TechDoc
WebSphere Message Broker / IBM Integration Bus

Routing Palette Nodes:
Route to Label, Label, Filter, Route,
Publication, Collector, Sequence, &
Resequence

(Message Flow Development)
# Table of Contents

Introduction ........................................................................................................................................... 3
Document Version .................................................................................................................................. 3
Product Naming History .................................................................................................................... 3
Product Component Terminology ....................................................................................................... 4
Message Flow Routing ...................................................................................................................... 5
Overview ............................................................................................................................................... 5
Filter Node ........................................................................................................................................... 5
ESQL Requirements ............................................................................................................................ 5
Route to Label and Label Nodes ......................................................................................................... 6
Publication Node .................................................................................................................................... 7
Compute Node Requirements .............................................................................................................. 8
Using the Publication Node ................................................................................................................ 9
Route Node ........................................................................................................................................... 9
XPath Expressions .............................................................................................................................. 9
Using the Route Node .......................................................................................................................... 9
Collector Node ..................................................................................................................................... 9
Sequence Node ..................................................................................................................................... 9
Resequence Node ................................................................................................................................ 12
Best Practices ......................................................................................................................................... 13
References ........................................................................................................................................... 14
Introduction

Document Version
This document describes how to utilize the routing Nodes within a Message Broker (WMB up to v8.x) or Integration Node (IIB v9.0 +) Message Flow. The Message Flow “Routing” Nodes are:

- Filter
- Label
- RouteToLabel
- Route
- Publication
- Collector
- Sequence
- Resequence

This document should, however, apply to most versions of these products. The contents of this document have been specifically verified on the following production versions:

- WebSphere Message Broker v7.0.0.2
- IBM Integration Bus v9.0.0.0

This documentation has been created and maintained by:

- Glen Brumbaugh

This documentation has been reviewed by:

- Glen Brumbaugh

This documentation was last updated:

- Version 1.0 April 2015

Product Naming History
The product currently known as IBM Integration Bus has been through a number of different product names during its several decade long evolution. The product was originally developed by the New Era of Networks (NEON) Corporation and was marketed and resold by IBM. IBM completely redesigned and rebuilt the product and released their own in-house developed product beginning with version 2.0. The product has had the following names and version numbers:

- MQSeries Integrator (MQSI) Version 1.0 – 2.0
- WebSphere MQSeries Integrator Version 2.1
- WebSphere Business Integration Message Broker (WBIMB) V5.0
For the remainder of this document, the product will be referred to as “Message Broker”. This is both for historical reasons and to signify that this documentation applies to both the WMB and IIB product versions.

**Product Component Terminology**

With the Version 9.0 product rename (to IBM Integration Bus), several key product architectural components were given new names; while continuing to fill virtually the same role they had previously filled. This documentation will continue to refer to the “old” names because the steps documented here refer to both old and new product versions.

The old and corresponding new names are as follows:

- Message Broker → Now called “Integration Node” (Beginning with v9.0)
- Execution Group → Now called “Integration Server” (Beginning with v9.0)
- Message Flow → Still called “Message Flow”
Message Flow Routing

Overview
There are a number of different Message Flow nodes used for routing. Each of the nodes in the “Routing” palette are described in the following Sections. Note that some of these nodes are NOT recommended for general use within New Message flows. Recommendations are provided for each node.

Filter Node
The “Filter” node is the simplest of the routing nodes and provides a simple “If-Then-Else” routing mechanism within a Message Flow. The “Filter” node is an ESQL node. The ESQL executed in this node must terminate with an ESQL RETURN statement. This statement must return a Boolean value. The route taken from this node depends upon the value of the RETURN statement.

This node has four output terminals. These terminals are:
- Failure (A failure in the ESQL code)
- True (ESQL code returns a “True” value)
- False (ESQL code returns a “False” value)
- Unknown (ESQL code returns a “NULL” value or does not “RETURN” a boolean)

Explicit connections are made between these output terminals and the input terminals of subsequent nodes. The output terminal to which the message is propagated depends solely on the ESQL RETURN expression. The ESQL code within this node may refer to a database. If it does, the Database is identified as a node property (‘Data Source’).

