

WM666 / ZM6661.0

### Notes:

You can create BAR files by using the IBM Integration Toolkit, the `mqsispackagebar` command, and the `mqsicreatebar` command.



## mqsipackagebar command

- Focuses on BAR *packaging* rather than *compiling*
    - Deploy IBM Integration Bus assets as source
    - Specify applications, libraries, and files to add to the BAR
    - Compile any message sets or Java code before you create the BAR file
  - Complements IBM Integration Toolkit and Integration API deployment
  - On z/OS, run this command by customizing and submitting BIPPACK
  - Available on all integration node hardware and operating systems
- Example:

```
mqsipackagebar -a MY.BAR -o my.msgflow my.dfdl my.esql
```

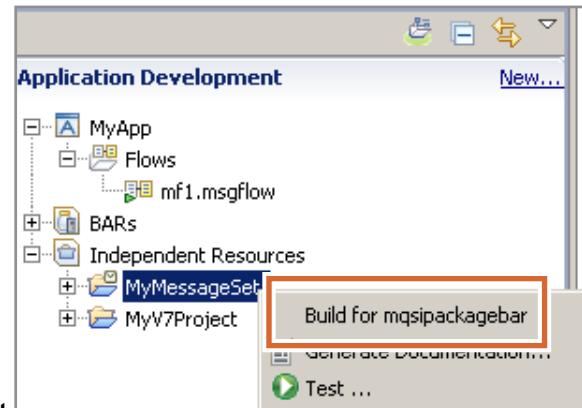


Figure 13-8. mqsipackagebar command

© Copyright IBM Corporation 2015

WM666 / ZM6661.0

### Notes:

In a development environment, you use the IBM Integration Toolkit to create a BAR file. Access to the IBM Integration Toolkit might not be available in a runtime environment, so a command option can be used to create the BAR file.

You can use the `mqsipackagebar` command to create deployable BAR files on servers that do not have the IBM Integration Toolkit installed.

Resources that you add to a BAR file by using the `mqsipackagebar` command are not compiled when they are added. To deploy a BAR file that you create by using this command, you must add deployable resources to the BAR file. For example, if you want to include Java code in your BAR file, you must first compile the file before you use the `mqsipackagebar` command to add them to your BAR file.

If you use a repository to store your message flows and models, you can write scripts that use commands and command tools that are available in the repository to deploy your message flow applications.

## mqsicreatebar command

- Create deployable BAR files that contain message flows and dictionaries
- Specify projects, applications, and libraries
- On Linux, the user ID must have write access to the workspace (**-data**) and BAR file location (**-b**) directories

Example:

```
mqsicreatebar -data C:\Workspace -b C:\Bars\appl.bar -a appl
```

© Copyright IBM Corporation 2015

Figure 13-9. mqsicreatebar command

WM666 / ZM6661.0

### Notes:

You can use the **mqsicreatebar** command to create deployable BAR files that contain message flows and dictionaries.

The example creates a BAR file that is named `appl.bar` in the `C:\Bars` directory. The BAR file includes application that is named `appl` that is in the `C:\Workspace` directory.

## Integration node properties

- Each integration node maintains a set of attributes that describe it:
  - Node-related such as `NodeLabel`, `Transaction`, and `DataSource`
  - Message-flow-related such as `MessageFlowLabel` and `AdditionalInstances`
  - Integration server-related such as `IntegrationServerName`, `IntegrationNodeName`, and `Family`
- Useful, at run time, to have real-time access to details of a specific node, flow, or integration
- Access some of these properties from ESQL programs, Mapping node, and Java programs
  - ESQL and Java access methods use NULL for unset values
- Subset of properties available from the JavaCompute nodes
  - Accessible through the classes `MbExecutionGroup` (for integration server), `MbNode`, and `MbBroker` (for integration node)

© Copyright IBM Corporation 2015

Figure 13-10. Integration node properties

WM666 / ZM6661.0

### Notes:

A message flow is said to be “environmentally aware” if it takes advantage of the integration node properties. For example, if a message flow routes messages differently when it is running in a production environment than it does when it is running in a development environment, then it is environmentally aware.

IBM Integration Bus retains a set of runtime properties for every integration node that you create. It can be useful to have real-time access to details of a specific node, flow, or integration node at the run time of your code.

At run time, IBM Integration Bus properties are constants: you cannot assign things to them, so SET statements cannot change their values. If a program tries to change the value of an IBM Integration Bus attribute, the following error message is sent: Cannot assign to a symbolic constant.

If your ESQL code already contains a variable with the same name as one of the IBM Integration Bus properties, your variable takes precedence, that is, your variable masks the IBM Integration Bus attribute.

To access the IBM Integration Bus attribute, use the form `SQL.<broker_attribute_name>`, for example: `SQL.BrokerName`

For more information, see the IBM Knowledge Center topic “List of integration node properties accessible from ESQL and Java.”

## Configuring for production

- Deployment descriptor similar to environment variables
  - No change to the source code
  - Customize databases and queue names for the target system (development, test, or production)

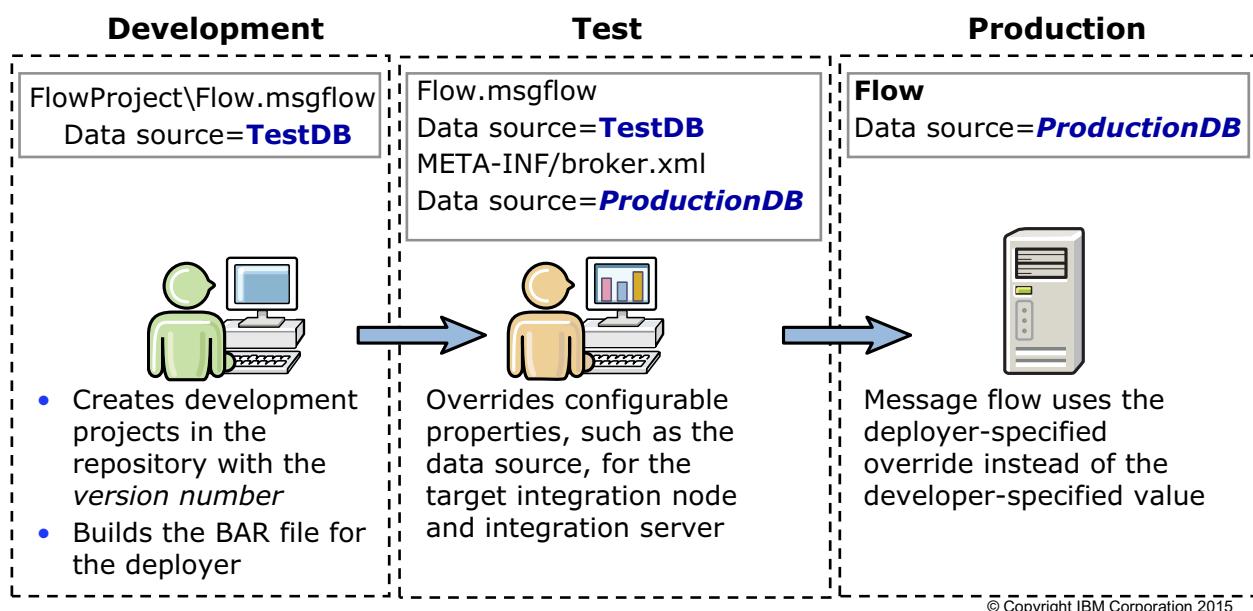


Figure 13-11. Configuring for production

WM666 / ZM6661.0

### Notes:

There are three primary phases for developing IBM Integration Bus message flows.

- In the Development phase, the message flow is developed and tested in an isolated environment. The data sources and targets, such as queues and databases are typically limited in scope and in many cases contain only a subset of the actual data. In this phase, the message flow typically uses the resources that are configured in the node properties.

A version number might be used to track the version of the code. The external source code management system or the `mqsicreatebar` command can add the version number. As an alternative to adding version information to the name of a resource, you can use the **Version** property of message flows and message models.

- In the Test phase, the message flow should be deployed to a system that exactly duplicates the Production system. During this phase, real data should be used to determine the performance of the flow such as the throughput and the number of instances that are required. In this phase, the message flow should point to the Test environment resources.
- In the Production phase, the message flow uses the production resources that override the resources that are specified during development. The resource values can be specified in the

BAR file editor or when the BAR file is deployed by using the `mqsiapplybaroverride` command.

## Environment-dependent message flows

- Node properties
  - Message Flow editor configures them in the message flow
  - Can be propagated to a subflow
  - BAR editor can override certain environment-dependent node properties in BAR
  - At run time, the actual value can be queried through IBM Integration Bus attributes in ESQL or Java code
  
- User-defined properties (UDP)
  - Message Flow editor declares initial value for these constants (default value) in Message Flow properties and in ESQL (`EXTERNAL variables`) or Java
  - BAR editor can override in BAR
  - At run time, actual value can be queried through ESQL variables or the `JavaCompute` node (`getUserDefinedAttribute` method)

© Copyright IBM Corporation 2015

Figure 13-12. Environment-dependent message flows

WM666 / ZM6661.0

### Notes:

Supplied node properties or user-defined properties can hold configuration data. You can configure a message flow for a particular operating system, task, or environment at deployment time without needing to change the implementation.

Use UDPs when IBM Integration Bus attributes do not supply that information. Typical uses of UDPs include specifying configuration information or maintaining version information.

A UDP is a user-defined constant that you can use in your ESQL or Java programs. Optionally, you can give the UDP an initial value when you declare it to your program. This initial value can be modified at design time by the developer, or overridden at deployment time by the deployer. At run time, subsequent program statements can query but not modify the value of the UDP after it is declared.

You can define a user-defined property in the following ways:

- In the ESQL code
- In the Message Flow editor
- Through a BAR file override, before the BAR file is deployed
- By using the Integration API

There is a precedence of overriding user-defined properties: Overriding the BAR file takes precedence over changes you make with the Message Flow editor. Changes that are made by using in the Message Flow editor take precedence over changes the ESQL code makes to UDPs.

## Example: UDP determines route in flow

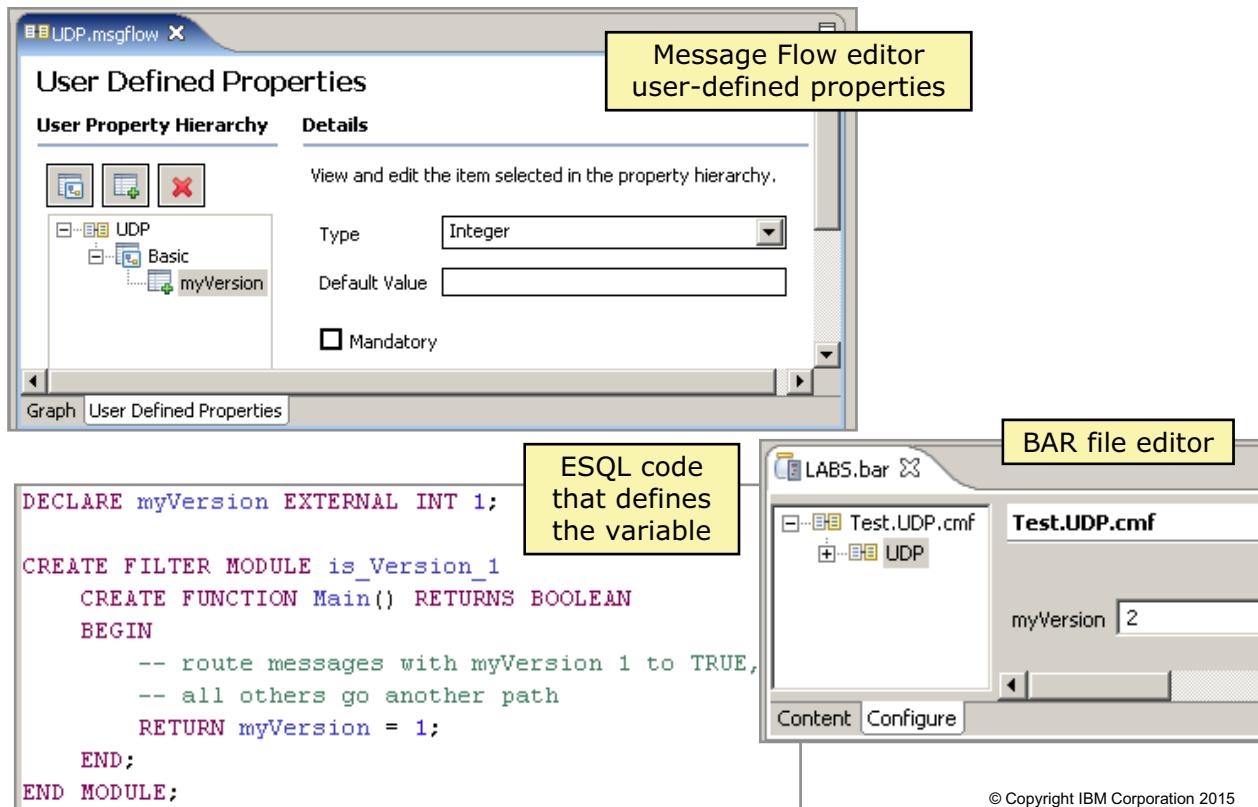


Figure 13-13. Example: UDP determines route in flow

WM666 / ZM6661.0

### Notes:

This example shows how UDPs can be used to route the message in a Filter node through alternative paths in the flow, depending on the BAR configuration.

