



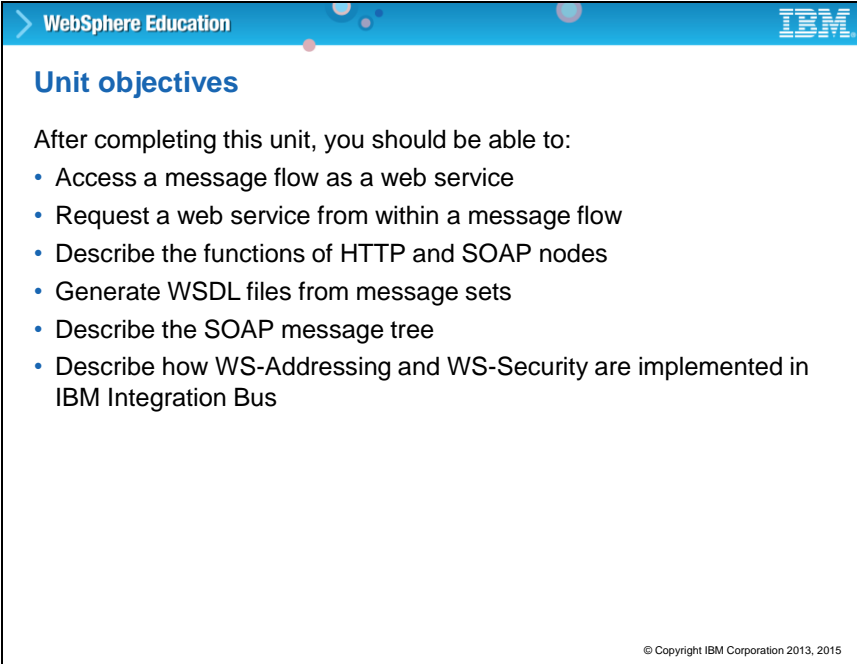
Slide 1

> WebSphere Education 


Supporting web services



© Copyright IBM Corporation 2013, 2015
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



The slide features a blue header bar with the text 'WebSphere Education' on the left and the IBM logo on the right. The main content area is white with a blue title 'Unit objectives'. Below the title, there is a paragraph followed by a bulleted list of six objectives. The footer contains a small copyright notice.

> WebSphere Education 

Unit objectives

After completing this unit, you should be able to:

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

© Copyright IBM Corporation 2013, 2015

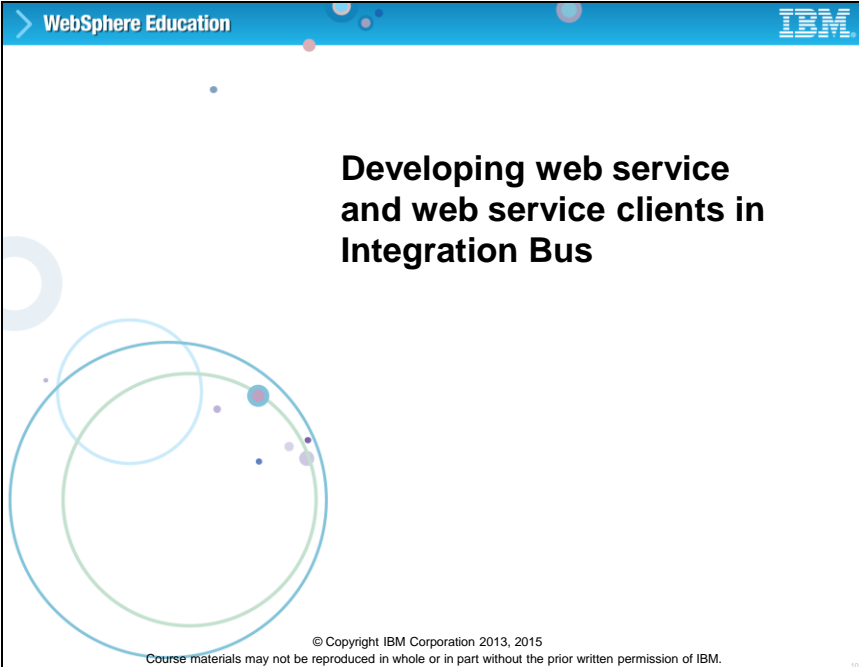
Unit objectives

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

After completing this unit, you should be able to:

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

Slide 3



The slide features a blue header bar with the text 'WebSphere Education' on the left and the IBM logo on the right. The main title, 'Developing web service and web service clients in Integration Bus', is centered in a bold, black font. To the left of the title is a decorative graphic consisting of several overlapping circles in light blue and green, with small colored dots scattered around them. At the bottom of the slide, there is a small copyright notice: '© Copyright IBM Corporation 2013, 2015. Course materials may not be reproduced in whole or in part without the prior written permission of IBM.'