The “Filter” node assumes that its ESQL module will have a name if the following format:
- messageFlowName_Filter (“messageFlowName” is the name of the Message Flow)

If the ESQL module is given another name, then that module name must be specified as a node property (‘Filter Expression’).

ESQL Requirements
The ESQL code used within the “Filter” node must be defined as a “Filter” module and must return a Boolean value. The sample code shown below illustrates the basic structure of the required ESQL “Filter” module.

ESQL Code

CREATE FILTER MODULE flowName_Filter

CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
   -- YOUR ESQL code goes here. Return “TRUE” or “FALSE”
   RETURN TRUE;
END;

END MODULE;

Note: The “Filter” node propagates an unchanged input message to an output terminal. There is thus only one message to refer to and the “Input” and “Output” correlation names are therefore not used.
Route to Label and Label Nodes

After the “Filter” node, the “Route to Label” and “Label” nodes are the next simplest of the routing nodes. Where the “Filter” node implements an “If-Then-Else” structure, the “Route to Label” node implements a “Case” structure, allowing as many branching options as necessary to be defined. The “Route to Label” node performs this routing based upon information contained in the LocalEnvironment portion of the input Message. In order to populate this data, the “Route to Label” node must be preceded in the Message Flow by one of the “Compute” (ESQL, Java, .NET, PHP) nodes that will be responsible for populating the necessary information in the LocalEnvironment portion of the Message Tree.

The “Route to Label” node does NOT use explicit connections from its output terminals to another nodes input terminal. Due to this, the “Route to Label” node only has a single output terminal. This terminal is:

- Failure

(Unable to either identify a label or find a matching node)

The “Route to Label” node will dynamically propagate the message to the “Label” node which has a “Label Name” property that matches the corresponding information contained within the LocalEnvironment. The inbound message can only be propagated to a single “Label” node.

LocalEnvironment

The Label name stored in the LocalEnvironment is stored in an array at a specific Location. The “Route to Label” node can be configured to route to either the “First” or the “Last” element in this array. This configuration is specified through the Mode property of the “Route to Label” node. The Label array is stored in the following location of the LocalEnvironment tree:

- LocalEnvironment.Destination.RouterList.DestinationData[index].LabelName

The string stored in the (LocalEnvironment) LabelName field must match the string stored in the LabelName property of a “Label” node in order for the routing to take place.

Note: The “Route to Label” node consumes either the first or last element of the DestinationData array, depending upon the Mode property of the node! This allows for some complex routing to be specified in the LocalEnvironment, which each “Route to Label” node consuming the next available routing option.

Compute Node

The Compute node must build the DestinationData array within the OutputLocalEnvironment tree. The Tree structure to be built is as follows:

- OutputLocalEnvironment
  - Destination
    - RouterList
      - DestinationData [index]
        - LabelName='label'

As may entries as desired can be placed in the DestinationData array, but the “Route to Label” node will only use the First or the Last entry in the array, depending upon how it is configured. A sample ESQL statement to set this field is as follows:

ESQL Code

```esql
SET OutputLocalEnvironment.Destination.RouterList.
    DestinationData[1].labelName='Label';
```
Note: When you configure the Compute Node, ensure that the “Mode” property is one that includes LocalEnvironment.

Route to Label Node
The “Route to Label” node has no ESQL and only a single property to configure (‘Mode’). This property will be set to “Route to First” or “Route to Last”. This property refers to the element in the DestinationData array that is to be used to determine the name of the destination “Label” node. This node has a terminal, but this should not normally need to be utilized.

Label Node
The “Label” node has a single property (‘Label Name’) that needs to be configured. When using the “Label” node with a “Route to Label” node, the string entered for the Label Name must match the value that is to be placed in the DataDestination array of the LocalEnvironment tree. The “Label” node can also be used independently of the “Route to Label” node!

Any node containing ESQL may propagate the inbound message to a “Label” node. This is done using the ESQL PROPAGATE statement. The parts of the tree to be propagated to the Label may also be specified. An example of using this statement to route a message to a label is as follows:

- PROPAGATE TO LABEL messageSource

The “messageSource” may be any of the following:

- InputLocalEnvironment / OutputLocalEnvironment
- InputRoot / OutputRoot
- InputExceptionList / OutputExceptionList

Since multiple Propagate statements may be coded, this allows “Label” nodes, in conjunction with an ESQL Compute node, to implement a “Fan-Out” pattern. Also, Error Handling flows are often implemented beginning with a Label so that they can be routed to from any ESQL node.