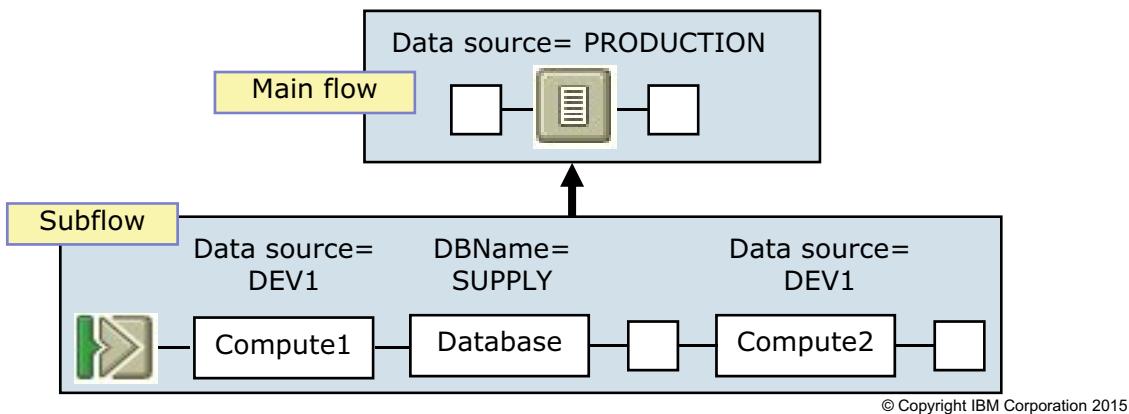
1. You declare a user-defined property that is named `myVersion` in the message flow.
2. In the ESQL in the Filter node, you declare the EXTERNAL variable `myVersion` to correspond with the user-defined property by the same name. The variable is initialized to 1 in the `DECLARE` statement.

The ESQL in the Filter node returns TRUE or FALSE depending on whether `myVersion` is equal to 1. Recall that this result determines which path the message is propagated through.

3. Like all user-defined properties, `myVersion` is available in the BAR file. Before deployment, you access the user-defined properties in the BAR File editor. In this example, `myVersion` is set to 2. The BAR file override takes precedence over the message flow editor and the ESQL. The message is routed through the **False** terminal of the Filter node because 1 does not equal 2 in the ESQL code test in the module.

## Promoted properties

- Message flow node property that is promoted to the level of the message flow in which it is included
- In the subflow:
  - Promote properties
  - Compatible properties from more than one node can be promoted to the same promoted property
- In the main flow, set (promoted) properties as the node properties of a subflow



© Copyright IBM Corporation 2015

Figure 13-14. Promoted properties

WM666 / ZM6661.0

### Notes:

In IBM Integration Bus, you can group similar properties. You can then “promote” the ability to alter their values to a higher level.

In the example, the two Compute nodes in the subflow each have a **Data source** property (system defined). The main flow that calls the subflow also has a **Data source** property.

You can define the **Data source** property in a subflow to be promoted to the message flow level. At that point, whenever you change the value of data source in the main flow, the new value is propagated to the subflow. So, in this example, even though the data source in the subflow is set to **DEV1**, setting data source in the main flow to **PRODUCTION** propagates that same value to the subflow.

When you mark a property as promoted, you cannot change its value at the lowest level of the declarations. In this example, that means you cannot change the value of **Data source** in the subflow.

To promote message flow node properties to the message flow level:

1. Open the message flow for which you want to promote properties in the IBM Integration Toolkit Application Development perspective.

2. Right-click the appropriate node and then click **Promote Property**. The **Available node properties** pane lists all available properties for all the nodes in the message flow. The properties for the node that you clicked are expanded.
3. Select the property or properties that you want to promote to the message flow. You can select multiple properties by holding down Ctrl and selecting the properties.
4. Click **Promote**. The **Target Selection** dialog box opens and displays valid targets for the promotion.
5. Select the destination group or property for the properties that you want to promote.
6. Click **OK** to confirm your selections and close the **Target Selection** dialog box.



## Runtime version information in IBM Integration Bus

- Version and key information can be defined when developing a message set
  - Define a default version in the default **Version** tag of the message set preferences
  - Define keywords in the **Documentation** property of the message set
- At deployment, a BAR file contains version information
- Displayed in IBM Integration Toolkit and Integration API applications

Property	Value
Info	
Additional Instances	0
Commit Count	1
Commit Interval	0
Coordinated Transaction	false
Deployment Time	October 21, 2005 10:54:13 PM CEST
Full Name	LAB9.cmf
Last Modified	October 21, 2005 10:54:06 PM CEST
Version	1.1
Keywords	
BAR File Name	C:\Workspaces\MQ666\LocalProject\LABS.bar
ErrorHandler Author	student
ErrorHandler Version	1.0

© Copyright IBM Corporation 2015

Figure 13-15. Runtime version information in IBM Integration Bus

WM666 / ZM6661.0

### Notes:

At deployment time, various pieces of version information about each file that is deployed inside the BAR file are stored. This information includes:

- The time the object was deployed.
- The time the object was last saved in the BAR file.
- The name of the BAR file that deployed the object.
- A user-specified **Version** keyword.
- Other user-defined keywords and their values. In this example, the user-defined keywords are **ErrorHandler Author** and **ErrorHandler Version**.

## Embedding version information in message flows

- Version is a property of a message flow
- For more information, embed **\$MQSI..MQSI\$** in unused fields
  - Main flows: Long description of a message flow

```
$MQSI Control flow version = v1.2 MQSI$
$MQSI Processing subflow version = v1.9 MQSI$
$MQSI Database subflow version = v1.8 MQSI$
```

- ESQL comments or strings

```
-- $MQSI Compiled by = Pat MQSI$
Set target = '$MQSI target = production only MQSI$'
```

- For subflows, use in **String** field of a Passthrough node
  - Use a different version keyword in each subflow

© Copyright IBM Corporation 2015

Figure 13-16. Embedding version information in message flows

WM666 / ZM6661.0

### Notes:

The IBM Integration Toolkit automatically embeds the Version keyword in message flows and dictionaries if a version property is entered in the BAR File editor.

Because there is only one version keyword value for the main flow, you must insert user keywords if you want to show subflow versions.



## Runtime version information: Embedding keywords

- XML files

**\$MQSI keyword = value MQSI\$**

Embedded in the file

- ZIP files

**\$MQSI keyword = value MQSI\$**

Embedded in the ZIP entry

(META-INF\keywords.txt)

- Not all editors contain IBM Integration Toolkit support for version information

	File format
Compiled message flows (.cmf)	XML
Message dictionaries (.dictionary)	XML
XSL stylesheets (.xsl, .xslt, and .xml)	XML
Java archives (.jar)	ZIP
XSD archives (.xsdzip)	ZIP
Inbound adapter configuration (.inadapter)	ZIP
Outbound adapter configuration (.outadapter)	ZIP

© Copyright IBM Corporation 2015

Figure 13-17. Runtime version information: Embedding keywords

WM666 / ZM6661.0

### Notes:

Many IBM Integration Bus objects can have extra information added to the object. This capability can provide information about an object after the object is deployed.

You can define custom keywords and their values. IBM Integration Bus interprets these as extra information that is displayed in the properties view.

Any keywords that you define must follow certain rules to ensure that the information can be parsed. In XML-based files, such as compiled message flows, these keyword definitions are embedded in the file. For ZIP-based files, such as JAR files within the BAR file, the keyword definitions are embedded in the `keywords.txt` file of the JAR file, which is under the `META-INF` directory.

## Linking with version control systems

- Many version control systems, such as CVS, use keyword systems that allow information to be expanded when files are checked out

Example:

- Concurrent Version System (CVS) or Revision Control System (RCS):
 

`$Revision: $` expands to `$Revision: 1.3 $`
- Source Code Control System (SCCS):
 

`%I%` expands to 1.3

- Use these tags inside `$MQSI..MQSI$` to conveniently embed source code repository information

```
$MQSI Version = %I% MQSI$  
$MQSI Revision = $Revision: $ MQSI$
```

- When files that contain these strings are checked out, the version control system resolves their values

© Copyright IBM Corporation 2015

Figure 13-18. Linking with version control systems

WM666 / ZM6661.0

### Notes:

You can take the version numbers of resources from the repository, and associate that version number with the message flows when they are deployed. With this association, you can display which version of the flow is deployed in the IBM Integration Toolkit.

An alternative method of assigning a version number to a message flow is to specify one in the IBM Integration Toolkit when the message flow is created.



## Accounting and statistics

- Set at integration server or message flow level
- Real-time flow, node, terminal, and thread usage information
  - Default level (message flow) has a minimal impact on performance
  - Terminal level generates a high volume of data, which increases in processor usage
- Short interval snapshot (about 20 seconds) and longer interval archive collection
- Statistics invocation and control are noninvasive by integration node commands
- Data can be written to various destinations
  - Published in XML under a topic when publication of events is enabled and a publish/subscribe is configured with IBM MQ or MQTT
  - UserTrace and System Management Facilities (SMF)

© Copyright IBM Corporation 2015

Figure 13-19. Accounting and statistics

WM666 / ZM6661.0

### Notes:

IBM Integration Bus includes event monitoring and auditing, which allow insight into the messages as they are being processed in a message flow.

Message flow accounting and statistics data records dynamic information about the runtime behavior of a message flow. For example, it indicates how many messages are processed and how large those messages are, and processor usage and elapsed processing times.

In IBM Integration Bus, an operator can capture the content of any message or part of any message at any time in the message flow as the message flows through the integration node. The content can be captured without changing the flow in any way.

## Statistics data details

Message flow	Threads
<ul style="list-style-type: none"> <li>• Integration server name</li> <li>• Integration node name</li> <li>• Flow name</li> <li>• Sample time, start, and end</li> <li>• "Archive" or "Snapshot"</li> <li>• Message: Size(MMA*), number, and bytes that are processed</li> <li>• Times(MMA*): Elapsed, CPU, and Wait</li> <li>• Time: Waiting for input</li> <li>• Threads: Total, or high water count</li> <li>• Commit or backout count</li> </ul>	<ul style="list-style-type: none"> <li>• Arbitrary thread number</li> <li>• Message: Total processed</li> <li>• Total Times: Elapsed, CPU, Wait, and Input Wait</li> <li>• Message: Min, and Max input size</li> </ul>
Nodes	
	<ul style="list-style-type: none"> <li>• Label name</li> <li>• Type</li> <li>• Times(MMA*): Elapsed, CPU, and Wait</li> <li>• Invocation count</li> <li>• Message: Size(MMA*), number, and bytes that are processed</li> <li>• Terminal labels and counts</li> </ul>

\* MMA is Minimum, Maximum, Average

© Copyright IBM Corporation 2015

Figure 13-20. Statistics data details

WM666 / ZM6661.0

### Notes:

Accounting and statistics data is associated with an accounting origin. If you want to track the behavior of a particular message flow, you can set a unique accounting origin for this message flow, and analyze its activity over a specified period. To do so, you must set the `Environment.Broker.Accounting.Origin` field in the message flow (or flows) that you want to track. You set this field in a Compute, Database, or Filter node that is in the message flow.

