Cataloging WebSphere MQ Applications in WSDL for Reuse in SOA

This IBM® Redpaper discusses the following specifications for IBM WebSphere® MQ:

- WebSphere MQ Service Definition Specification, which can be used to describe WebSphere MQ applications as services for use in SOA
- Its companion document, the WebSphere MQ IRI Specification

These specifications are now available as SupportPac™ MA93 and can be downloaded from the SupportPac site:


This IBM Redpaper includes an example scenario that illustrates how to:

- Use these specifications to define SOAP and non-SOAP WebSphere applications as services.
- Register these services to WebSphere Service Registry and Repository.
- Access the registered services from a message flow in WebSphere Message Broker.

WebSphere MQ Service Definition Specification

The WebSphere MQ Service Definition Specification and its companion document, the WebSphere MQ IRI Specification, specify how to describe WebSphere MQ applications as services and how to send messages over WebSphere MQ in a standardized way.

Note: For brevity, WebSphere MQ is often abbreviated as WMQ.

The service definition specification defines the WMQ service binding, which defines the properties and rules that are needed to describe a WebSphere MQ application as a service. It covers both SOAP and non-SOAP WebSphere MQ messages and the MQI structures and
APIs that must be used to send the messages. Specifically, it defines how a WebSphere MQ service implementation should:

- Connect to a queue manager and select the appropriate message destinations.
- Handle message properties and header fields.
- Handle the message content and properties for SOAP.

It also defines the message exchange patterns that the WMQ service binding supports.

WebSphere MQ IRI Specification V1.0 defines the format of Internationalized Resource Identifiers (IRIs), as defined in RFC3987, for identifying addressable resources that are used in WebSphere MQ. It is primarily used with the WebSphere MQ Service Definition Specification but should be applicable wherever WebSphere MQ IRI is needed to describe WebSphere MQ resources. This newly defined IRI scheme is called the **wmq IRI**.

**IRI specification**

The wmq IRI scheme helps identify and address WebSphere MQ queues and topics. The WebSphere MQ Service Definition Specification uses it to describe WebSphere MQ applications as services in a service-oriented architecture (SOA).

The basic syntax of the wmq IRI scheme is:

```
wmq:/*wmq-dest
```

where `wmq-dest` is one of the following options:

- `msg/queue/queue_name`
- `msg/topic/topic_name`

Example 1 shows the full syntax of the wmq IRI scheme.

**Example 1  wmq IRI syntax**

```
wmq-iri = "wmq:" [ "//" connection-name ] "/" wmq-dest ["?" parm *("&" parm)]

connection-name = tcp-connection-name / other-connection-name
    tcp-connection-name = ihost [ ":" port ]
    other-connection-name = 1*(iunreserved / pct-encoded)

wmq-dest = queue-dest / topic-dest
    queue-dest = "msg/queue/" wmq-queue ["@" wmq-qmgr]
        wmq-queue = wmq-name
        wmq-qmgr = wmq-name
        wmq-name = 1*48( wmq-char )
    topic-dest = "msg/topic/" wmq-topic
        wmq-topic = segment *( ":" segment )
        segment = 1*(iunreserved / pct-encoded)

parm = parm-name "=" parm-value
    parm-name = 1*(iunreserved / pct-encoded)
    parm-value = *(iunreserved / pct-encoded)

wmq-char = ALPHA / DIGIT / "." / ":" / %x2F / %x25 ; Encode "/" and "%"
    ihost = ; see [RFC3987]
    port = ; see [RFC3987]
    iunreserved = ; see [RFC3987]
    pct-encoded = ; see [RFC3986]
    ALPHA = ; see [RFC4234]
    DIGIT = ; see [RFC4234]
```
Table 1 contains examples of IRI specifications for WebSphere MQ resources.

Table 1  IRI messaging resource specifications

<table>
<thead>
<tr>
<th>Resource type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple queue address on the local default queue manager</td>
<td><code>wmq:/msg/queue/INS.QUOTE.REQUEST</code></td>
</tr>
<tr>
<td>Qualified queue address (default to remote queue manager MOTOR.INS)</td>
<td><code>wmq:/msg/queue/INS.QUOTE.REQUEST@MOTOR.INS</code></td>
</tr>
<tr>
<td>Simple queue address through a connection to named queue manager</td>
<td><code>wmq:/msg/queue/INS.QUOTE.REQUEST?connectQueueManager=MOTOR.INS</code></td>
</tr>
<tr>
<td>Queue address through a TCP/IP client connection</td>
<td><code>wmq://insurance.hursley.ibm.com:1415/msg/queue/INS.QUOTE.REQUEST?transportType=&quot;TCP&quot;&amp;channelName=&quot;SYSTEM.DEFAULT.SVRCONN&quot;</code></td>
</tr>
</tbody>
</table>

**WMQ service binding properties**

The WMQ service binding defines properties that can be used in wmq IRI or WSDL annotations. These properties specify connection details, message exchange patterns, message header fields, and message content.