WebSphere Education

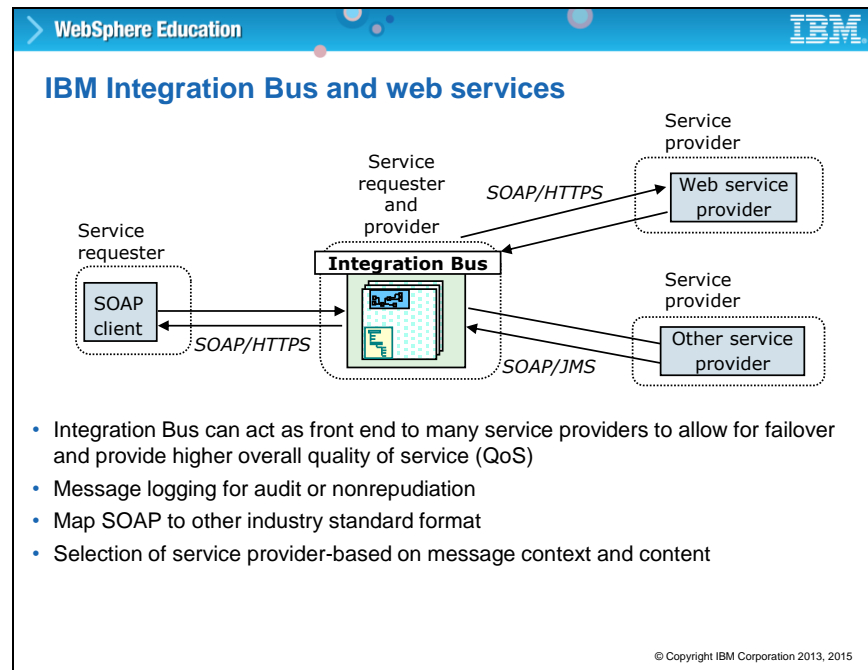
IBM

Developing web service and web service clients in Integration Bus

© Copyright IBM Corporation 2013, 2015
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Topic 1: Developing web service and web service clients in Integration Bus



Web service support in Integration Bus was introduced in the prerequisite course *IBM Integration Bus V10 Application Development I*. This topic reviews web service support in IBM Integration Bus.



IBM Integration Bus and web services

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

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

 WebSphere Education 

Approaches to developing web services

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

© Copyright IBM Corporation 2013, 2015

Approaches to developing web services

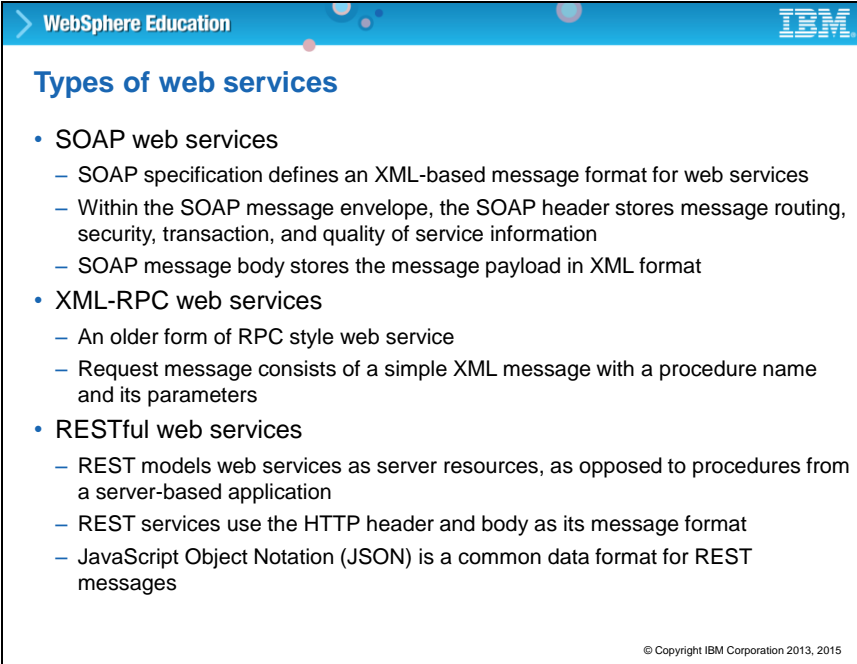
The Integration Toolkit supports three primary approaches to developing web services.