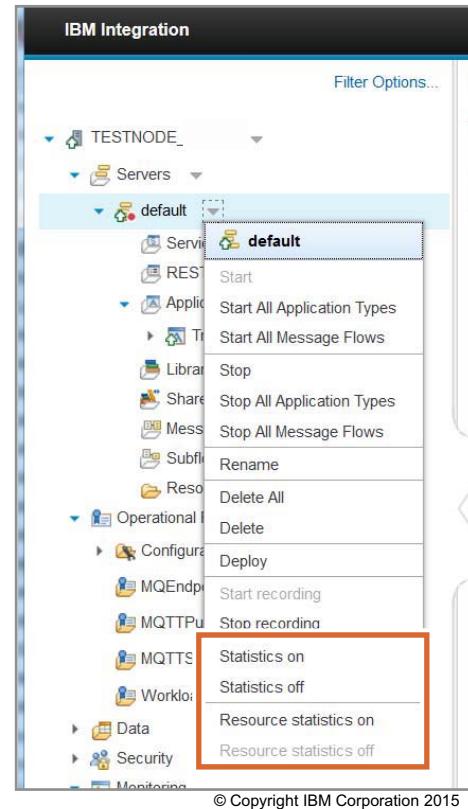
For example, if the integration node hosts a set of message flows associated with a particular client in a single integration server, you can set a specific value for the accounting origin for all these flows. You can then analyze the output that is provided to assess the use that the client or department makes of the integration node, and charge them.

If you want to track the behavior of a particular message flow, you can set a unique accounting origin for this message flow, and analyze its activity over a specified period.



## Message flow statistics in IBM Integration Bus

- Start and stop data collection by using the **mqsichangeflowstats** command or the IBM Integration web user interface
  - Request snapshot data collection by using the IBM Integration web user interface
  - Request snapshot data collection, archive data collection, or both by using the **mqsichangeflowstats** command
- Default level of statistics (message flow) collection has minimal performance effect on integration node performance
- View statistics data as it is generated by clicking the **Statistics** tab in the IBM Integration web user interface



© Copyright IBM Corporation 2015

Figure 13-21. Message flow statistics in IBM Integration Bus

WM666 / ZM6661.0

### Notes:

You can activate and display the message flow and resource statistics in the IBM Integration web user interface or by using IBM Integration Bus commands.

## Governance and policy-driven flows

- Many businesses are moving toward a policy-driven business processing and messaging applications, where business rules are encapsulated in registries or repositories
  - Application software can query repositories to determine the processing steps, alternative control flows, and information
  - Policy-driven flows make message flows dynamic, where an outside entity (the rules in the repository) can control how a message flow operates
  - To change a business rule, only the repository must be updated, not each application that uses the business rule
- DecisionService node implements business rules in message flows to provide operations like routing, validation, and transformation
- IBM WebSphere Registry and Repository can be used to manage business rules
  - EndpointLookup and the RegistryLookup nodes are used in message flow to access information

© Copyright IBM Corporation 2015

Figure 13-22. Governance and policy-driven flows

WM666 / ZM6661.0

### Notes:

In some applications, it might be beneficial to have business rules or even program code that is dynamically selected and loaded at run time from an external repository.

IBM Integration Bus provides connectivity to the IBM Decision Server and WebSphere Registry and Repository.

The DecisionService node implements business rules in message flows by using the IBM Decision Server.

WebSphere Registry and Repository can be used to manage external resources. In a message flow, the EndpointLookup node and RegistryLookup node can be used to retrieve information from WebSphere Registry and Repository.

## DecisionService node

- Allows IBM Integration Bus to call business rules that run on a component of IBM Decision Server that is provided with IBM Integration Bus
- Use this node in a message flow to control operations such as routing, validation, or transformation
- Before you deploy message flows that contain a DecisionService 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 2015

Figure 13-23. DecisionService node

WM666 / ZM6661.0

### Notes:

## EndpointLookup node



- Retrieves the service endpoint information that is held in the WebSphere Service Registry and Repository
  - Includes the web service URL destination for SOAP and HTTP request nodes
- WSDL describes service endpoint information that is related to a WebSphere Service Registry and Repository service
  - WSDL definition defines a service by using the interface (`portType`) that is available at a specified port
  - WSDL port defines the endpoint information that is required to access the service
- Retrieved data is placed in the local environment tree and is available to subsequent nodes
- Can automatically set up the destination URL that a subsequent `SOAPRequest`, `SOAPAsyncRequest`, or `HTTPRequest` node can use, depending on the value of the **Match Policy** property
- Input message is not modified

© Copyright IBM Corporation 2015

Figure 13-24. EndpointLookup node

WM666 / ZM6661.0

### Notes:

You can use the `EndpointLookup` node to retrieve service endpoint information that is held in the WebSphere Service Registry and Repository.

The `EndpointLookup` node retrieves service endpoint information that is related to a WebSphere Service Registry and Repository service described by WSDL. A WSDL definition defines a service in terms of an interface (referred to as a `portType`) made available at a specified port. The WSDL port defines the endpoint information that is required to access the service. Endpoints are retrieved according to search criteria that node properties define, optionally supplement or override by local environment definitions at run time.

The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the web service URL destination that is used by the SOAP and HTTP request nodes can also be set.

The input message is not modified.



## RegistryLookup node

- Works like the EndpointLookup node, but retrieves any type of information that is held in the WebSphere Service Registry and Repository, not just endpoint information  
Example: Any type of “constant” or “master data” information such as server names and IP addresses that an application might need
- Retrieved data is placed in the local environment tree and is available to subsequent nodes
- If more than one “set” of information is retrieved, based on the search criteria, the **Match Policy** property dictates whether only the first matching set is returned, or all matching sets are returned
- Input message is not modified

© Copyright IBM Corporation 2015

Figure 13-25. RegistryLookup node

WM666 / ZM6661.0

### Notes:

You can use the RegistryLookup node to retrieve any type of entity that is held in the WebSphere Service Registry and Repository. Data is retrieved according to search criteria that the node properties define. The entities that match the specified search criteria are stored in the local environment.

The input message is not modified.



## Monitoring support in IBM Integration Bus

- Events are published on a well-known topic over IBM MQ or MQTT for multiple concurrent consumers
- Event format is XML
  - Allows Integration Bus to integrate with other monitoring applications
  - Allows entire message to be captured and logged to a database for audit purposes
- Events are optionally produced within the same transaction sync point for optimum performance
- IBM Business Monitor integration
  - Monitor and analyze the key performance indicators
  - Automatic generation of a monitor model
  - Comprehensive sample built-in

© Copyright IBM Corporation 2015

Figure 13-26. Monitoring support in IBM Integration Bus

WM666 / ZM6661.0

### Notes:

After it is created, the monitoring profile can be deployed with an existing flow, or to an already running flow. It can be turned on or off with the change flow monitoring command.

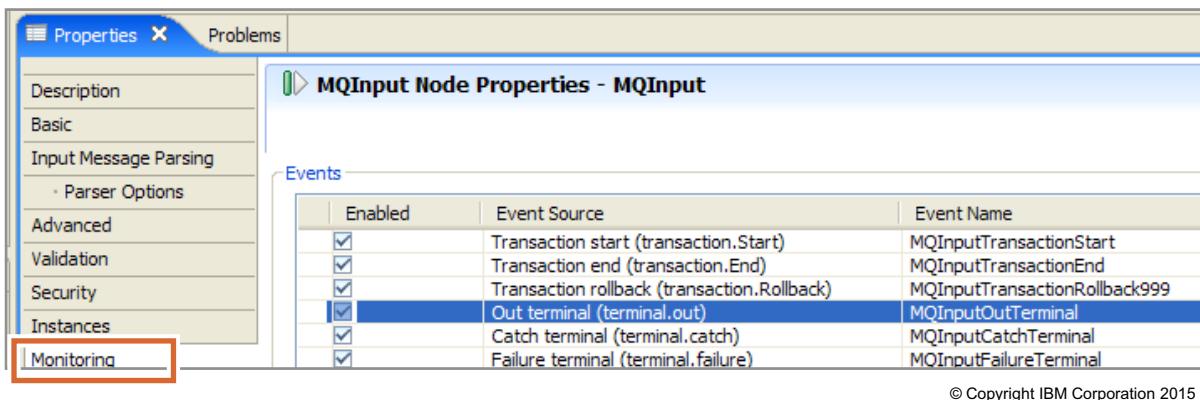
Events are published over a publish/subscribe network as a well-defined topic. The topic is named with the name of integration node, integration server, and the name that is given to the event as it is generated. It is possible to generate an event at design-time, or operationally at any point throughout the flow, that contains the entire payload or any piece of it.

A message driven bean captures the publication and pushes it into common event infrastructure for integration with IBM Business Monitor. IBM Business Monitor can be used to trend the data; for example, analyzing the volume of changes, or breaking out the information by geographical area.

 WebSphere Education 

## Generate monitoring events in the Integration Toolkit

- Generate monitoring and audit events from the message flow
  - Every Integration Bus node includes a **Monitoring** tab to generate events
  - Transaction start, end, and roll back are sent from input nodes from any terminal on any node
  - Configure payload data, content style, identity, correlation, and sequencing data
  - Noninvasive nature allows a monitoring profile to be applied to existing flows
  - Event filters to limit exact conditions for an event creation; for example:  
msg.Price>100



© Copyright IBM Corporation 2015

Figure 13-27. Generate monitoring events in the Integration Toolkit

WM666 / ZM6661.0

### Notes:

With monitoring and auditing you can capture information about your message flow. For example, it might be necessary to capture the first three fields of an input message at a certain point in the flow. The other requirement might be to capture the entire payload of the output message.

The capture requirements are defined in the IBM Integration Toolkit by using the **Monitoring** tab on the node properties. With the **Monitoring** tab, you can declare the event that you want to see from this node, such as a Transaction start event or a Transaction end event by selecting the check box next to the event. In the example, the MQInput node has monitoring enabled at all points for transaction: Transaction start, Transaction end, Transaction rollback, **Out** terminal, **Catch** terminal, and **Failure** terminal.

It is also possible to declare sequencing information, correlating information, and what the payload looks like.

Event filters can be defined to generate only events when a certain condition applies, such as when a field in the message exceeds a specified value.



## Configure event monitoring by using commands

- Create monitoring profiles on the integration node by using the `mqsicreateconfigurableservice` and the `mqsicangeproperties` commands
- Activate monitoring for the flow by typing:  
`mqsicangeflowmonitoring -c active`
- Enable event sources by typing:  
`mqsicangeflowmonitoring -i enable`
- Ensure that the event sources are enabled by using the `mqsireportflowmonitoring` command
  
- A monitoring profile configurable service overrides the monitoring properties of a message flow
- The monitoring profile can be used to customize events without redeploying the message flow

© Copyright IBM Corporation 2015

Figure 13-28. Configure event monitoring by using commands

WM666 / ZM6661.0

### Notes:



## Record and replay

- Enable record, edit, and replay of in-flight data
- Universal support for file, messaging, web service, EIS, and other data
- Browser tools for role-based access to users, auditors, and administrators
- Multiple data formats (binary, text, and XML) with customizable encryption and data masking

The screenshot shows the Data Capture Store interface. At the top, there are tabs for 'Data viewer' and 'Replay list'. Below that, a dropdown menu is set to 'Trades' and there is a checkbox labeled 'Mark for replay'. On the right, there is a 'Customize' button. The main area is a table titled 'No filter applied' with columns: Event time, Local Transaction ID, Parent Transaction ID, Global Transaction ID, Data, Errors, and Event name. The table contains five rows of 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

© Copyright IBM Corporation 2015

Figure 13-29. Record and replay

WM666 / ZM6661.0

### Notes:

IBM Integration Bus can record and replay in-flight data. While monitoring and auditing is restricted to capturing data at specific endpoints only, record and replay captures and manages message data as it is processed. The record and replay function gives you the ability to capture data that contains errors (or is badly formatted), correct it, and then resubmit it for processing.