The format of the properties depends on where they are used:

- When properties are included in the IRI, they are specified as part of the IRI query string. A question mark separates them from the destination and then they appear as name/value pairs that are separated by an ampersand.
- When properties are included in the WSDL, they appear as XML extension elements that are allowed in several places in the WSDL.

The following locations are checked for these properties:

- wmq IRI. The wmq IRI can be specified programmatically, be included as the location address IRI in a WSDL document, come from a WS-Addressing endpoint reference, or come from the environment.
- WSDL elements or attributes, including the location address IRI.

Some properties have restrictions on where they can be placed.

**Native MQ binding properties**

The native MQ binding properties affect how the binding behaves regardless of the message payload format. They are used to define:

- The connection information necessary to connect to a WebSphere MQ destination, which includes the queue manager name and queue or topic name. For example, the following binding properties fit this category:
  - `wmqservice:destinationName`
  - `wmqservice:connectionQueueManager`
- The client connection information necessary to connect to a queue manager in client connection mode. For example, the following binding properties fit this category:
  - `wmqservice:channelName`
  - `wmqservice:connectionName`
MQMD header fields, such as message data format, message type (request/reply), replyTo destination, message ID, correlation ID, and quality of service information, including priority, persistence, and so forth. For example, the following binding properties fit this category:

- `wmqservice:msgId`
- `wmqservice:correlId`
- `wmqservice:persistence`
- `wmqservice:replyTo`
- `wmqservice:format`

The `wmqservice:format` property defines the format of the message payload. For non-SOAP messages this can be set to any value.

For SOAP messages, this property defines the encoding of the SOAP payload:

- **MQSOAP11** for XML SOAP 1.1 messages. This is equivalent to setting the `wmqservice:contentType` property to `text/xml`.
- **MQSOAP12** for XML SOAP 1.2 messages. This is equivalent to setting the `wmqservice:contentType` property to `application/soap+xml`.
- **MQSWA11** for SOAP with Attachments messages. This is equivalent to setting the `wmqservice:contentType` property to `multipart/related; type="text/xml"`.
- **MQSWA12** for SOAP with Attachments messages. This is equivalent to setting the `wmqservice:contentType` property to `multipart/related; type="application/soap+xml"`.
- **MQMTOM11** for MTOM messages. This is equivalent to setting the `wmqservice:contentType` property to `multipart/related; type="application/xop+xml"; start-info="text/xml"`.
- **MQMTOM12** for MTOM messages. This is equivalent to setting the `wmqservice:contentType` property to `multipart/related; type="application/xop+xml" start-info="application/soap+xml"`.

**RFH2** header fields. For example, the following binding properties fit this category:

- `wmqservice:targetAction`
- `wmqservice:bindingVersion`

**User properties**

**MQ SOAP binding properties**

The SOAP binding properties provide additional information in the RFH2 header for message headers and properties when the message payload is in SOAP format (as defined by the `wmqservice:format` property or the WSDL binding).

Additional RFH2 message properties for SOAP messages are:

- `wmqservice:contentType`
  
  This allows additional information about the MIME type.

- `wmqservice:soapAction`
  
  This provides a URI identifying the intent of the request (the same function as the `soapAction` for SOAP/HTTP).
Guidance on specifying properties
With properties, the IRI can be used as a simple address or as a complex and complete service description. For example:

wmq://insurance.hursley.ibm.com:1415/msg/queue/INS.QUOTE.REQUEST@MOTOR.INS?
ReplyTo=msg/queue/INS.QUOTE.REPLY@BRANCH452&
persistence=MQPER_NOT_PERSISTENT&targetAction=GetQuote

However, it is recommended that the information in the IRI be limited to minimal addressing data, for the following reasons:

- Over-specifying the IRI reduces flexibility. For example, the example IRI confines the service to a host called insurance.hursley.ibm.com. It is better to let the queue manager or service consumer make decisions about routing.
- The IRI is usually treated as an opaque string so it might not be possible to search the contents of the IRI, for example, from WebSphere Service Registry and Repository. However, properties that are stored in WSDL extensions are searchable.