The “SOAPExtract” node is also capable of routing the contents of a SOAP message to a “Label” node. This is done through configuring properties in the “SOAPExtract” node. When used in this manner, the “Label” node must have LabelName field that matches the SOAP Operation.

Publication Node
The “Publication” node performs the role of a “publisher” in a “Publish & Subscribe” messaging architecture. Note that when using modern WebSphere MQ this node is completely unnecessary! A simple “MQOutput” node can be used (pointing to a Queue Alias of a Topic). In fact, this is a superior design since it decouples the application from the routing information. Traditional Message Broker Publish/Subscribe processing embeds the routing information within both the Publisher and Subscriber as a “Topic String”.

This node has two output terminals. No output terminals are necessary for connection with Subscribers, since these connections are done dynamically by the MQ Publish & Subscribe engine. The node’s two output terminals are:

- Out (All publications)
- NoMatch (No subscriber was registered for this publication)

The “Publication” node has only two publishing properties that can be configured and these properties are not normally configured. The two properties are:

- Implicit Stream Naming (Use MQ Input queue name as a Stream name)
Subscription Point

The *Implicit Stream Naming* checkbox allows the name of the MQInput queue to be used as a Stream Name. This is only needed to support Message Flows designed for WMQ v6.x and earlier. In these versions, Publish & Subscribe was handled by WMB, not WMQ, and “Streams” were a WMB concept. Streams essentially act as a root level topic or namespace to separate Publish/Subscribe traffic.

The *Subscription Point* property provides a similar capability to the deprecated “Streams” feature, but for WMQ v7.0 and later. A Subscription point allows the definition of a “Topic Stream” to be narrowed by appending the “Topic Stream” after the “Subscription Point”, with the two being separated by a Forward Slash ('/'). The Subscription Point thus is a way of defining a Parent Topic String. Naturally, the Subscriber must specify the entire resulting Topic String for a successful subscription to take place. The use of either of these fields is not recommend for any new Message Flow development. Instead, the correct Topic Strings should be defined in MQ Topic and Subscription objects.

**Compute Node Requirements**

The "Publication" node must be preceded by a “Compute” node that will build the proper MQRFH2 Header in the Message Tree to contain the information required by the “Publication” node. The ESQL code necessary to construct a working MQRFH2 head for the “Publication” node is as follows:

**ESQL Code**

```sql
DECLARE char_command CHARACTER 'Publish';
DECLARE char_topic CHARACTER 'Topic/Stuff';
DECLARE char_domain CHARACTER 'xmlnsc';

SET OutputRoot.MQMD.Format = MQFMT_RF_HEADER_2;
CREATE NEXTSIBLING OF OutputRoot.MQMD DOMAIN 'MQRFH2' NAME 'MQRFH2';

SET OutputRoot.MQRFH2.(MQRFH2.Field)Version = MQRFH_VERSION_2;
SET OutputRoot.MQRFH2.(MQRFH2.Field)Format = MQFMT_STRING;
SET OutputRoot.MQRFH2.(MQRFH2.Field)Encoding = MQENC_NATIVE;
SET OutputRoot.MQRFH2.(MQRFH2.Field)CodedCharSetId = MQCCSI_INHERIT;
SET OutputRoot.MQRFH2.(MQRFH2.Field)Flags = MQRFH_NO_FLAGS;
SET OutputRoot.MQRFH2.(MQRFH2.Field)NameValueCCSID = 1208;

CREATE LASTCHILD OF OutputRoot.MQRFH2.MQRFH2 DOMAIN 'MQRFH2' NAME 'psc';
SET OutputRoot.MQRFH2.psc.Command = char_command;
SET OutputRoot.MQRFH2.psc.Topic = char_topic;

CREATE LASTCHILD OF OutputRoot.MQRFH2.MQRFH2 DOMAIN 'MQRFH2' NAME 'mcd';
SET OutputRoot.MQRFH2.mcd.MSD = char_domain;

SET OutputRoot.Properties.Topic = char_topic;
```

Note that the “char_topic” and “char_domain” variables will have to be changed to the correct Topic String and Message Domain parser values.
Using the Publication Node

Note: This node is architecturally unsound and its use is not recommended by this author, except for special cases.