The record and replay function in IBM Integration Bus is constructed around the integration node HTTP/REST web interface.

At a high level, the primary operations that are involved in record and replay are:

1. Some type of trigger event occurs that causes message details to be enqueued. For example, some type of processing error occurs in a message flow.
2. The integration node records the message in a database.
3. Using IBM Integration Bus and the HTTP/REST web interface, a user can query saved messages.
4. If authorized, the user can edit individual messages. For example, the user might review the failed message to determine the cause of the failure, and then correct the data in the record.

5. If authorized, the user can replay messages to the same or a different source. For example, the user can resubmit the corrected message to IBM MQ for reprocessing.



## Record and replay features

- Data recording provides a view of in-flight data
- Editing facilities to graphically edit and repair recorded data
- Replay information to allow for resubmission or redelivery processing
- Multiple sources and targets handle various message transport types
- Broad range of data formats so the most common data formats can be processed
- Web-based tools for record, edit, and replay
- Role-based access provides different job functions with different user interfaces
- Security for encryption and masking of sensitive information
- Independent source and target so no changes are needed for information producers and consumers
- Simple deployment integrates with existing infrastructure

© Copyright IBM Corporation 2015

Figure 13-30. Record and replay features

WM666 / ZM6661.0

### Notes:

The figure lists the record and replay features.

If you need an audit record of messages that pass through the integration node, you can record those messages. After you record the data, you can view it in several ways. By using the web interface, you can view a list of recorded messages, or you can view details of a specific message. You can also view recorded data by using the Integration API or the IBM Integration Bus REST API.



## Performance tuning

- Overall performance is a function message flow performance, which is a function of:
  - Integration node performance
  - Integration node queue manager performance
  - External application performance
  - Operating system performance
  - Hardware performance
- A message flow is just one part in the end-to-end integration solution
  - Keep in mind when designing or tuning
  - For security, transaction control, and elapsed time

© Copyright IBM Corporation 2015

Figure 13-31. Performance tuning

WM666 / ZM6661.0

### Notes:

The message rate that is possible to achieve when running a message flow is a function of the performance characteristics of a number of different but related items:

- Message flow: How it is written.
- Integration node: How it is configured.
- Databases: Frequency of use by message flows and the type of operation.
- Integration node queue manager: The efficiency of MQGET and MQPUT calls sent from message flows.
- Operating system: Affects overall performance.
- Hardware: The speed and number of processors, memory, number, speed, and frequency of access to disk storage, all contribute to the message rate.



## Common causes of performance problems

- Lack of resources with which to run the message flow
- A poorly configured environment
- Poor response time from a dependent application
- Inefficient processing algorithm for the message flow
- Inefficient or excessive ESQL processing within the message flow

© Copyright IBM Corporation 2015

Figure 13-32. Common causes of performance problems

WM666 / ZM6661.0

### Notes:

This figure lists some of the common causes of message flow and integration node performance problems.

## IBM Integration Bus tuning options

Tuning option	Component	AIX	Solaris	HP	Windows
Trusted integration node	Integration node	N	Y	Y	Y
Multiple instances	Integration node	Y	Y	Y	Y
Multiple integration servers	Integration node	Y	Y	Y	Y
Trace	Integration node	Y	Y	Y	Y
Topology of IBM MQ components*	Queue manager	Y	Y	Y	Y
Trusted channel and listener*	Queue manager	Y	Y	Y	Y
Queue manager logging*	Queue manager	Y	Y	Y	Y

**\*Note:** These tuning options are for an integration node that is associated with an IBM MQ queue manager.

© Copyright IBM Corporation 2015

Figure 13-33. IBM Integration Bus tuning options

WM666 / ZM6661.0

### Notes:

Tuning options cover the integration, an associated queue manager (if configured), and any databases that contain business data across each of the hardware and operating systems on which IBM Integration Bus is implemented.

In some environments, it is possible to run an integration node as an IBM MQ trusted application to streamline MQPUT and MQGET calls by the integration node. Performance is improved because the integration node interacts directly with the queue manager rather than an intermediary agent process. The increase in performance is at the cost of not isolating the queue manager from errant program behavior, such as poorly coded plug-in node, which can cause the integration node queue manager to fail.

The benefit that is obtained from running an integration node in trusted mode depends on the processing within the message flows. Different message flows running in the same integration node might see different results. A message flow that has a high proportion of MQPUT or MQGET processing of nonpersistent messages sees more benefit. This processing is equivalent to a simple MQInput to MQOutput test.

Message flows in which most of the processing is manipulating the contents of a message see a much lower benefit from using a trusted integration node and might see no gain at all.

In the figure, a value of **Y** indicates that the option applies or is available. It is not the default value but can be enabled. A value of **N** indicates that the option is not available or is not suggested. Each of these options is described in more detail next.

## Multiple copies of a message flow

	More instances	More integration servers	More integration nodes
Separation level	Thread	Process	Integration node
More resources	Storage	Storage	Storage, processor, and disk
Management processes	Low	Medium	Highest
Scaling effectiveness	Good (limited on Solaris)	Most efficient	Good for horizontal scaling; otherwise, expensive

© Copyright IBM Corporation 2015

Figure 13-34. Multiple copies of a message flow

WM666 / ZM6661.0

### Notes:

This table summarizes the options for running more copies of a message flow for throughput.

Another message flow instance (within an integration server) involves another thread that is running in that integration server process. In some situations, more instances do not provide sufficient separation.

Different integration servers provide separation at the process or address space level, which the operating system enforces. With multiple integration servers, it would be possible to run common message flow on different servers. This architecture provides the maximum level of separation but is the most expensive in hardware resources.

In the figure, the **Management processes** row indicates how much extra effort is involved with each of the techniques. The amount of work to add instances is low, since it is only the number of copies in the integration server that is changed. If there are many integration servers, managing them in the IBM Integration Toolkit can become a problem.

Creating multiple integration nodes affects performance the most. Each integration node queue manager must be connected into the existing topology. A new integration node and queue manager

must be created and more memory, processor, and disk space are required on the integration node server.

When you must add more copies of a message flow, first use more instances to increase message throughput. If adding more instances is not an option, try multiple integration servers to fully use the server on which the integration node is running, or to use as much of it as possible. If further increases in message throughput are needed, use multiple integration nodes. Establish the best use of more instances and integration servers before replicating the integration node.

When assigning message flows to integration servers, business requirements might dictate that certain message flows should not run in the same integration server. This might affect the policy that you use to assign message flows to integration servers.

In some situations, a message flow might have heavy resource requirements, such as the requirement for many processor cycles or memory. By assigning such flows to a restricted number of integration servers, the effects can be limited. This situation can be true where a message flow has a large high water mark for virtual memory use. If such a flow is assigned to all integration servers, the total virtual memory requirements are large, which places a greater demand on the memory of the server. This case might lead to a noticeable increase in paging or swapping. If the message flow is only assigned to a limited number of integration servers, the effect is contained.

To determine how many copies of the message flow are required, you must know (or estimate) what the target message rate is and compare that to the rate achieved with one copy of the message flow. Measurements are also required. Next, you must establish an overall goal. Do you want to achieve a particular message rate, reach a particular processor usage, or run in the most efficient way (although that might not lead to the maximum message rate)?

Run a single copy of the message flow with a realistic message. If the message flow processes multiple types of messages, make sure that all of the required message types are present and in their correct proportions. Continuously load the input queues to get a message flow that is running in a steady state. Measure the message throughput with tools, such as `vmstat`, to also determine the relative state of the system.

From these initial measurements, it should be possible to see whether the processor or I/O limits the message flow and affects the message rate if more copies of the message flow are deployed. It might also mean that you must change the hardware configuration to overcome an I/O constraint.

## Scaling message throughput

- Ability to scale message throughput depends on:
  - Sufficient resources (processor, memory, and disks)
  - Ability to schedule multiple pieces of work in parallel
  - A suitable application (lack of contention)
- Hardware is a planning issue
- An integration node can run multiple copies of a message flow
- Contention is an application design issue, and the following factors affect it:
  - Message type
  - Level of queue access
  - Nature of any database processing (insert-delete-update versus read-only)

© Copyright IBM Corporation 2015

Figure 13-35. Scaling message throughput

WM666 / ZM6661.0

### Notes:

It is difficult to accurately predict the precise benefits that are obtained when running multiple copies of a message flow since it depends on the design of the message flow. However, message flows must have certain characteristics to scale well.

- They should have low levels of I/O, either queue or database related. I/O normally imposes a lower maximum message rate than is the case when the processor limits processing.
- They should have low levels of queue access.
- They should have low levels of database insert, delete, and update activity. Read processing is far more scalable.

While the ideal is to produce message flows that give the level of required performance with minor tuning of the hardware, more benefit might be obtained by upgrading to faster processors or disks.

By adding more copies of a message flow, your aim is to balance throughput and resources. It is often difficult to follow this ideal for a number of reasons. What can often happen is that you reach the limitation of a resource, perhaps the speed at which you can insert messages into a database. In such a case, the improvement of more copies reduces and you start to reach a plateau.

The ability to increase message throughput by running more copies of the message flow is an important consideration for many users. They must know that no inherent limitations prevent them from growing vertically (within a server) and horizontally (across servers) as they require.

To increase throughput with any transaction or messaging software, some fundamentals must be in place.

- Physical resources must be in place. That is, processor, memory, and disks.
- There must be the ability to schedule concurrent pieces of work (multiprogramming).
- There must be minimum contention within the application so that there is no conflict between running multiple copies concurrently. For example, if each processing element needed access to a central control record, it would be a point of contention and would limit throughput gains from running multiple copies.

Ensuring that sufficient hardware is available is really a case of recognizing what is required. You must implement the message flows, test, and measure to adequately determine the requirements.

Contention between message flows is not something you can predict. It depends on how the message flows are coded and the data that they reference. Items to be aware of are high levels of access to few queues and database processing. If messages are located over multiple input queues, there is less chance of contention than if a single queue is used for all message processing.

Message type is also a factor. Persistent messages are locked for a longer time than nonpersistent messages because I/O takes place in the log. In addition, the maximum message rate that is achieved with persistent messages is lower than messages as there is an obvious I/O limitation.

Consider carefully any database processing. Avoid control records that multiple message flows use. Realize that insert, delete, and update processing is more intensive than read processing. With insert, delete, and update activity, changes must be logged.



## Tuning the integration node

- When connecting to IBM MQ, run as a trusted application to improve communication with the queue manager
- Traces and trace nodes
  - Integration node trace uses significant processing resources
  - Disable trace node output in IBM Integration web user interface or by using the `mqsichangetrace -n` command

© Copyright IBM Corporation 2015

Figure 13-36. Tuning the integration node

WM666 / ZM6661.0

### Notes:

Turn off all user and service level traces within the integration node.

Disable Trace node output by using the `mqsichangetrace`. You can use the `mqsichangetrace` command to suppress Trace node output without deleting the Trace nodes from the message flow. Deactivated Trace nodes incur no extra processing.

## Tuning the IBM MQ queue manager

- Examine the performance of connected queue managers and make sure that all of them are optimally configured and tuned
- Run the IBM MQ queue manager channels and listener as trusted applications to help reduce processor consumption and allow greater throughput
- Logging
  - When using persistent messages, review the log buffer settings and the speed of the device on which the queue manager log is located to ensure the queue manager log is efficient
  - Overhead varies with hardware, operating system, and message flow
  - Minimize I/O times

© Copyright IBM Corporation 2015

Figure 13-37. Tuning the IBM MQ queue manager

WM666 / ZM6661.0

### Notes:

If the message flows handle IBM MQ messages, ensure that the integration node is as close as possible to the messages to be processed. If output messages must be moved to a remote queue manager, processing cost increases for those messages if the messages are persistent. With persistent messages, the performance of logging over multiple queue managers contributes to slower throughput. Each message must be logged multiple times as it passes from one queue manager to another.

When persistent messages are used, the queue manager must successfully send a synchronous “write” at commit time to the log before a unit of work is complete. The “write” can significantly add to the extra resources of message processing, which becomes apparent when you compare the rates at which persistent and nonpersistent messages can be processed.