Message exchange patterns

The WebSphere MQ Service Definition Specification describes the messaging exchange patterns that must be supported by implementations of the specification and the behavior of the requesting and responding nodes in the pattern.

A WebSphere MQ service binding implementation must support request-response and one-way message exchange patterns. These are described in the specification for implementations using native MQ bindings and those with SOAP 1.1 messages.

If the message payload is SOAP 1.2, the implementation must support the following message exchange patterns:

- [Link](http://www.w3.org/2003/05/soap/mep/request-response/) (see Request-Response Message Exchange Pattern from SOAP 1.2 Part 2)
- [Link](http://www.w3.org/2006/08/soap/mep/one-way/) (see SOAP 1.2 Part 3)

WSDL bindings

Services sending SOAP and non-SOAP messages over WebSphere MQ can be described in WSDL with WSDL 1.1 extensions that are defined in the WebSphere MQ Service Definition Specification.

Extensions for WMQ SOAP bindings
The specification defines the following extensions for use in WMQ SOAP bindings:

- MQ namespace:

  ```xml
  <binding name="Soap_MQ" type="tns:Soap_MQPortType"
  xmlns:wmqservice="http://www.ibm.com/xmlns/prod/wmq/bindings/1.0">
  ```

- Transport attribute that defines WMQ in the <soap:binding>:

  ```xml
  <soap:binding style="document"
  transport="http://www.ibm.com/xmlns/prod/wmq/transport"
  ```

- Service binding properties, such as:

  - `<wmqservice:persistence>MQPER_PERSISTENT</wmqservice:persistence>`
  - `<wmqservice:format>MQSOAP11</wmqservice:format>`
The soapAction attribute in the <soap:operation> element
The extensions allow the use of soapAction in a <soap:operation> element for SOAP/WMQ bindings.

The specification defines the following for use in the WMQ port:

- The <soap:address> specified using the location IRI. For example:
  <wsdl:port binding="yc:UpdatePortBinding" name="UpdatePort">
  <soap:address location="wmq:/msg/queue/OUTPUTQ"/>
  </wsdl:port>

- Service binding properties. For example:
  <wmqservice:replyTo>msg/queue/REPLYQ</wmqservice:replyTo>

Example 2 shows the format of WSDL that is used for SOAP/WMQ messages. Note that the <wmqservice:format> property shown in the example is optional.

**Example 2  WSDL bindings for SOAP/WMQ messages**

```xml
<definitions ...>
  <binding ... xmlns:wmqservice="http://www.ibm.com/xmlns/prod/wmq/bindings/1.0" >
    <soap:binding ...
      <soap:operation soapAction="http ...">
        <input>
          <soap:body use="literal"/>
        </input>
        <output>
          <soap:body use="literal"/>
        </output>
      </soap:operation>
    </soap:binding>
  </binding>
  <port ...
    location="wmq:/msg/queue/QnameReq"/>
  <wmqservice:replyTo>msg/queue/QnameReply</wmqservice:replyTo>
</port>
</definitions>
```

All messages sent in the native WMQ binding should be modeled in XML schema. For messages encoded in non-XML format, you can use XML schema with proprietary schema annotations (such as the broker message set definition).

The Data Format Description Language Working Group (DFDL-WG) is defining a standard for non-XML schema that, once ratified, can describe native MQ messages that standard XML schema cannot describe. For more information, see:

Extensions for WebSphere MQ native bindings (non-SOAP)

The specification defines the following extensions for MQ applications that do not use SOAP messages for use in native WMQ bindings:

- **MQ namespace**

  ```xml
  <wsdl:definitions ...
  xmlns:wmqservice="http://www.ibm.com/xmlns/prod/wmq/bindings/1.0">
  ...
  </wmqservice:binding/>
  ...
  </wmqservice:format>MQSTR</wmqservice:format>
  ...
  </wmqservice:property>...
  </wmqservice:property>
  </operation>
  ...
  </wmqservice:body />
  </input>
  ...
  </wmqservice:body />
  </output>
  </operation>
  </binding>
  ...
  </wmqservice:address location="wmq:/msg/queue/Qname"/>
  ...
  </wmqservice:connectQueueManager>Qmgr</wmqservice:connectQueueManager>
  ...
  </wmqservice:replyTo>msg/queue/QnameReply</wmqservice:replyTo>
  </port>
  </definitions>
```