There is no reason to tightly couple the delivery paradigm with the message flow, nor to add the complexity of additional ESQL “Compute” nodes to accomplish what is simple configuration in MQ. It is far easier to change the Publish/Subscribe processing within the MQ Layer than to have to modify and redeploy a Message Flow!

The “Publication” node does have one feature than could justify its occasional use. This feature is the “NoMatch” terminal. This node is capable of detecting if there were no Subscribers for the current message and explicitly handling that situation. If this is a business requirement, then this node can be used to satisfy that requirement. For simple publishing, this node should not be used.

Route Node

The “Route” node is arguably one of the most flexible, and potentially one of the easiest to use, of all of the routing nodes. The node was first provided in Message Broker v6.1. The “Route” node executes a series of XPath expressions and makes routing decisions based upon the expression evaluations. The node comes with some built-in output terminals, but additional dynamic output terminals may also be defined. The built-in terminals are as follows:

- Failure (An error was detected within the node)
- Default (Message did not match any XPath expression)
- Match (Dynamic output terminal for use in XPath expressions)

Output terminals may be dynamically added by right-clicking the output terminal and adding a new output terminal. This process also provides the new terminal with a user defined name. A terminal will need to be added for each additional node that needs to be connected to the “Route” node.

Each XPath expression is associated with a dynamic output terminal. The XPath expressions are contained in a node property (“Filter Table”). The expressions are evaluated in the order in which they are specified in the table. The message can either be routed to the output terminal of the first expression that evaluates as “true” or to the output terminals of all expressions that evaluate as “true”. This behavior is controlled by the “Distribution Mode” property (“First” or “All”). These are the only two node properties that need to be configured for this node.

As with most other Message Flow nodes, connections must be drawn of the “Route” terminal and the input terminals of the subsequent nodes. determine which output terminal to use and the connector determines that particular output terminal. See the “MQ Queues”.

Several Routing nodes that process groups of messages use WMQ queues to store messages during processing. These nodes include:

- RESEQUENCE
- COLLECTOR

These nodes commit incoming MQ messages and store the collections in progress as MQ messages on internal Message Broker queues. These messages are stored in the following queues:

- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EDA.COLLECTIONS
The first message of each new collection is written to the “COLLECTIONS” queue. Subsequent messages in the collection are written to the “EVENTS” queue.

**Configurable Service**
These nodes have an associated type of Message Broker Configurable Service. The Configurable Service is associated with the node through a node property (Configurable Service). A Collector Configurable Service can specify the following two node properties:

- Queue Name prefix (SYSTEM.BROKER.EDA.*.prefix.*) forcing node use of separate queues
- Collection Expiry period (in seconds)

**XPath Expressions** Section for additional XPath details.

**Using the Route Node**
While the Route node is conceptually very straightforward, it is not well supported within the Toolkit. There is no debugging support for the XPath expressions themselves. This means that if the node does not perform as expected, the only recourse is to rely upon knowledge of the XPath language and “trial and error” debugging.

**Sequence Node**
The “Sequence” node inserts a sequence number into each message of a messages. The expressions to determine group membership, as well as to determine the sequence number, are both XPath expressions. See the “MQ Queues” Section for additional XPath details.

Several Routing nodes that process groups of messages use MQ queues to store messages during processing. These nodes include:

- RESEQUENCE
- COLLECTOR

These nodes commit incoming MQ messages and store the collections in progress as MQ messages on internal Message Broker queues. These messages are stored in the following queues:

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

The first message of each new collection is written to the “COLLECTIONS” queue. Subsequent messages in the collection are written to the “EVENTS” queue.