When tuning logging parameters, also consider changing those parameters that are related to log buffer and extent size. Large log extents mean less frequent switching between extents. The extra processing that results from the logging also varies with hardware and operating system.

In a system in which queue managers are interconnected, both queue managers must be configured for efficient logging. Efficient logging is required because the log on both queue

managers is used as the messages move across the IBM MQ channel that connects the two queue managers.

## Tuning databases

- Usage varies with the message flow composition
- Dynamic caching
  - Might offer significant gains
  - Monitor effectiveness
- Number of database connections
  - For each integration node, one connection per thread (ODBC-DSN)
  - Adjust the MAXAPPLS settings for a database (DB2)
  - Connections are held until the integration node stops
- Minimize log and data I/O time
- Turn off trace (database and ODBC)

© Copyright IBM Corporation 2015

Figure 13-38. Tuning databases

WM666 / ZM6661.0

### Notes:

The extent to which a database that contains business data is accessed in a message flow depends on the composition of the message flow. If there is no access, no tuning is required.

Usage of business data can vary from read-only activity through update-insert-delete. The type of processing that takes place determines where you must place your tuning effort.

Ensure that both database and ODBC trace are turned off. ODBC tracing is controlled differently on each of the different hardware and operating system types.

Where message flows frequently access a database, a significant performance benefit can be obtained by caching the dynamic SQL, which is generated when access occurs. Without such caching, each data access must be compiled by using an SQL `PREPARE` statement and then run by using an SQL `EXECUTE` statement. The compilation is relatively expensive. The aim is to significantly reduce the number of times it takes place.

Ensure that each database is configured with sufficient connections. The connection from the message flow to the database manager is made on the first database request in the flow.

Depending on the number of connections that are defined within the database manager and the

number of instances of the message flow, not all requests can be connected, which obviously reduces the throughput potential.

Examine the allocation of data and index buffers, and ensure that there are enough to meet your needs. Also, verify that good buffer hit ratios are obtained.

Consider creating a read-only view of those tables, which are rarely, if ever, updated. A read-only view reduces locking by the database manager and leads to better read performance.

When a database insert, delete, and update activity takes place, a synchronous write to the database manager log is required. It is important to ensure that the database log is on a fast device.

A BLOB written to a database is an immediate write to disk. It is not buffered like other data. If BLOB I/O is frequent, it is beneficial to locate the data portion of the database on a fast device.

## Detecting and removing obstructions

- Performance testing of the message flows is essential; run message flows individually and collectively
  - Run each flow with 1, 2, and 4 integration nodes
  - Mix the message flows in an integration server
  - Measure, measure, and measure
- Emulate the production environment
  - Run on the same hardware and operating system as the production environment
  - Run the same configuration as the production environment
- Run well beyond the current requirements
- Experiment to find the best configuration for the environment
- Start with whatever message type is required for production
  - Ensure that it is possible to fully use the hardware
  - Find the constraint if at all possible
- If the system is I/O bound, try increasing the number of message flows
- Test with bad data to ensure correct error handling

© Copyright IBM Corporation 2015

Figure 13-39. Detecting and removing obstructions

WM666 / ZM6661.0

### Notes:

It is essential to undertake performance testing of message flows before production.

You might observe that all is well when running few copies of a message flow but that things change when more copies are running. It is essential to discover this change in behavior before production.

First, determine the effects of running an increasing number of copies of each message flow. Run message flows in isolation, and in combination, to see whether there are any adverse effects.

Be sure to test in the same environment as production, ideally by using the same type of hardware and operating system.

If the message flow is processing IBM MQ messages, test with nonpersistent messages at first, even if you intend to use persistent messages in production. Using nonpersistent messages remove the limit (I/O impact) that the use of persistent messages uses. Such testing can make a constraint apparent, which would not surface otherwise.

In performance testing, it is important to reflect the production environment as fully as possible. In most cases, it is typical to have a continual flow of messages through the system, rather than processing messages in batches. What often happens in testing is that many messages are loaded

onto a queue and then processed. This approach produces different results from a continual topping up of the input queues.



## Unit summary

Having completed this unit, you should be able to:

- Deploy applications and shared libraries at run time to affect the visibility of resources
- Use promoted properties, user-defined properties, and operational policies to develop environment-aware message flows
- Dynamically route messages in a message flow by using external registries and registry lookup nodes to allow policy-driven message flows to meet governance requirements
- Add monitoring and auditing to a message flow
- Perform basic performance analysis on message flows

© Copyright IBM Corporation 2015

Figure 13-40. Unit summary

WM666 / ZM6661.0

### Notes:

## Checkpoint questions

1. How can you embed version information in your message flows? (Pick two).
  - a. By modifying an integration node property
  - b. By configuring the **Version** property of the message flow
  - c. By adding a user-defined keyword between \$MQSI tags
2. True or False: The EndpointLookup and RegistryLookup nodes can be used with any governance repository manager.
3. What type of information about the runtime environment can be queried in the message flow?

© Copyright IBM Corporation 2015

Figure 13-41. Checkpoint questions

WM666 / ZM6661.0

### Notes:

Write your answers here:

- 1.
- 2.
- 3.

## Checkpoint answers

1. How can you embed version information in your message flows?  
(Pick two).
  - a. By modifying an integration node property
  - b. By configuring the **Version** property of the message flow
  - c. By adding a user-defined keyword between \$MQSI tags

**Answer:** b and c.
2. True or False: The EndpointLookup and RegistryLookup nodes can be used with any governance repository manager  
**Answer: False.** These nodes are designed to work with the WebSphere Service Registry and Repository.
3. What type of information about the runtime environment can be queried in the message flow?  
**Answer: IBM Integration Bus attributes (integration server, message flow, and node-related) and user-defined attributes.**

© Copyright IBM Corporation 2015

Figure 13-42. Checkpoint answers

WM666 / ZM6661.0

### Notes:

## Exercise 11



Creating a runtime-aware message flow

© Copyright IBM Corporation 2015

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

10.1

Figure 13-43. Exercise 11

WM666 / ZM6661.0

### Notes:



## Exercise objectives

After completing this exercise, you should be able to:

- Add a user-defined property to a message flow
- Promote subflow properties to the main flow
- Define custom keywords
- Set configurable properties in the BAR file
- View BAR file properties at run time

© Copyright IBM Corporation 2015

---

Figure 13-44. Exercise objectives

WM666 / ZM6661.0

### Notes:

See the *Student Exercise Guide* for detailed instructions.

# Unit 14. 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 2015

Figure 14-1. Unit objectives

WM666 / ZM6661.0

### Notes:



## Course learning objectives

After completing this course, you should be able to:

- Describe the features and uses of the IBM Integration Bus
- Develop, deploy, and test message flow applications
- Generate message flow applications from predefined patterns
- Use IBM Integration Bus problem determination aids to diagnose and solve development and runtime errors
- Describe the function and appropriate use of IBM Integration Bus processing nodes
- Write basic Extended Structured Query Language and Java programs to transform data
- Use the IBM Graphical Data Mapping editor to transform data

© Copyright IBM Corporation 2015

Figure 14-2. Course learning objectives

WM666 / ZM6661.0

### Notes:



## Course learning objectives

After completing this course, you should be able to:

- Define, use, and test simple XML and Data Format Description Language (DFDL) data models
- Describe supported transport protocols and how to call them in message flows

© Copyright IBM Corporation 2015

Figure 14-3. Course learning objectives

WM666 / ZM6661.0

### 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 2015

Figure 14-4. To learn more on the subject

WM666 / ZM6661.0

### 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 2015

---

Figure 14-5. Unit summary

WM666 / ZM6661.0

### Notes:

# Appendix A. ESQL examples

## What this unit is about

This appendix supplements the ESQL information that the course units provide. It contains examples of ESQL statements that might be used in an ESQL program for a Compute node, or ESQL statements in a Filter node, or Database node.

This unit is not intended to be taught as a course unit.

## References

For information about ESQL syntax, see the IBM Knowledge Center

## ESQL

- Extends the constructs of the SQL language to define the behavior of nodes in a message flow
  - Test, calculate, and manipulate fields in logical message
  - Reference message header fields
  - Change field properties
- Based on standard SQL 3

### Examples:

```
RETURN Body.Msg.Name = 'IBM';
SET OutputRoot.XML.Msg.Name = 'IBM';
SET OutputRoot.XML.Msg.Name=InputBody."First.Name";
```

© Copyright IBM Corporation 2015

Figure A-1. ESQL

WM666 / ZM6661.0

### Notes:

## Some ESQL syntax

- ESQL keywords not case-sensitive  
Examples: `SET set SeT.....`
- Correlation names and field references **are** case-sensitive  
Examples: `OutputRoot, InputBody.Msg.field`
- Character constants and literals are enclosed with single quotation marks (' ')
- Field references are enclosed with double quotation marks (" ") if the reference:
  - Contains special characters in name, such as a blank space
  - Is an ESQL reserved word
- Statement delimiter is a semicolon (;
- Comments
  - One line: -- (two dashes)
  - Block: /\*..comment here ..\*/

© Copyright IBM Corporation 2015

Figure A-2. Some ESQL syntax

WM666 / ZM6661.0

### Notes:

## ESQL data types for numbers and strings

- DECIMAL

Precision 1 - 31, scale 0 - 30

`SQL_C_CHAR`

- BIT

`b'10110'`

- FLOAT

`1.23e-45 6.7890E2` (64 bit)

`SQL_C_DOUBLE`

- BLOB

`x'3C493E'`

8-bit bytes, 2 digits per byte

`SQL_C_BINARY`

- INTEGER, INT

`+123 -890 0x123ab 0X3B`

64-bit 2-complement form

`SQL_C_LONG`

- CHARACTER

`'ABC' = 'ABC'`

`SQL_C_CHAR`

```
DECLARE MyDec DECIMAL 12345678.1111112;
SET OutputRoot."BLOB"."BLOB"=x'3C493E';
```

© Copyright IBM Corporation 2015

Figure A-3. ESQL data types for numbers and strings

WM666 / ZM6661.0

### Notes:

## Example: String functions

```

DECLARE I INTEGER;
SET I = POSITION('or' IN 'Hello World');
SET I = LENGTH('Hello World');

DECLARE S CHARACTER;
SET S = SUBSTRING('Hello World' FROM 8 FOR 2);
SET S = OVERLAY('Hello World'
    PLACING 'My' FROM 1 FOR 5);
SET S = REPLACE ('Hello','l','xx');
SET S = REPLICATE ('Hello',3);
SET S = TRANSLATE ('Hello World','eo','a');

SET S = TRIM(LEADING '!' FROM '!Hello!');
SET S = TRIM(BOTH ' ' FROM ' Hello ');
SET S = UPPER('Hello');
SET S = LOWER('Hello');

SET S = 'Hello' || 'World';

```

Output: 8  
Output: 11

Output: 'or'

Output: 'My World'  
Output: 'Hexxxxo'  
Output: 'HelloHelloHello'  
Output: 'Hall Wrld'

Output: 'Hello!'  
Output: 'Hello'  
Output: 'HELLO'  
Output: 'hello'

Output: 'HelloWorld'

© Copyright IBM Corporation 2015

Figure A-4. Example: String functions

WM666 / ZM6661.0

### Notes:

## ESQL data types for date, time, and Boolean

- DATE

`DATE '2001-10-03'`

`SQL_C_DATE`

- GMTIME, GMTIMESTAMP

Greenwich Mean Time

- TIME

`TIME '23:09:01.123456'`

`SQL_C_TIME`

- INTERVAL

- Either YEAR and MONTH (variable length)
- Or DAY HOUR MINUTE SECOND (fixed length)

`INTERVAL '90' MINUTE`

`INTERVAL '1-06' YEAR TO MONTH`

- TIMESTAMP

`TIMESTAMP '2001-10-03`

`23:09:01.123456'`

`SQL_C_TIMESTAMP`

- BOOLEAN

`TRUE, FALSE, UNKNOWN (NULL)`

`SQL_C_BIT`

```
Declare V1 Interval (CURRENT_DATE - Date '2015-01-01') DAY;
Declare V2 Interval (CURRENT_TIME - Time '09:00:00') HOUR TO SECOND;
```