- **wmqservice:binding element**

  Indicates that the binding uses the native WMQ transport

- **wmqservice:body element**

  Specifies how the message parts appear as parts of the WMQ message body

- **wmqservice:address element**

  Defines the endpoint where the message should be sent. For example:

  ```xml
  <wmqservice:address location="wmq:/msg/queue/Q1"
  ...
  </wmqservice:address>
  ...
  </wmqservice:property>...
  </wmqservice:property>
  ...
  </wmqservice:body />
  </input>
  ...
  </wmqservice:body />
  </output>
  </operation>
  </binding>
  ...
  </wmqservice:address location="wmq:/msg/queue/Qname"/>
  ...
  </wmqservice:connectQueueManager>Qmgr</wmqservice:connectQueueManager>
  ...
  </wmqservice:replyTo>msg/queue/QnameReply</wmqservice:replyTo>
  </port>
  </definitions>
```

- **wmqservice:property element**

  Example 3 shows the format of WSDL bindings for native MQ messages.

**Example 3  WSDL bindings for native MQ messages**

Example scenario

This section explains how you can use the WebSphere MQ Service Definition Specification to allow WebSphere MQ applications to be reused and composed as Web services. It uses a scenario based on a fictitious insurance company to illustrate how to describe two WebSphere MQ services in WebSphere Service Registry and Repository and how to use those registry entries in a WebSphere Message Broker message flow.
**Original architecture**

An insurance company has a distributed Customer Management and Quotation application that uses WebSphere MQ to send messages from operators to back-end systems. This solution (Figure 1) provides two services:

- When a customer’s circumstances change (for example, they move), they can phone or write the company to update their customer record. The Call Center operator records the new customer information and submits it to the Update Customer Records application. The information is sent as a message in the CUSTOMER.UPDATE.INPUTQ queue. This is a one-way (fire and forget) messaging operation using a SOAP formatted message.

- Customers interested in a new insurance policy can call the Call Center for a quote for the insurance premium. The operator takes the customer information and requests a quote from the Calculate Premium application that is sent in the CALCULATE.PREMIUM.INPUTQ queue. The response is returned in the CALCULATE.PREMIUM.OUTPUTQ queue. This request-response messaging operation uses “Plain Old XML” (POX) formatted messages.

![Original solution architecture](image)

This system is working very well, but the company would like customers and mobile staff to be able to use it over the Internet. Their plan is to add an HTTP interface to the applications. They would also like to document and catalog their existing applications.

**New design**

To document their existing applications, the company decides to use the WebSphere MQ Service Definition because they can describe the message formats that the applications use, along with the qualities of service they expect and the queues and queue managers they are deployed to. They decide to store their new service definitions in the WebSphere Service Registry and Repository so that the services can be audited and queried at design time and for runtime routing. To add an HTTP interface to the services, they decide to use WebSphere Message Broker. Figure 2 on page 9 shows the new design for how they plan to develop message flows to handle the service requests.
Clients can submit one of two types of requests: to update a customer record or a rate quote (calculate premium). The port type specified in the request message defines the request type. A specific message flow in WebSphere Message Broker handles each type. Another flow acts as the front-end.

WSDL files describing the customer update and request quote services are published to WebSphere Service Registry and Repository. They use the binding specification in the WMQ Service Definition Specification to define the properties for connecting to the MQ application.

The front-end flow accepts the incoming requests, performs a lookup for the service location, and then calls the service. The messages are either XML or SOAP (Example 4).

**Example 4  Update customer records request SOAP message**

```xml
<?xml version="1.0" encoding="UTF-8" ?>
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <SOAP-ENV:Header>
  <PortType>UpdatePortType</PortType>
 </SOAP-ENV:Header>
 <SOAP-ENV:Body>
  <address>
   <id>1</id>
   <firstName>Joe</firstName>
   <lastName>Silver</lastName>
   <address1>USA</address1>
   <address2>North Carolina</address2>
   <town>Raleigh</town>
   <postCode>2273</postCode>
  </address>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Example 5 shows an XML request message for a rate quote.

```xml
<vehicleDetails>
    <make>Jaguar</make>
    <model>XJ12</model>
    <year>1984</year>
    <cost>1200</cost>
</vehicleDetails>
```

Example 6 shows an XML response message with the rate quote.

```xml
<quote>
    <premiumCost>2000</premiumCost>
</quote>
```

**UpdateCustomer WSDL file**

The UpdateCustomer.wsdl file (Example 7) defines an interface for updating customer address information and defines the following information:

- **Port type** `UpdatePortType`, which is used in the registry lookup.
- **Operation** `updateCustomerAddress`
  
  This is a one-way operation. It has an input of type `Address` complex type.
- **Binding** `UpdatePortBinding`, which specifies that the data will be transmitted using SOAP over MQ.
- **The address of the Web service**, which is set to:
  
  `wmq:/msg/queue/CUSTOMER.UPDATE.INPUTQ`

  This means the request messages will be sent to the `CUSTOMER.UPDATE.INPUTQ` queue.
- **The application uses one-way message exchange** and so there is no `ReplyTo` parameter.