The first approach is to use an integration service. An integration service is a specialized application that acts as a container for SOAP web services that are defined with a WSDL document. Integration services provide a structured solution for quickly building a web service. An integration service implements WSDL web service operations as subflows. The integration service container manages the SOAP message parsing and HTTP message transport options. With this approach, developers focus on implementing business logic, not managing the lower-level message serialization and de-serialization tasks.

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

Last, you can build any type of web service with HTTP nodes. With HTTP nodes, you can build web services that do not use SOAP messages. For example, RESTful web services store the message payload in the HTTP message body itself. To implement a REST service or to call a REST service from a message flow, you must use HTTP nodes.

This unit concentrates on the SOAP and HTTP nodes for web services solutions. Integration services and REST are covered later in this course.



The slide is titled "Types of web services" and is part of a "WebSphere Education" presentation, as indicated by the header. It features a blue header bar with the IBM logo on the right. The content is organized into three main bullet points, each with sub-points. The first bullet point is "SOAP web services", followed by "XML-RPC web services", and then "RESTful web services". The slide concludes with a small copyright notice: "© Copyright IBM Corporation 2013, 2015".

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

© Copyright IBM Corporation 2013, 2015


Types of web services


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

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

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

Representational State Transfer (REST) models web services as server resources, instead of a set of procedures from a server application. REST services use the HTTP header and body as its message format. JavaScript Object Notation (JSON) is a common data format for REST messages. Developing integration solutions by using REST is taught in Unit 12.



WebSphere Education 

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

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



© Copyright IBM Corporation 2013, 2015

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

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

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

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

 WebSphere Education 

HTTP listeners

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

© Copyright IBM Corporation 2013, 2015

HTTP listeners

In Integration Bus, you can choose between an integration node listener or an integration server listener to manage HTTP messages.

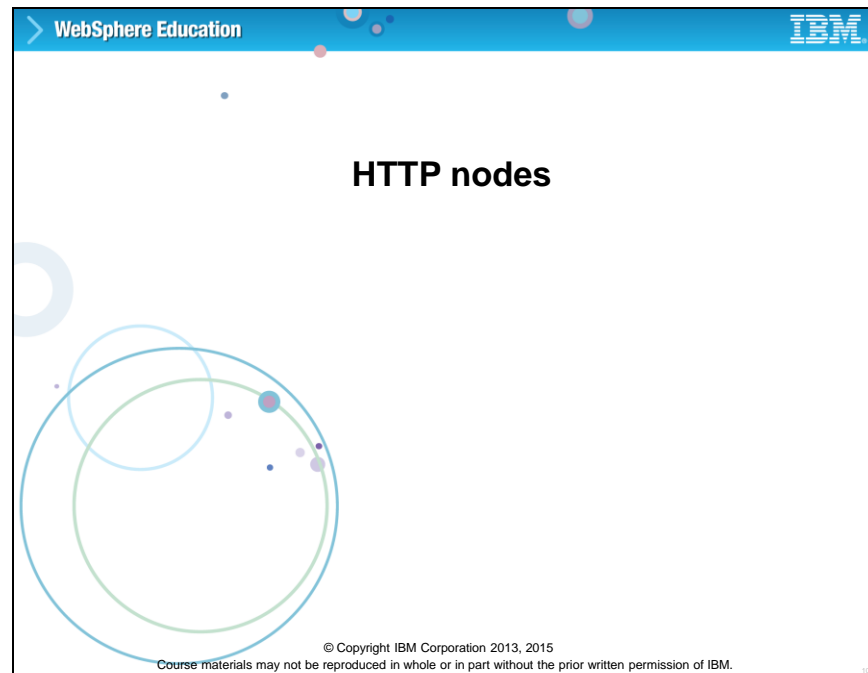
The integration node listener is the default listener when the message flow contains HTTP nodes. It requires access to SYSTEM.BROKER queues on an IBM MQ queue manager, which must be associated with the integration node.