© Copyright IBM Corporation 2015

Figure A-5. ESQL data types for date, time, and Boolean

WM666 / ZM6661.0

### Notes:

## Example: Functions for date and time

```
DECLARE D DATE CURRENT_DATE;  
  
DECLARE T TIME CURRENT_TIME;  
  
DECLARE TS TIMESTAMP CURRENT_TIMESTAMP;  
  
DECLARE GT GMTIME CURRENT_GMTIME;  
  
DECLARE IV INTERVAL LOCAL_TIMEZONE;  
  
DECLARE I INTEGER;  
SET I = EXTRACT(HOUR FROM CURRENT_TIME);
```

© Copyright IBM Corporation 2015

Figure A-6. Example: Functions for date and time

WM666 / ZM6661.0

### Notes:

## ESQL operators

- Arithmetic
  - Numeric, datetime, intervals: + -
  - Numeric, intervals: \* /
  - Unary (negation): -
  - Concatenate: ||
  - Order of precedence
- Comparisons: < > <= >= <> =
  - BETWEEN
 

Example: 3 BETWEEN 1 AND 10
  - IN
 

Example: 'A' IN (1, 'ABC', 123.4, 'A')
  - LIKE
 

Multiple characters: %  
'ABC' LIKE 'A%' → true

Single character: \_  
'ABC' LIKE 'A\_'" → false
  - IS, IS NOT tests NULLs (NULL, UNKNOWN, TRUE, FALSE)
 

Example: myVar IS NOT NULL

© Copyright IBM Corporation 2015

Figure A-7. ESQL operators

WM666 / ZM6661.0

### Notes:

## Example: Numeric functions

<b>CEIL</b> (3.673);	Round up to next integer, returns 4.0
<b>FLOOR</b> (3.673);	Round down to previous integer, returns 3.0
<b>ROUND</b> (3.673, 1);	Round with given precision, returns 3.7
<b>TRUNCATE</b> (3.673, 1);	Truncate to given precision, returns 3.6
<b>ABS</b> (-3.673);	Absolute value, returns 3.673
<b>MOD</b> (3, 7);	Modulus (remainder), returns 1
<b>SQRT</b> (4);	Square root, returns 2E+1
DECLARE I INTEGER; SET I = <b>BITAND</b> (7,12);	Bitwise AND for integers, returns 4

Other functions: BITNOT, BITOR, BITXOR

Many more mathematical functions for trigonometry, and so forth

© Copyright IBM Corporation 2015

Figure A-8. Example: Numeric functions

WM666 / ZM6661.0

### Notes:

## OrderMsg sample message

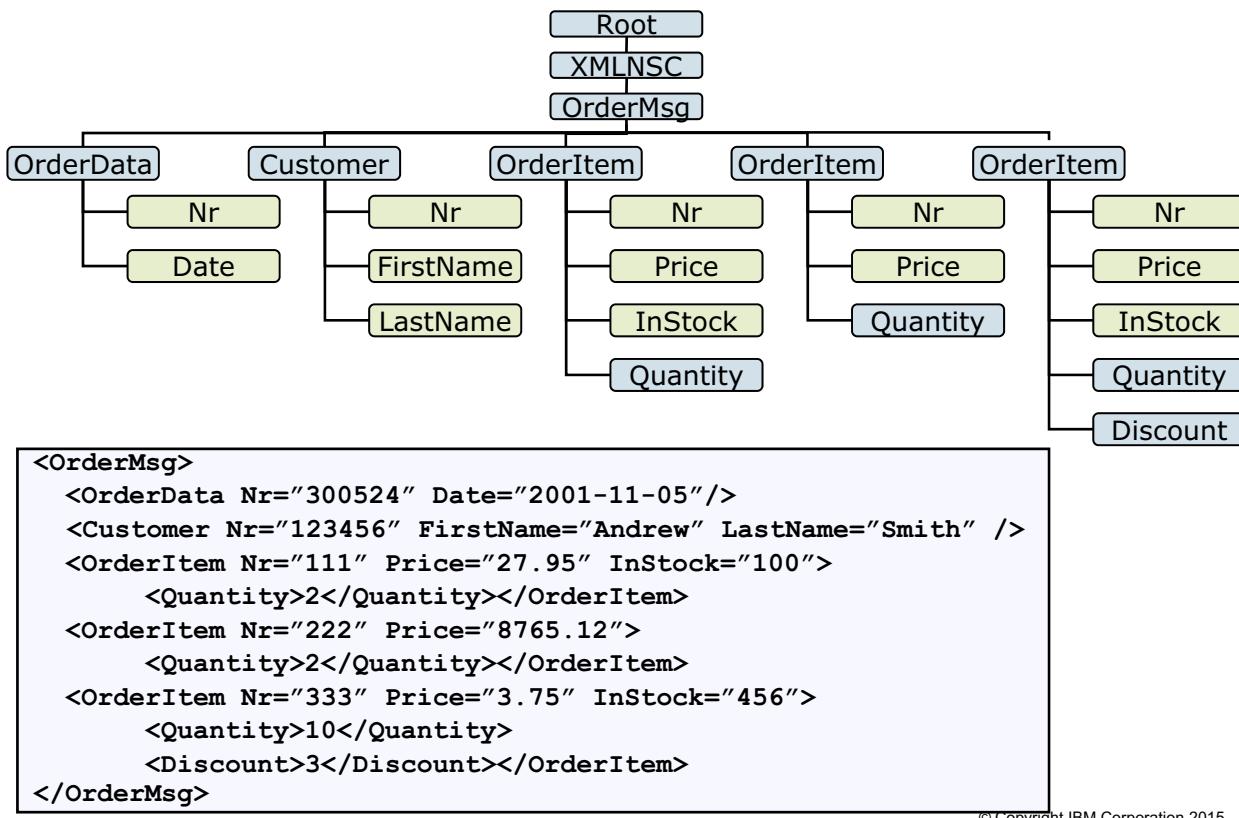


Figure A-9. OrderMsg sample message

WM666 / ZM6661.0

### Notes:

Use this example input file and sample message structure for the examples in this appendix.

## Field references in tree structures

- Paths are dot separated
- [i] Index (Origin 1; <1 is LAST)
- [ ] Array of (a list)
- ( ) Search for type (Name, XMLNSC.Attribute and so forth)
- :
- : Namespace followed by name
- \*
- \* Any element of any type, name, value (or namespace)
- { }
- { } Evaluate this expression (must yield CHAR string)

### Examples:

```
DECLARE i INTEGER CARDINALITY(InputRoot.*[]);
SET rowCust.'Customer-'||C = InputBody.Customer[<2];
SET OutputRoot.MRM.addr:addressDetails.addr:postcode = 'ZZ01 4WW';
SET OutputRoot.XMLNSC.A.(XMLNSC.Attribute)B = 3;
```

© Copyright IBM Corporation 2015

Figure A-10. Field references in tree structures

WM666 / ZM6661.0

### Notes:

## Examples: SET

**What is the value of V after the following ESQL statements?**

```
SET V = InputBody.OrderMsg.OrderItem.*[2];
```

Answer: '27.95'

```
SET V = InputBody.OrderMsg.OrderItem[<].*[<3];
```

Answer: '456'

```
SET V = InputBody.OrderMsg.OrderItem[2];
```

Answer: This statement fails because the right side of the SET statement points to a structure. A structure cannot be assigned to a normal ESQL variable, which is a scalar.

Assigning a structure to a message field is valid. For example:

```
SET OutputRoot.XMLNSC.OrderMsg.NewField  
=InputBody.OrderMsg.OrderItem[2];
```

© Copyright IBM Corporation 2015

Figure A-11. Examples: SET

WM666 / ZM6661.0

### Notes:

The SET statement assigns a value to a SQL variable.

**Syntax:** `SET identifier = expression`

## Examples: SET with CARDINALITY

**What is the value of V after the following ESQL statements?**

```
SET V = CARDINALITY(InputBody.OrderMsg.OrderItem[]) ;
```

Answer: 3

```
SET V = CARDINALITY(InputBody.OrderMsg.*[]) ;
```

Answer: 5

```
SET V = CARDINALITY(InputBody.OrderMsg.OrderItem) ;
```

Answer: This statement does not deploy. CARDINALITY needs a list (field array) as its argument.

```
SET V = CARDINALITY(InputBody.OrderMsg.*);
```

Answer: This statement does not deploy. Cardinality needs a list (field array) as its argument. To be syntactically correct, the list must be expressed with brackets ([]).

© Copyright IBM Corporation 2015

Figure A-12. Examples: SET with CARDINALITY

WM666 / ZM6661.0

### Notes:

The CARDINALITY function returns the number of elements in a list. A common use of this function is to determine the number of fields in a list before iterating over them.

#### Syntax: `CARDINALITY(ListExpression)`

CARDINALITY returns an integer value that gives the number of elements in the list that is specified by *ListExpression*. *ListExpression* is any expression that returns a list.

All the following return a list:

- A LIST constructor
- A field reference with the [] array indicator
- Some SELECT expressions (not all return a list)

## More SET examples (1 of 2)

- Copy the entire input message to the output message:

```
SET OutputRoot = InputRoot;
```

- Copy the body of the input message to the output message (which is also in the XML domain):

```
SET OutputRoot.XMLNSC = InputBody;
```

- Output the last name in uppercase:

```
SET OutputRoot.XMLNSC.OrderMsg.Customer.LastName =
UPPER(InputBody.OrderMsg.Customer.LastName);
```

- Delete any trailing spaces from the Customer.Nr field:

```
SET OutputRoot.XMLNSC.OrderMsg.Customer.Nr =
TRIM(TRAILING ' ' FROM InputBody.OrderMsg.Customer.Nr);
```

© Copyright IBM Corporation 2015

Figure A-13. More SET examples (1 of 2)

WM666 / ZM6661.0

### Notes:

## More SET examples (2 of 2)

- Append a new field (NewField) at the message body:

```
SET OutputRoot.XMLNSC.OrderMsg.NewField = 'any text' ;
```

Note: The XML specification requires exactly one root tag, which is in this case OrderMsg. So, you must append NewField below the OrderMsg tag.

- Create an output message in which the top-level XML element (the XML root tag) is named CustMsg instead of OrderMsg and the remainder of the message is the same as in the input message:

```
SET OutputRoot.XMLNSC.CustMsg = InputBody.OrderMsg;
```

- In the output message, double the Price for the first OrderItem:

```
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[1].Price =
2*CAST(InputBody.OrderItem[1].Price AS DECIMAL);
```

You must explicitly **cast** to DECIMAL for the calculation, but the XML parser (OutputRoot.XML) does an implicit cast to character

© Copyright IBM Corporation 2015

Figure A-14. More SET examples (2 of 2)

WM666 / ZM6661.0

### Notes:

## REFERENCE variables

- Use the DECLARE statement to define a variable, the data type of the variable and optionally, its initial value
- REFERENCE data type holds the location of a field in a message
  - It cannot hold the location of a constant, a database table, a database column, or another reference

Example:

```
DECLARE myRef REFERENCE TO InputBody.OrderMsg
```

© Copyright IBM Corporation 2015

Figure A-15. REFERENCE variables

WM666 / ZM6661.0

### Notes:

## Moving REFERENCE variables

- Move cursor
  - PARENT 
  - PREVIOUSSIBLING       NEXTSIBLING 
  - FIRSTCHILD       LASTCHILD

```
DECLARE myRef REFERENCE TO InputBody.OrderMsg;
MOVE myRef FIRSTCHILD;
MOVE myRef NEXTSIBLING NAME 'OrderItem';
MOVE myRef LASTCHILD TYPE XMLNSC.Attribute;
MOVE myRef PREVIOUSSIBLING REPEAT TYPE;
```

- Check success of MOVE or DECLARE...REFERENCE

```
WHILE LASTMOVE (myRef) DO...
```

© Copyright IBM Corporation 2015

Figure A-16. Moving REFERENCE variables

WM666 / ZM6661.0

### Notes:

## Example: REFERENCE variables (1 of 3)

**What is the value of V after each of the following ESQL statements?**

```
DECLARE V CHAR;
SET OutputRoot = InputRoot;
DECLARE myRef REFERENCE TO
OutputRoot.XMLNSC.OrderMsg.OrderData;

SET V=FIELDNAME(myRef);

MOVE myRef NEXTSIBLING NAME 'OrderItem';
SET V=FIELDNAME(myRef);

MOVE myRef LASTCHILD TYPE XMLNSC.Attribute;
SET V=FIELDNAME(myRef);
```

Value of V

OrderData

OrderItem

InStock

© Copyright IBM Corporation 2015

Figure A-17. Example: REFERENCE variables (1 of 3)

WM666 / ZM6661.0

### Notes:

## Example: REFERENCE variables (2 of 3)

**What is the value of V after each of the following ESQL statements?**

DECLARE V CHAR; SET OutputRoot = InputRoot; DECLARE myRef REFERENCE TO OutputRoot.XMLNSC.OrderMsg.OrderItem.Discount;  SET V=FIELDNAME (myRef) ;  MOVE myRef PARENT; SET V=FIELDNAME (myRef) ;  MOVE myRef FIRSTCHILD; SET V=FIELDNAME (myRef) ;	<u>Value of V</u>
	Root
	Root
	Properties

© Copyright IBM Corporation 2015

Figure A-18. Example: REFERENCE variables (2 of 3)

WM666 / ZM6661.0

### Notes:

In this example, `myRef` was declared for a non-existing field, and so it points to `Root`. If a move is not successful, such as in the first move because `Root` does not have a parent, the dynamic reference stays fixed. No runtime errors occur.