```xml
<wsdl:definitions name="UpdateService" targetNamespace="http://example.org" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:yc="http://example.org"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <wsdl:types>
        <xsd:schema targetNamespace="http://example.org" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:complexType name="Address">
                <xsd:sequence>
                    <xsd:element name="id" type="xsd:string"/>
                    <xsd:element name="firstName" type="xsd:string"/>
                    <xsd:element name="lastName" type="xsd:string"/>
                    <xsd:element name="address1" type="xsd:string"/>
                    <xsd:element name="address2" type="xsd:string"/>
                    <xsd:element name="town" type="xsd:string"/>
                    <xsd:element name="postCode" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:schema>
    </wsdl:types>
```
The RequestQuote.wsdl file (Example 8) defines an interface for requesting a rate quote for a vehicle and defines the following information:

- **Ports** QuotePort and Service QuoteService, which are used in the registry lookup.
- **Operation** getVehicleQuote, which is a request-response operation. It has an input of type VehicleDetails complex type and the output of type Quote complex type.
- **Binding** QuotePortBinding, which specifies that the data will be transmitted using XML over MQ.
- **The address of the Web service**, which is set to:

  `wmq:/msg/queue/CALCULATE.PREMIUM.INPUTQ?ReplyTo=msg/queue/CALCULATE.PREMIUM.OUTPUTQ`

  This means the request messages will be sent to the CALCULATE.PREMIUM.INPUTQ queue, while the response messages will be sent to CALCULATE.PREMIUM.OUTPUTQ queue.

**Example 8  RequestQuote WSDL file**

```xml
<wsdl:definitions name="QuoteService" targetNamespace="http://example.org"
xmlns:wmqservice="http://www.ibm.com/xmlns/prod/wmq/bindings/1.0"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:yc="http://example.org">
  <wsdl:types>
```
<xsd:schema targetNamespace="http://example.org"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="VehicleDetails">
    <xsd:sequence>
      <xsd:element name="make" type="xsd:string"/>
      <xsd:element name="model" type="xsd:string"/>
      <xsd:element name="year" type="xsd:string"/>
      <xsd:element name="cost" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Quote">
    <xsd:sequence>
      <xsd:element name="premiumCost" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

<wsdl:types>
  <wsdl:message name="getVehicleQuoteResponse">
    <wsdl:part name="quote" type="yc:Quote"/>
  </wsdl:message>
  <wsdl:message name="getVehicleQuoteRequest">
    <wsdl:part name="vehicleDetails" type="yc:VehicleDetails"/>
  </wsdl:message>
  <wsdl:portType name="QuotePortType">
    <wsdl:operation name="getVehicleQuote">
      <wsdl:input message="yc:getVehicleQuoteRequest"/>
      <wsdl:output message="yc:getVehicleQuoteResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="QuotePortBinding" type="yc:QuotePortType">
    <wmqservice:binding/>
  </wsdl:binding>
  <wsdl:operation name="getVehicleQuote">
    <wsdl:input>
      <wmqservice:body/>
    </wsdl:input>
    <wsdl:output>
      <wmqservice:body/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
</wsdl:types>

<wsdl:service name="QuoteService">
  <wsdl:port binding="yc:QuotePortBinding" name="QuotePort">
    <wmqservice:address
      location="wmq:/msg/queue/CALCULATE.PREMIUM.INPUTQ?ReplyTo=msg/queue/CALCULATE.PREMIUM.INPUTQ"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
Message flow

The front-end flow (Figure 3) is generic. It routes to any service regardless of whether it is a fire-forget or request-response call and handles the routing of SOAP and XML messages.

```
# HTTPInput node
This node has the XMLNS domain.

# AddMQMD Compute node
This node (Example 9) creates a new message with an MQMD header to create a MQ message to be sent to the services (WebSphere MQ applications).