**Configurable Service**
These nodes have an associated type of Message Broker Configurable Service. The Configurable Service is associated with the node through a node property (Configurable Service). A Collector Configurable Service can specify the following two node properties:

- Queue Name prefix (SYSTEM.BROKER.EDA.*.prefix.*) forcing node use of separate queues
- Collection Expiry period (in seconds)

**XPath Expressions**’ Section for a summary of XPath syntax. The sequence number must be an integer number, but the initial sequence number value for a group may be specified as a node property (all groups must begin with the same sequence number).

A new group, beginning with the specified initial sequence number, is begun when a new group identifier, specified in the XPath Group Identifier expression, is encountered. The last message in
the group can be determined in three different ways. These three mechanisms, all specified as node properties, are as follows:

- Identifying the final sequence number.
- An XML predicate expression that returns a boolean value of ‘True’.
- Expiration time period reached (specified in seconds).

The “Sequence” node has a single input terminal (‘In’) and two output terminals: ‘Out’ and ‘Failure’. These terminals are summarized as follows:

- In (Input message)
- Out (Output message)
- Failure (Failure within node and/or XPath expression failure)

Resequence Node

The “Resequence” node sorts a group of incoming messages by their “Sequence” node, the expressions to determine group membership, as well as the location of the sequence number, are both XPath expressions. See the “MQ Queues” Section for a summary of XPath syntax. Note that the expression for the sequence number may either be a path or an expression that computes an integer sequence number. This sequence number does not have to have been created by a “Sequence” node, but the Sequence numbers must be incremented in steps of 1, with no gaps.

Note: THIS IS A “BLOCKING” Node. An input message will not be processed until its sequence number has been reached.

Note: THIS NODE COMMITS ANY EXISTING TRANSACTION FOR AN INBOUND MESSAGE. All incoming message are written, and committed, to internal MQ queues before being processed.
A new group is begun when a new group identifier, specified in the XPath Sequence Group Identifier expression, is encountered. Both the first and last sequence numbers in the group can be determined in three different ways. These three mechanisms, all specified as node properties, are as follows:

- Identifying the initial or final sequence numbers (integers).
- An XML predicate expression that returns a boolean value of ‘True’.
- Expiration time period reached (specified in seconds); all messages blocked until expiry.

The “Resequence” node has an important node property (Missing Message Timeout) than can affect how input messages are processed. This property specifies how long (in seconds) the node should wait for the next message.

The “Resequence” node has a single input terminal (‘In’) and multiple output terminals. These terminals are summarized as follows:

- In (Input message)
- Out (Output message)
- Missing (Next message arrives after Missing Message Timeout interval)
- Expire (All messages in group sequenced after a “missing” message)
- Failure (Failure within node and/or XPath expression failure)
- Catch (Catch a downstream failure)

Collector Node
The “Collector” node implements a “Fan-In” message flow. This node is thus similar in function to the Aggregation “AggregateReply” node (See the Aggregation (Parallel Processing) companion TechDoc for more complete information). Since this node combines multiple incoming messages into a single Message Tree, the output of this node can only be processed by a “Compute” (ESQL, Java, .NET, or PHP) type of node.

Input messages can arrive on one or more dynamically defined input terminals. Dynamic input terminals can be added by right-clicking on the node and selecting “Add Input Terminal”. Each of these terminals has a unique name and can have a different mechanism for determining whether an input message is a member of an existing collection or a member of a new collection. Collection membership can be based upon a number of different criteria. These criteria include:

- Matching the contents specified by a correlation path
- Number of input messages
- Time duration

A new collection is started whenever an input message arrives that does not meet the criteria for inclusion in an existing collection that is in progress. Collections being processed are stored in internal MQ queues. A collection is completed when the completion criteria for every input terminal has been satisfied.

Collections are routed to the output terminal based upon the node Event Coordination property. Collections can either be released as soon as they are complete or they can be held until a message is received on the input Control terminal. If control message are used, they may either release the oldest or all completed collections.

Note: THIS IS A”BLOCKING” Node. An input message will not be processed until the collection is complete.
Note: THIS NODE COMMITS ANY EXISTING TRANSACTION FOR AN INBOUND MESSAGE. All incoming messages are written, and committed, to an internal queue before being processed.