Use the `LASTMOVE` function to check the success of the move.

## Example: REFERENCE variables (3 of 3)

**What is the value of V after each of the following ESQL statements?**

```

DECLARE V CHAR;
SET OutputRoot = InputRoot;
DECLARE myRef REFERENCE TO
OutputRoot.XMLNSC.OrderMsg.OrderItem[<];
SET V=FIELDNAME(myRef);

CREATE PREVIOUSSIBLING of myRef TYPE Name
NAME 'OrderItemNew';

SET V=FIELDNAME(myRef);

CREATE FIRSTCHILD of myRef FROM
OutputRoot.XMLNSC.OrderMsg.OrderItem.Nr;
SET V=FIELDNAME(myRef);

```

Value of V

OrderItem

OrderItem

OrderItem

© Copyright IBM Corporation 2015

Figure A-19. Example: REFERENCE variables (3 of 3)

WM666 / ZM6661.0

### Notes:

The modified message would appear as follows:

```

<OrderMsg>
  <OrderData Nr="300524" Date="2001-11-05"/>
  <Customer Nr="123456" FirstName="Andrew" LastName="Smith"/>
  <OrderItem Nr="111" Price="27.95" InStock="100">
    <Quantity>2</Quantity></OrderItem>
    <OrderItem Nr="222" Price="8765.12">
      <Quantity>2</Quantity>
    </OrderItem>
    <OrderItemNew/>
  <OrderItem Nr="111" Nr="333" Price="3.75" InStock="456">
    <Quantity>10</Quantity>
    <Discount>3</Discount>
  </OrderItem>
</OrderMsg>

```

## Deleting and reordering fields

- Delete a field or structure

- Delete and free memory; useful for large messages

```
DELETE FIELD OutputRoot.XMLNSC.OrderMsg.Customer;
```

```
DECLARE myRef REFERENCE TO
    OutputRoot.XMLNSC.OrderMsg.OrderItem[1];
DELETE NEXTSIBLING OF myRef;
```

- Set to NULL

```
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[2]=NULL;
```

- Detach (cut and paste)

```
DETACH myRef;
```

- Reorder: Paste detached structure to new position in message tree

```
ATTACH myRef TO OutputRoot.XMLNSC.OrderMsg AS FIRSTCHILD;
```

© Copyright IBM Corporation 2015

Figure A-20. Deleting and reordering fields

WM666 / ZM6661.0

### Notes:

## Copying an entire message example

- Copy the entire message, and create a single field that is named WorkDate with the current date before the first OrderItem
- Use REFERENCE variables and a CREATE statement

```
SET OutputRoot = InputRoot;  
  
DECLARE myRef REFERENCE TO OutputRoot.XMLNSC.OrderMsg.Customer;  
  
CREATE NEXTSIBLING OF myRef TYPE Name NAME 'WorkDate' VALUE CURRENT_DATE;
```

© Copyright IBM Corporation 2015

Figure A-21. Copying an entire message example

WM666 / ZM6661.0

### Notes:

## Reordering output fields example

- Create an output message where the Order data is moved after the Customer data

Solution 1

```
SET OutputRoot = InputRoot;
DECLARE myRef REFERENCE TO OutputRoot.XMLNSC.OrderMsg.OrderData;
DETACH myRef;
ATTACH myRef TO OutputRoot.XMLNSC.OrderMsg.Customer AS NEXTSIBLING;
```

Solution 2

```
SET OutputRoot = InputRoot;
SET OutputRoot.XMLNSC.OrderMsg.OrderData = NULL; /* delete field */
DECLARE myRef REFERENCE TO OutputRoot.XMLNSC.OrderMsg.Customer;
CREATE NEXTSIBLING OF myRef FROM InputBody.OrderMsg.OrderData;
```

Solution 3

```
CALL ddCopyMessageHeaders();
SET OutputRoot.XMLNSC.OrderMsg.Customer = InputBody.OrderMsg.Customer;
SET OutputRoot.XMLNSC.OrderMsg.OrderData = InputBody.OrderMsg.OrderData;
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[] = InputBody.OrderMsg.OrderItem[];
```

© Copyright IBM Corporation 2015

Figure A-22. Reordering output fields example

WM666 / ZM6661.0

### Notes:

Solution 3 is cumbersome for long messages because you must build each field with a separate ESQL statement; it is better to use a Mappings node.



## NULL value

- NULL is distinct state
  - Not 0 or ''
  - SQL\_NULL\_DATA
  - All ESQL data types (except REFERENCE)
- If value is:
  - Unknown
  - Undefined
  - Uninitialized
- In general, if any operand is NULL then value is NULL
- Use NULL to delete field or structure

© Copyright IBM Corporation 2015

Figure A-23. NULL value

WM666 / ZM6661.0

### Notes:

## Example: Using NULL

- Delete (omit) the Customer structure from the output message:

```
SET OutputRoot.XMLNSC.OrderMsg.Customer = NULL;
```

- Delete (omit) the last OrderItem structure from the output message:

```
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[<] = NULL;
```

© Copyright IBM Corporation 2015

Figure A-24. Example: Using NULL

WM666 / ZM6661.0

### Notes:

## Example: Using NULL to modify the message structure

```
<OrderMsg>
  <OrderData Nr="300524" Date="2001-11-05"/>
  <Customer Nr="123456" FirstName="Andrew" LastName="Smith" />
  <OrderItem Nr="111" Price="27.95" InStock="100">
    <Quantity>2</Quantity></OrderItem>
  <OrderItem Nr="222" Price="8765.12">
    <Quantity>2</Quantity></OrderItem>
  <OrderItem Nr="333" Price="3.75" InStock="456">
    <Quantity>10</Quantity>
    <Discount>3</Discount></OrderItem>
  </OrderMsg>
```

Input message

```
SET OutputRoot=InputRoot;
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[1] = NULL;
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[2] = NULL;
```

ESQL code

```
<OrderMsg>
  <OrderData Nr="300524" Date="2001-11-05"/>
  <Customer Nr="123456" FirstName="Andrew" LastName="Smith"/>
  <OrderItem Nr="222" Price="8765.12">
    <Quantity>2</Quantity>
  </OrderItem>
</OrderMsg>
```

Output message

© Copyright IBM Corporation 2015

Figure A-25. Example: Using NULL to modify the message structure

WM666 / ZM6661.0

### Notes:

The result might not be what you expected. Because repeating elements are stored on a stack, when you take one above, the next takes its place. So, you should delete repeating elements in reverse order.



**Important**

Always consider the order in which ESQL statements are run.

## Creating fields or attributes

- Create fields (tags) or attributes
  - At any point in message
  - A scalar field

```
SET OutputRoot=InputRoot;
CREATE FIELD OutputRoot.XMLNSC.OrderMsg.Customer.AnAttr
  TYPE XMLNSC.Attribute VALUE 'C';
```

```
DECLARE myRef REFERENCE TO OutputRoot.XMLNSC;
CREATE PREVIOUSSIBLING OF myRef DOMAIN 'MQRFH2';
```

- Create a structure (per duplication)

```
DECLARE myRef REFERENCE TO OutputRoot.XMLNSC.OrderMsg;
CREATE FIRSTCHILD OF myRef FROM InputBody.OrderMsg.OrderItem[3];
```

© Copyright IBM Corporation 2015

Figure A-26. Creating fields or attributes

WM666 / ZM6661.0

### Notes:

## CAST: Transform into other data type

- CAST can convert character string into BLOB or BIT and vice versa, if CCSID and ENCODING specified
- ESQL generally does not do automatic casting
  - Runtime error if SET MyChar=MyInteger;
  - Some implicit CASTs for comparison and arithmetic operators
  - Recommendation: If in doubt, CAST
- CAST can use format patterns as in EXCEL and ICU

```

DECLARE B BLOB X'48656c6c6f';
          Output: 'X'48656c6c6f'
DECLARE S CHAR CAST(B AS CHAR);
SET S = CAST(B AS CHAR CCSID 850); Output: 'Hello'
SET S = CAST(CURRENT_DATE AS CHAR FORMAT 'dd-MM-yy') Output: '24-10-05'
```

© Copyright IBM Corporation 2015

Figure A-27. CAST: Transform into other data type

WM666 / ZM6661.0

### Notes:

## IF, THEN, ELSE statements

- Conditionally process ESQL statements

```
IF InputBody.Message.Department = 'Sales'  
THEN  
    SET OutputRoot.XMLNSC.Msg.Type = 'S';  
    SET OutputRoot.XMLNSC.Msg.DeptNo = '4711';  
ELSE  
    SET OutputRoot.XMLNSC.Msg.Type = 'X';  
    SET OutputRoot.XMLNSC.Msg.DeptNo = '9999';  
END IF;
```

© Copyright IBM Corporation 2015

Figure A-28. IF, THEN, ELSE statements

WM666 / ZM6661.0

### Notes:

## CASE function

- Like ?: Expression in C, but more powerful
- Simple form: 1:n test

```
SET OutputRoot.XMLNSC.Msg.Type = CASE InputBody.Msg.Department
    WHEN 'Sales' THEN 'S'
    WHEN 'Distribution' THEN 'D'
    ELSE 'X'
END;
```

- Searched form: n:n test

```
SET OutputRoot.XMLNSC.Msg.CustomerType = CASE
    WHEN InputBody.Msg.Custno = '007' THEN 'TopSecret'
    WHEN InputBody.Msg.OrderValue > 100000 THEN 'HighValue'
    ELSE 'Normal'
END;
```

- Specialized: NULLIF

```
SET OutputRoot.XMLNSC.Msg.CustNo = NULLIF(InputBody.Msg.Custno, '007')
```

© Copyright IBM Corporation 2015

Figure A-29. CASE function

WM666 / ZM6661.0

### Notes:

## CASE statement

- Like **SWITCH** statement in C, but more powerful
- Evaluates and runs multiple statements
- Simple form:  $1:n$  test

```
CASE InputBody.Msg.Department
    WHEN 'Sales' THEN SET OutputRoot.XMLNSC.Msg.Type = 'S';
    WHEN 'Distribution' THEN CALL handleDistribution();
    ELSE
        CALL handleUnknown();
        RETURN;
END CASE;
```

- Searched form:  $n:n$  test

```
CASE
    WHEN InputBody.Msg.Custno = '007' THEN
        SET OutputRoot.XMLNSC.Msg.CustomerType='TopSecret';
    WHEN InputBody.Msg.OrderValue > 100000
    THEN CALL handleHighValue();
END CASE;
```