Example 9  AddMQMD Compute node
DECLARE soapenv NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
CREATE COMPUTE MODULE InsuranceFlow_AddMQMD
  CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN
    SET OutputRoot.Properties = InputRoot.Properties;
    CREATE NEXTSIBLING OF OutputRoot.Properties DOMAIN('MQMD');
    SET OutputRoot.XMLNS = InputRoot.XMLNS;
    SET OutputLocalEnvironment.Destination.HTTP.RequestIdentifier = InputLocalEnvironment.Destination.HTTP.RequestIdentifier;
    RETURN TRUE;
  END;
END MODULE;

# IsSOAP Filter node
This node (Example 10) validates Body.soapenv:Envelope to see if it is a SOAP message.

Example 10  IsSOAP Filter node
CREATE FILTER MODULE InsuranceFlow_IsSOAP
  CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN
    IF Body.soapenv:Envelope IS NOT NULL THEN
      RETURN TRUE;
    ELSE
      RETURN FALSE;
    END IF;
  END;
END MODULE;
```
If Body.soapenv:Envelope has a value, the flow is routed to the ExtractPortType compute node; otherwise, it is routed to the Non-SOAP Endpoint Lookup compute node.

**ExtractPortType Compute node**

This node (Example 11) retrieves the PortType name from the SOAP header of the input message and puts it in the local environment:

OutputLocalEnvironment.ServiceRegistryLookupProperties.Name

The message is passed without alteration.

*Example 11  ExtractPortType Compute node*

```sql
CREATE COMPUTE MODULE InsuranceFlow_ExtractPortType
    CREATE FUNCTION Main() RETURNS BOOLEAN BEGIN
        SET OutputLocalEnvironment = InputLocalEnvironment;
        RETURN TRUE;
    END;
END MODULE;
```

**Note:** Dynamic endpoint look-up at runtime is only available in the WSRR nodes of WebSphere Message Broker V6.1. Using the local environment to override WSRR node properties is not applicable in prior versions.

**Endpoint Lookup node**

This node (Figure 4) is supplied with the Message Broker Toolkit V6.1 and performs the registry search based on the PortType Name field. This name is set to a value of *temp* in the node properties because it will be overridden by the value set in:

OutputLocalEnvironment.ServiceRegistryLookupProperties.Name

When the node finds a matching PortType name, it retrieves the service endpoint from WSDL and places it in LocalEnvironment.Destination.HTTP.RequestURL.

*Figure 4  Endpoint Lookup node*
Non-SOAP Endpoint Lookup JavaCompute node

This node (Example 12) uses the WebSphere Service Registry and Repository V6.1 Web services client for lookup. It retrieves extensions that are associated with the QuotePort port and then gets the value of the address_location user-defined property.

The address_location property is created automatically when the RequestQuote WSDL (Example 8 on page 11) is loaded to the WebSphere Service Registry and Repository. It has the following value:

```
wmq:/msg/queue/CALCULATE.PREMIUM.INPUTQ?ReplyTo=msg/queue/CALCULATE.PREMIUM.OUTPUTQ
```

This value will be placed in LocalEnvironment.Destination.HTTP.RequestURL.

Example 12  Non-SOAP Endpoint Lookup JavaCompute node

```java
import java.net.MalformedURLException;
import java.util.List;
import javax.xml.rpc.ServiceException;
import org.apache.tuscany.sdo.util.SDOUtil;
import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.MbException;
import com.ibm.broker.plugin.MbMessage;
import com.ibm.broker.plugin.MbMessageAssembly;
import com.ibm.broker.plugin.MbOutputTerminal;
import com.ibm.serviceregistry.ServiceRegistryException;
import com.ibm.serviceregistry.sdo.ExtensionLogicalObject;
import com.ibm.serviceregistry.sdo.GraphQuery;
import com.ibm.serviceregistry.sdo.UserDefinedProperty;
import com.ibm.serviceregistry.sdo.helper.DataFactory;
import com.ibm.serviceregistry.sdo.helper.TypeConstants;
import com.ibm.serviceregistry.ws.WSRRCoreSDOClient;

public class NonSOAPEndpointLookup extends MbJavaComputeNode {

    public void evaluate(MbMessageAssembly inAssembly) throws MbException {
        MbOutputTerminal out = getOutputTerminal("out");
        MbOutputTerminal alt = getOutputTerminal("alternate");