In addition to the dynamic input terminals previously mentioned (added by right clicking on the node in the Toolkit), the “Collector” node has a single (static) input terminal (‘Control’) and four output terminals. The static terminals are as follows:

- **Control** (Trigger completion of an in-progress collection)
- **Output** (Output message containing complete collection)
- **Failure** (Internal node or XPath error)
- **Expire** (Output message with an incomplete collection; terminal “Timeout reached”)
- **Catch** (Downstream error caught and rolled-back to this node)

Collections can be completed in the following ways:

- Specifying the number of messages in a collection for the terminal
- Specifying an expiration interval for the terminal
- Sending a message to the control terminal (See *Event Coordination* property)

The Message Tree in the *Out* terminal has the following structure:
MQ Queues
Several Routing nodes that process groups of messages use WMQ queues to store messages during processing. These nodes include:

- RESEQUENCE
- COLLECTOR

These nodes commit incoming MQ messages and store the collections in progress as MQ messages on internal Message Broker queues. These messages are stored in the following queues:

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

The first message of each new collection is written to the “COLLECTIONS” queue. Subsequent messages in the collection are written to the “EVENTS” queue.
Configurable Service
These nodes have an associated type of Message Broker Configurable Service. The Configurable Service is associated with the node through a node property (Configurable Service). A Collector Configurable Service can specify the following two node properties:

- Queue Name prefix (SYSTEM.BROKER.EDA.prefix.*) forcing node use of separate queues
- Collection Expiry period (in seconds)

XPath Expressions
The Message Broker Toolkit contains an XPath expression editor to help construct XPath expressions. When using the editor, the XPath separator (“/”) must be entered at the end of a node before the editor will supply additional prompts. The editor is somewhat useful, but it requires a basic knowledge of XPath syntax.

Note: If the XPath expression uses Namespace prefixes, then the editor must be used, as this is the only way that the Namespace prefix can be linked to the XPath expression!

The following is a summary of XPath syntax that should prove useful in constructing expressions:

- XPath expressions can yield a set of nodes, a string, a number, or a Boolean. For use in the “Route” node, they must yield a Boolean result.
- Unlike ESQL, which uses a period (“.”) to separate nodes in an expression, XPath uses a forward slash (“/”).
- XPath has some shortcut & wildcard capabilities:
  - A single period (“.”) represents the current node.
  - A double period ("..") represents the parent node.
  - An asterisk ("*") represents a single node.
  - An “At Sign” asterisk combination (“@*”) represents any attribute of a node.
  - Two forward slashes (“//”) represents zero or more intervening nodes.
- Predicates are placed inside brackets ("[" and "]"). The simplest predicate is an index number. Predicates may be expressions and include functions. Predicates are used to further qualify the element that they follow.
- Indexes in XPath begin numbering with “1” (not zero).
- There are numerous functions in XPath. There are String, Numeric, Boolean, Conversion, and Namespace functions. These are listed in the editor.
- There are a number of operators in XPath. These are also listed in the editor.

Note: XPATH expressions are not well supported within the Message Broker Toolkit. The Toolkit does not provide interactive debugging support and incorrect expressions do not “fail”, they simply work differently than expected. It is recommended that you have sufficient XPATH knowledge and the necessary eternal testing tools if you plan to use these expressions.

Best Practices

- Use of the Label Node: Consider using the “Label” node, along with ESQL PROPAGATE statements, as a common routing point for Error Handling and/or Logging functions.
• **Use of the Route Node:** The “Route” node should only be used if both the development and maintenance staffs have a solid understanding of XPath expressions. This is due both to the lack of debugging support with the Toolkit for XPath expressions as well as the fact that these expressions do not always behave as they were envisioned. In this respect, the XPath expressions are similar to UNIX Regular Expressions (RegEx) and require thorough testing.

• **Use of the Publication Node:** The “Publication” node should not be used in any new Message Flow development. The necessary processing can be accomplished far more simply, and be far easier to maintain, when implemented administratively within the MQ layer. Write to an “MQOutput” node and have that node be a Queue Alias pointing to a Topic.

**References**

- IBM – Integration Bus – Knowledge center (v9.0) - Routing

- IBM – Integration Bus – Knowledge center (v9.0) – XPath Expressions