© Copyright IBM Corporation 2015

Figure A-30. CASE statement

WM666 / ZM6661.0

### Notes:



## WHILE

- Use a WHILE loop to repeat a sequence of statements, for example:

```
DECLARE I INTEGER 1;  
WHILE I <= 10 DO  
    SET OutputRoot.XMLNSC.Msg.MYARRAY[I] = I;  
    SET I = I + 1;  
END WHILE;
```

© Copyright IBM Corporation 2015

Figure A-31. WHILE

WM666 / ZM6661.0

### Notes:

## RETURN statement

- When used in the Main function, the RETURN statement stops processing of the module and returns control to the next node in a message flow
  - In the Compute node and Database nodes, if *expression* is anything other than TRUE, propagation of the message is stopped
  - In the Filter node, the message is propagated to the terminal that matches the value of *expression*: TRUE, FALSE, and UNKNOWN.
- When used in a function or a procedure, the RETURN statement stops processing of that function and returns control to the calling expression

© Copyright IBM Corporation 2015

Figure A-32. RETURN statement

WM666 / ZM6661.0

### Notes:



## Example: RETURN in a Filter node

- Test if the message came from queue ‘OrderInQ’.

```
RETURN Root.MQMD.SourceQueue = 'OrderInQ' ;
```

- Test if the customer has a hyphenated LastName (like Miller-Smith).

```
RETURN Body.OrderMsg.Customer.LastName LIKE '%-%' ;
```

© Copyright IBM Corporation 2015

Figure A-33. Example: RETURN in a Filter node

WM666 / ZM6661.0

### Notes:

## Procedures

- Subroutines with IN, OUT, and INOUT parameters and optional RETURNS value
  - Can be used recursively
  - Language can be ESQL, Java, or database (stored procedure)

```

CREATE PROCEDURE navigate (IN root REFERENCE, INOUT answer CHARACTER)
LANGUAGE ESQL
BEGIN
    SET answer = answer || 'Reached Field...Name:' || FIELDNAME(root);
    DECLARE cursor REFERENCE TO root;
    MOVE cursor FIRSTCHILD;
    IF LASTMOVE(cursor) THEN
        SET answer = answer || 'Field has children... drilling down ';
    ELSE
        SET answer = answer || 'Listing siblings... ';
    END IF;
    WHILE LASTMOVE(cursor) DO
        CALL navigate(cursor, answer);
        MOVE cursor NEXTSIBLING;
    END WHILE;
    SET answer = answer || 'Finished siblings... Popping up ';
END;
  
```

© Copyright IBM Corporation 2015

Figure A-34. Procedures

WM666 / ZM6661.0

## Notes:



## Appendix B. List of abbreviations

<b>ACORD</b>	Association for Cooperative Operations Research and Development
<b>API</b>	Application programming interface
<b>ASCII</b>	American Standard Code for Information Interchange
<b>BCD</b>	binary-coded decimal
<b>BLOB</b>	binary large object
<b>BPM</b>	Business process management
<b>CCDT</b>	Client channel definition table
<b>CDA</b>	Common Debug Architecture
<b>CPU</b>	Central Processing Unit
<b>CRM</b>	Customer relationship management
<b>CSV</b>	comma-separated values
<b>CVS</b>	Concurrent Version System
<b>CWF</b>	Custom wire format
<b>DFDL</b>	Data format description language
<b>DHCP</b>	Dynamic Host Configuration Protocol
<b>DLQ</b>	dead-letter queue
<b>DOM</b>	Document Object Model
<b>DOS</b>	disk operating system
<b>DPL</b>	Distributed Program Link
<b>DSN</b>	Data source name
<b>DTD</b>	document type definition
<b>EBCDIC</b>	Extended Binary Coded Decimal Interchange Code
<b>EDI</b>	Electronic data interchange
<b>EIS</b>	Enterprise information system
<b>ERP</b>	Enterprise resource planning
<b>ESB</b>	Enterprise service bus
<b>ESQL</b>	Extended structure query language
<b>FTP</b>	File transfer protocol
<b>HFS</b>	hierarchical file system
<b>HIPAA</b>	Health Insurance Portability and Accountability Act
<b>HTTP</b>	Hypertext transfer protocol

<b>IaaS</b>	Infrastructure as a service
<b>IBM</b>	International Business Machines Corporation
<b>IDE</b>	Integrated development environment
<b>IIOP</b>	Internet Inter-ORB Protocol
<b>IP</b>	Internet Protocol
<b>IPaaS</b>	Integration platform as a service
<b>JAR</b>	Java archive
<b>JAXB</b>	Java Architecture for XML Binding
<b>JCA</b>	Java EE Connector Architecture
<b>JDBC</b>	Java Database Connectivity
<b>JDWP</b>	Java Debug Wire Protocol
<b>JMS</b>	Java Message Service
<b>JNDI</b>	Java Naming and Directory Interface
<b>JSON</b>	JavaScript Object Notation
<b>LDAP</b>	Lightweight Directory Access Protocol
<b>MMA</b>	Minimum, Maximum, Average
<b>MQI</b>	Message queue interface
<b>MQMD</b>	MQ message descriptor
<b>MQTT</b>	MQ Telemetry Transport
<b>ODBC</b>	Open Database Connectivity
<b>OGF</b>	Open Grid Forum (OGF)
<b>OPC</b>	Open platforms communication
<b>PaaS</b>	Platform as a service
<b>POP</b>	Post Office Protocol
<b>PoS</b>	Point of sales
<b>RCP</b>	Rich Client Platform
<b>RCS</b>	Revision Control System
<b>REST</b>	Representational State Transfer
<b>RRS</b>	Resource Recovery Services
<b>SCA</b>	Service component architecture)
<b>SCCS</b>	Source Code Control System
<b>SDSF</b>	System Display and Search Facility
<b>SFTP</b>	Secure File Transfer protocol

---

<b>SMF</b>	System Management Facilities
<b>SMTP</b>	Simple mail transfer protocol
<b>SOAP</b>	A lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. SOAP can be used to query and return information and invoke services across the Internet.
<b>SQL</b>	Structured Query Language
<b>SSL</b>	Secure sockets layer
<b>TCB</b>	task control blocks
<b>TCP</b>	Transfer control protocol
<b>TDS</b>	Tagged delimited syntax
<b>UDDI</b>	Universal Description, Discovery, and Integration
<b>UDP</b>	User-defined property
<b>URI</b>	Uniform Resource Identifier
<b>URL</b>	Universal resource locator
<b>USB</b>	Universal Serial Bus
<b>UUID</b>	Universally unique identifier
<b>W3C</b>	World Wide Web Consortium
<b>WSDL</b>	Web service description language
<b>XA</b>	extended architecture
<b>XML</b>	Extensible Markup Language
<b>XPath</b>	XML Path
<b>XSD</b>	XML schema document
<b>XSDL</b>	XML schema Definition Language
<b>XSL</b>	Extensible Stylesheet Language
<b>zFS</b>	z/OS file system



# Appendix C. Resource guide

Completing this IBM Training course is a great first step in building your IBM Middleware skills. Beyond this course, IBM offers several resources to keep your Middleware skills on the cutting edge. Resources available to you range from product documentation to support websites and social media websites.

## Training

- **IBM Training website**
  - Bookmark the IBM Training website for easy access to the full listing of IBM training curricula. The website also features training paths to help you select your next course and available certifications.
  - For more information, see: <http://www.ibm.com/training>
- **IBM Training News**
  - Review or subscribe to updates from IBM and its training partners.
  - For more information, see: <http://bit.ly/IBMTrafficEN>
- **IBM Certification**
  - Demonstrate your mastery of IBM Middleware to your employer or clients through IBM Professional Certification. Middleware certifications are available for developers, administrators, and business analysts.
  - For more information, see: <http://www.ibm.com/certify>
- **Training paths**
  - Find your next course easily with IBM training paths. Training paths provide a visual flow-chart style representation of training for many IBM products and roles, including developers and administrators.
  - For more information, see:  
<http://www-304.ibm.com/jct03001c/services/learning/ites.wss/us/en?pageType=page&c=a0003096>

## Social media links

Connect with IBM Middleware Education and IBM Training, and learn about the latest courses, certifications, and special offers by seeing any of the following social media websites.

- **Twitter**
  - Receive concise updates from Middleware Education a few times each week.

- Follow Middleware Education at: [twitter.com/websphere\\_edu](http://twitter.com/websphere_edu)
- **Facebook:**
  - Follow IBM Training on Facebook to keep in sync with the latest news and career trends, and to post questions or comments.
  - Find IBM Training at: [facebook.com/ibmtraining](http://facebook.com/ibmtraining)
- **YouTube:**
  - See the IBM Training YouTube channel to learn about IBM training programs and courses.
  - Find IBM Training at: [youtube.com/IBMTutorial](http://youtube.com/IBMTutorial)

## Support

- **Middleware Support portal**
  - The Middleware Support website provides access to a portfolio of downloadable support tools, including troubleshooting utilities, product updates, drivers, and Authorized Program Analysis Reports (APARS). The Middleware Support website also provides links to online Middleware communities and forums for collaboratively solving issues. You can now customize the IBM Support website by adding or deleting portlets to show the most important information for the IBM products that you work with.
  - For more information, see:  
<http://www.ibm.com/software/websphere/support>
- **IBM Support Assistant**
  - The IBM Support Assistant is a local serviceability workbench that makes it easier and faster for you to resolve software product issues. It includes a desktop search component that searches multiple IBM and non-IBM locations concurrently and returns the results in a single window, all within IBM Support Assistant.
  - IBM Support Assistant includes a built-in capability to submit service requests; it automatically collects key problem information and transmits it directly to your IBM support representative.
  - For more information, see: <http://www.ibm.com/software/support/isa>
- **IBM Education Assistant**
  - IBM Education Assistant is a collection of multimedia modules that are designed to help you gain a basic understanding of IBM software products and use them more effectively. The presentations, demonstrations, and tutorials that are part of the IBM Education Assistant are an ideal refresher for what you learned in your IBM Training course.

- For more information, see:  
<http://www.ibm.com/software/info/education/assistant/>

## Middleware documentation and tips

- **IBM Redbooks**
  - The IBM International Technical Support Organization develops and publishes IBM Redbooks publications. IBM Redbooks are downloadable PDF files that describe installation and implementation experiences, typical solution scenarios, and step-by-step “how-to” guidelines for many Middleware products. Often, Redbooks include sample code and other support materials available as downloads from the site.
  - For more information, see: <http://www.ibm.com/redbooks>
- **IBM documentation and libraries**
  - IBM Knowledge Centers and product libraries provide an online interface for finding technical information on a particular product, offering, or product solution. The IBM Knowledge Centers and libraries include various types of documentation, including white papers, podcasts, webcasts, release notes, evaluation guides, and other resources to help you plan, install, configure, use, tune, monitor, troubleshoot, and maintain Middleware products. The Knowledge Center and library are located conveniently in the left navigation on product web pages.
- **developerWorks**
  - IBM developerWorks is the web-based professional network and technical resource for millions of developers, IT professionals, and students worldwide. IBM developerWorks provides an extensive, easy-to-search technical library to help you get up to speed on the most critical technologies that affect your profession. Among its many resources, developerWorks includes how-to articles, tutorials, skill kits, trial code, demonstrations, and podcasts. In addition to the Middleware zone, developerWorks also includes content areas for Java, SOA, web services, and XML.
  - For more information, see: <http://www.ibm.com/developerworks>

## Services

- IBM Software Services for Middleware are a team of highly skilled consultants with broad architectural knowledge, deep technical skills, expertise on suggested practices, and close ties with IBM research and development labs. The Middleware Services team offers skills transfer, implementation, migration, architecture, and design services, plus customized workshops. Through a worldwide network of services specialists, IBM Software Service for Middleware

makes it easy for you to design, build, test, and deploy solutions, helping you to become an on-demand business.

- For more information, see:

<http://www-935.ibm.com/services/us/en/it-services/systems/middleware-services/>

# Bibliography

## Web URLs:

IBM Knowledge Center for IBM Integration Bus V10:

[http://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)

IBM Integration Bus DeveloperWorks:

<http://www.ibm.com/developerworks/websphere/zones/businessintegration/wmb.html>





**IBM**  
®