        MbMessage inMessage = inAssembly.getMessage();
        // create new empty message
        MbMessage outMessage = new MbMessage(inMessage);
```
// copy the local environment, and the exception list, and then create the output assembly
MbMessage outLocalEnv = new MbMessage(inAssembly.getLocalEnvironment());
MbMessage outExcep = new MbMessage(inAssembly.getExceptionList());
MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly, outLocalEnv, outExcep, outMessage);

String endpoint = null;
try {
    // construct the Web service client to interact with WSRR 6.1
    String url = "http://localhost:9080/WSRR6_1/services/WSRRCoreSDOPort";
    WSRRCoreSDOClient serviceRegistry = new WSRRCoreSDOClient(url);

    // register SDO objects in order for any client to use them
    SDOUtil.registerStaticTypes(com.ibm.serviceregistry.sdo.SdoFactory.class);

    // prepare the graph query
    GraphQuery graphQuery = (GraphQuery) DataFactory.INSTANCE.create(TypeConstants.SR_URI, TypeConstants.TYPE_GRAPHQUERY);

    // retrieve all the extensions in the registry of 'QuotePort' port
    String query = "/WSRR/WSDLService[@name="QuoteService"]/ports[@name="QuotePort"]/extensions";
    graphQuery.setQueryExpression(query);
    List graphQueryResults = serviceRegistry.executeQuery(graphQuery);

    // retrieve the first extension object
    ExtensionLogicalObject extensionLogicalObject = (ExtensionLogicalObject) graphQueryResults.get(0);
    List extensions = extensionLogicalObject.getUserDefinedProperties();

    // retrieve the first user-defined property which will contain the endpoint (property name 'address_location')
    UserDefinedProperty userDefinedProperty = (UserDefinedProperty)extensions.get(0);
    if (userDefinedProperty.getName().equalsIgnoreCase("address_location")) {
        endpoint = userDefinedProperty.getValue();
    } else {
        userDefinedProperty = (UserDefinedProperty)extensions.get(1);
        endpoint = userDefinedProperty.getValue();
    }

    // fill the endpoint in 'LocalEnvironment.Destination.HTTP.RequestURL'
    outLocalEnv.getRootElement().evaluateXPath("?Destination/?HTTP/?^RequestURL[set-value("+
    endpoint+")]"");
} catch (ServiceRegistryException e) {
    // handle exception
} catch (MalformedURLException e) {

// handle exception
} catch (ServiceException e) {
    // handle exception
    out.propagate(outAssembly);
}

Note: The following JAR files must be placed in the shared-classes directory:
- ServiceRegistryClient.jar of WebSphere Service Registry and Repository V6.1
- sdo-int.jar of WebSphere Service Registry and Repository V6.1
- com.ibm.ws.webservices.thinclient_6.1.0.jar of WebSphere Application Server V6.1

ParseURL JavaCompute node
This node (Example 13) parses the URL returned by Endpoint Lookup node to find the name of the destination queue. The node puts the destination queue name in:
LocalEnvironment.Destination.MQDestinationList.DestinationData.queueName

If a ReplyTo exists in the URL, it is set in the ReplyToQ field in the MQMD header and in LocalEnvironment.MQ.GET.QueueName. The MQGet node uses this queue name to fetch the MQ message later in this flow. In the case of a one-way call (customer update), the URL is:
wmq:/msg/queue/CUSTOMER.UPDATE.INPUTQ

In the case of a request-response call (rate quote), the URL will be:
wmq:/msg/queue/CALCULATE.PREMIUM.INPUTQ?ReplyTo=msg/queue/CALCULATE.PREMIUM.OUTPUTQ

Example 13 ParseURL JavaCompute node
import com.ibm.broker.javacompute MbJavaComputeNode;
import com.ibm.broker.plugin.*;

public class URLParser extends MbJavaComputeNode {

    public void evaluate(MbMessageAssembly inAssembly) throws MbException {
        MbOutputTerminal out = getOutputTerminal("out");
        MbOutputTerminal alt = getOutputTerminal("alternate");

        MbMessage inMessage = inAssembly.getMessage();

        // create new message
        MbMessage outMessage = new MbMessage(inMessage);
        MbMessage outLocalEnv = new MbMessage(inAssembly.getLocalEnvironment());
        MbMessage outExcep = new MbMessage(inAssembly.getExceptionList());
        MbMessageAssembly outAssembly = new
        MbMessageAssembly(inAssembly, outLocalEnv, outExcep, outMessage);

        try {
            // Get the request URL from
            //LocalEnvironment.Destination.HTTP.RequestURL'
            String RequestURL = outLocalEnv.getRootElement().getFirstChild().getLastChild().getFirstChild().getValue().toString();
// Parse the request URL to retrieve the queue name in which the
// following MQOutput node will put the MQ message
int queueIndex = RequestURL.indexOf("wmq:/msg/queue/");
int replyToQIndex;
String queueName, replyToQ;

// Check to see if it is a fire-forget or request-response mode, this is