The integration server embedded listener is the default listener when the message flow contains SOAP Input, SOAP Reply, and SOAP AsyncResponse nodes.

As an option, you can change the configuration such that some integration servers use the integration node listener for HTTP nodes, SOAP nodes, or both. You can change other integration servers to use the embedded listener for HTTP nodes, SOAP nodes, or both.

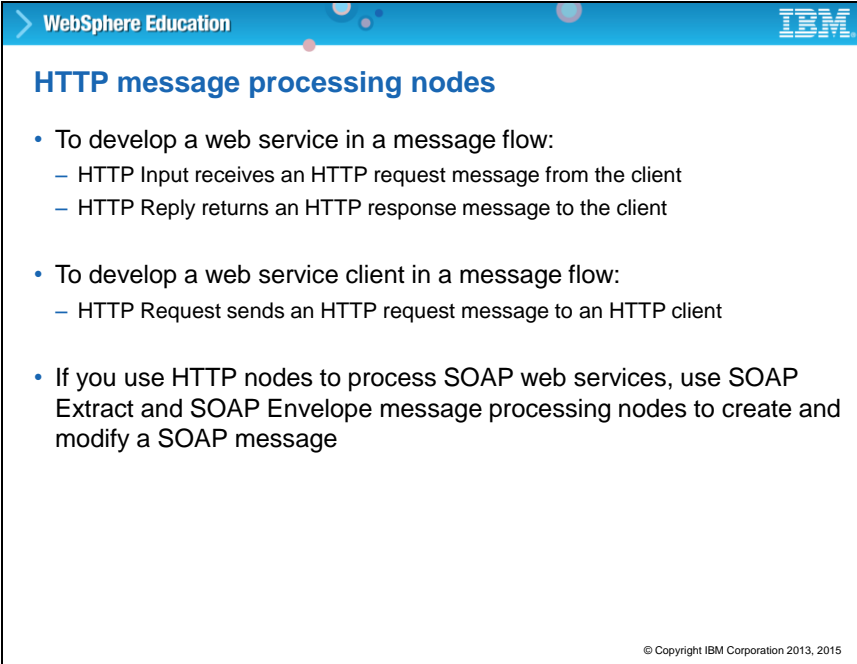
If you choose to disable the integration node listener, the integration server listeners are used for all HTTP and SOAP nodes, even if you did not explicitly enable support for them. So, for example if you set all relevant integration node and integration server properties to "false", the integration server listeners handle all HTTP messages.

Slide 9




Topic 2: HTTP nodes

This topic describes how to use the HTTP nodes in a message flow.



The slide is titled "HTTP message processing nodes" and is part of a "WebSphere Education" presentation. It contains three bullet points describing how to develop web services and clients using HTTP nodes. The IBM logo is in the top right corner, and a copyright notice is at the bottom right.

WebSphere Education 

HTTP message processing nodes

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

© Copyright IBM Corporation 2013, 2015

HTTP message processing nodes

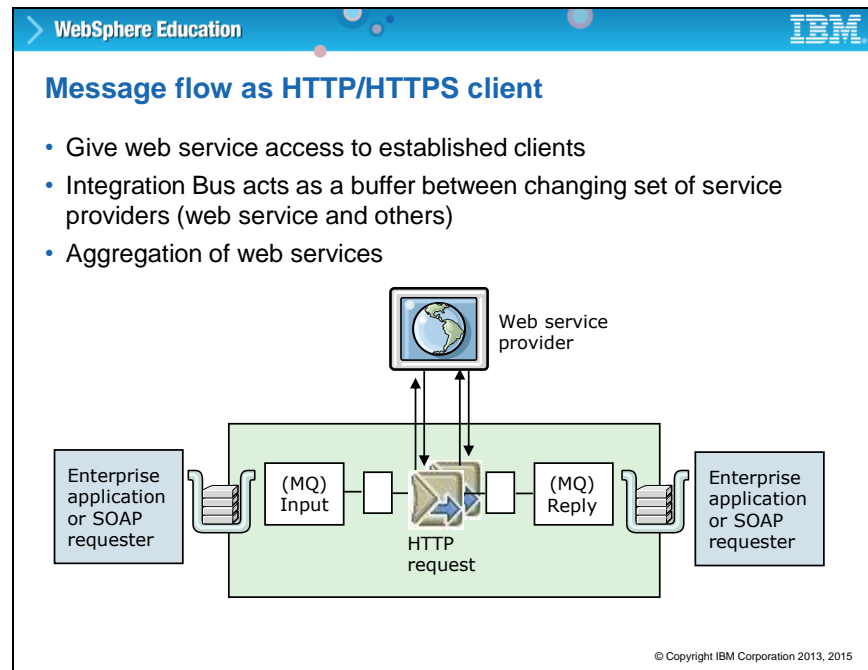
HTTP nodes support web service provider and consumer scenarios with request/response.