done by checking the presence of 'ReplyToQ' in the request URL
if((replyToQIndex = RequestURL.indexOf("?ReplyTo=msg/queue/"))==-1)
{
    // Get the queue name from the request URL
    queueName = RequestURL.substring(queueIndex+15,RequestURL.length());

    // Set the queue name in
    // 'LocalEnvironment.Destination.MQDestinationList.DestinationData.queueName', this
    // will be used by MQOutput node to put the MQ message to the specified queue
    outLocalEnv.getRootElement().evaluateXPath("Destination/?MQDestinationList/?DestinationData/?@queueName[set-value('"+queueName+"')]");
}
else
{
    // Get the queue name from the request URL
    queueName = RequestURL.substring(queueIndex+15,replyToQIndex);

    // Get the queue name on which the response will arrive
    replyToQ = RequestURL.substring(replyToQIndex+19,RequestURL.length());

    // Set the queue name in
    // 'LocalEnvironment.Destination.MQDestinationList.DestinationData.queueName'
    outLocalEnv.getRootElement().evaluateXPath("Destination/?MQDestinationList/?DestinationData/?@queueName[set-value('"+queueName+"')]");

    // Set the 'ReplyToQ' in 'MQMD.ReplyToQ'
    outMessage.getRootElement().getFirstElementByPath("/MQMD").createElementAsFirstChild(MbElement.TYPE_VALUE, "ReplyToQ", replyToQ);

    // Set 'LocalEnvironment.MQ.GET.QueueName', this will be used by
    // MQGet node to get the MQ message from the specified queue
    outLocalEnv.evaluateXPath("$GET/?@QueueName[set-value('"+replyToQ+"')]");
}
out.propagate(outAssembly);
}

finally {
    // clear the outMessage
    outMessage.clearMessage();
}
}
MQOutput node
This node puts the message in the destination queue specified in the destination list (set earlier by ParseURL JavaCompute node):
LocalEnvironment.Destination.MQDestinationList.DestinationData.queueName

HasReply Filter node
This node (Example 14) checks whether the call is a request-response or one-way call by validating the value of MQMD.ReplyToQ.

If MQMD.ReplyToQ has a value, then the flow is routed to the MQGet node; otherwise, it is routed directly to the HTTPReply node.

Example 14  HasReply Filter node

CREATE FILTER MODULE InsuranceFlow_HasReply
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
  IF Root.MQMD.ReplyToQ IS NOT NULL THEN
    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
END;
END MODULE;

Note: The request-response or fire-forget calls are considered with respect to the message flow.

MQGet node
This node gets the message from the queue specified in the destination list (set earlier by ParseURL JavaCompute node):
LocalEnvironment.MQ.GET.QueueName

DeleteMQMD Compute node
This node (Example 15) propagates the message without the MQMD header.

Example 15  DeleteMQMD Compute node

CREATE COMPUTE MODULE InsuranceFlow_DeleteMQMD
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
  SET OutputRoot.Properties = InputRoot.Properties;
  SET OutputRoot.XMLNS = InputRoot.XMLNS;
  RETURN TRUE;
END;
END MODULE;

HTTPReply node
This node will send the reply to the client. The reply is associated with the correct request using the value in the LocalEnvironment.Destination.HTTP.RequestIdentifier.
The team that wrote this Redpaper

This book was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Raleigh Center.

**Rosaline Makar** is a software engineer at the IBM Cairo Technology Development Center (C-TDC) in Egypt. She holds a Bachelor of Science degree in Computer Engineering from Ain Shams University, Egypt, 2005, and is currently preparing for a Master of Science degree in Computer Science. Her areas of expertise include Web services, WebSphere Integration Developer, WebSphere Process Server, and WebSphere Enterprise Service Bus.

**Mark Phillips** has been working in WebSphere MQ development at the IBM UK Hursley Lab since 1995. He is currently a Senior Software Engineer on the WebSphere MQ Technical Strategy team with responsibility for Web services and Messaging clients. He is studying for a Master of Science degree in Software Engineering.

**Carla Sadtler** is a Consulting IT Specialist at the ITSO, Raleigh Center. She writes extensively about the WebSphere and Patterns for e-business areas. Before joining the ITSO in 1985, Carla worked in the Raleigh branch office as a Program Support Representative. She holds a degree in mathematics from the University of North Carolina at Greensboro.
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
This document REDP-4350-00 was created or updated on January 28, 2008.

Send us your comments in one of the following ways:

- Use the online Contact us review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Redbooks (logo) ®
IBM®
SupportPac™
WebSphere®

The following terms are trademarks of other companies:

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.