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

To develop a message flow to provide web services, you can use the HTTP Input node to receive an HTTP message from an HTTP client. You then use the HTTP Reply node to return an HTTP response message to the client.

If the message flow needs to access a web service, you can use the HTTP Request node to 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 before the message is propagated to subsequent nodes in the message flow.

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



Message flow as HTTP/HTTPS client

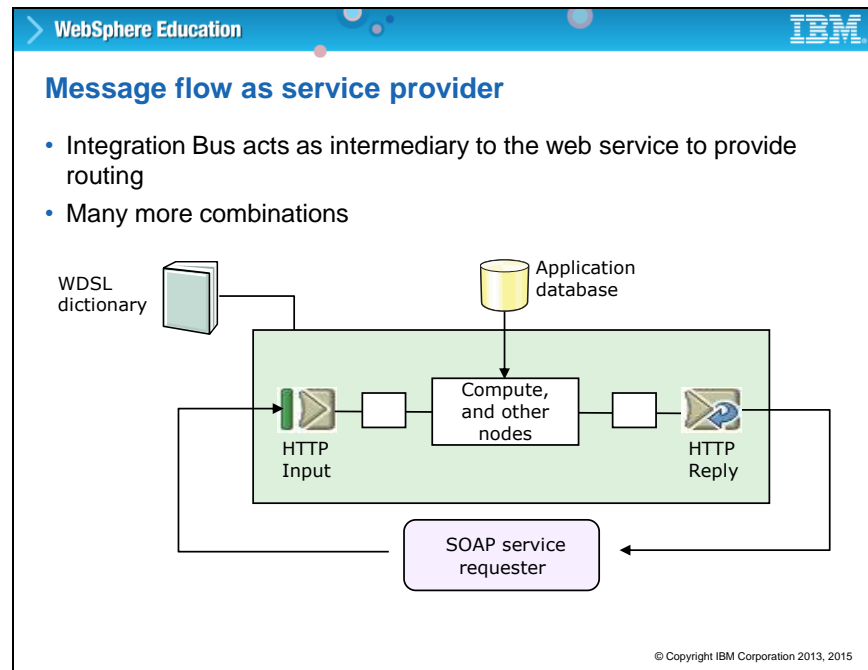
The message flow in the example uses the HTTP Request node to interact with a web service by using all or part of the input message as the request sent to that service. The node can also be configured to create an output message from the contents of the input message before the message is propagated to subsequent nodes in the message flow.

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

The **Default Web service URL** property on the HTTP Request node determines the destination URL for a web service request. You can configure a Compute node before the HTTPRequest node within the message flow to override the value that is set in the property.

The HTTPRequest node does full HTTP request/reply synchronously.

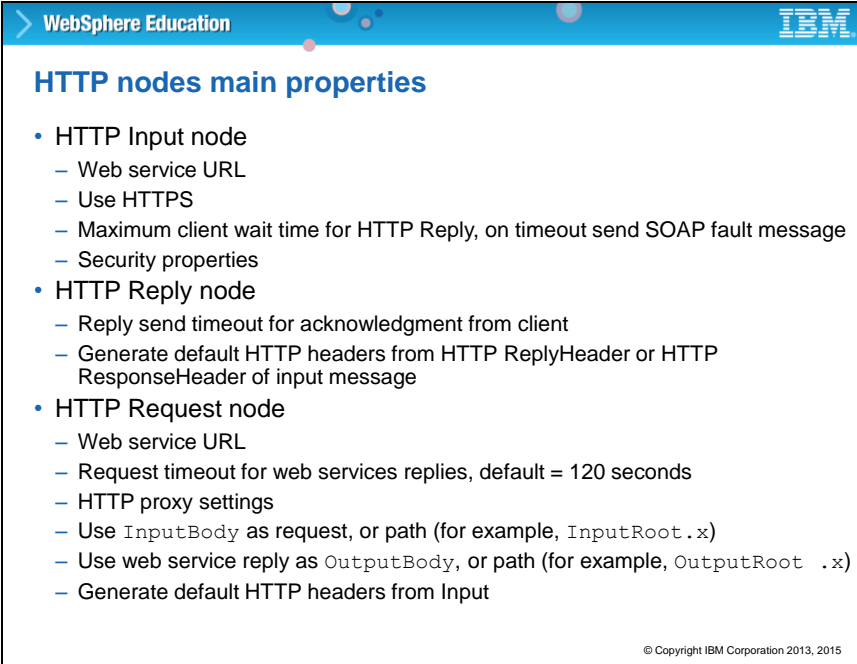
If it is specified on the URL, the HTTPRequest node uses HTTPS.



Message flow as service provider

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

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 is already looked up. An attribute on each HTTP Input node qualifies the requests for which the node is responsible.



The slide is titled "HTTP nodes main properties" and is part of a WebSphere Education presentation. It lists the main properties for three types of HTTP nodes: HTTP Input, HTTP Reply, and HTTP Request. The properties are organized into three bullet points, each corresponding to a node type. The HTTP Input node properties include Web service URL, Use HTTPS, Maximum client wait time for HTTP Reply, and Security properties. The HTTP Reply node properties include Reply send timeout for acknowledgment from client and Generate default HTTP headers from HTTP ReplyHeader or HTTP ResponseHeader of input message. The HTTP Request node properties include Web service URL, Request timeout for web services replies (default 120 seconds), HTTP proxy settings, Use InputBody as request or path (for example, InputRoot.x), Use web service reply as OutputBody or path (for example, OutputRoot.x), and Generate default HTTP headers from Input. The slide also includes a copyright notice for IBM Corporation 2013, 2015.

WebSphere Education

HTTP nodes main properties

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

© Copyright IBM Corporation 2013, 2015

HTTP nodes main properties

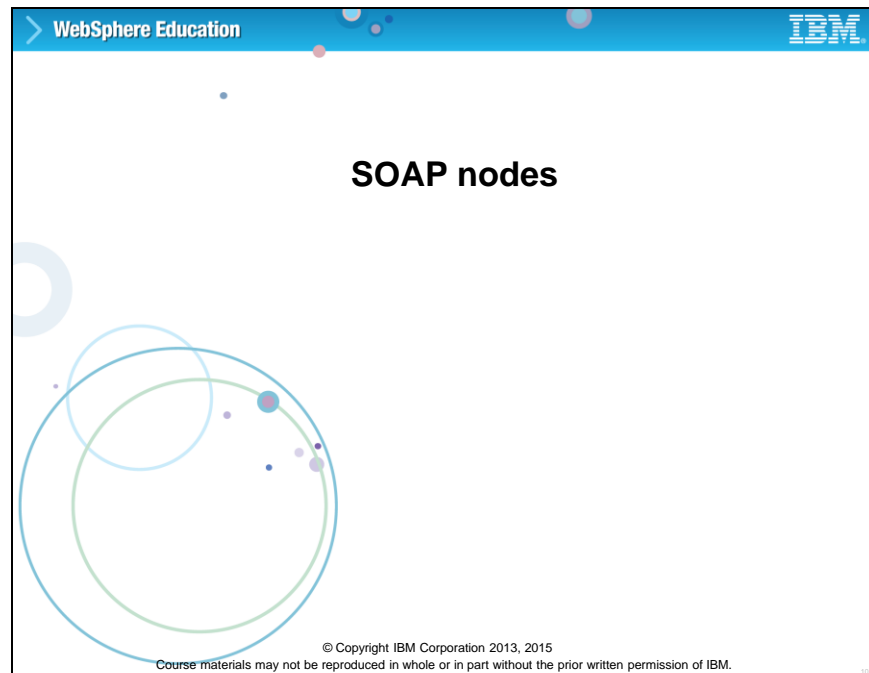
The HTTP nodes are the HTTP Input node, HTTP Reply node, and HTTP Request node. This slide highlights the main properties for the HTTP nodes.

The HTTP Input node receives an HTTP message from an HTTP client for processing by a message flow. **Path suffix for URL** identifies the source of web service requests. **Use HTTPS** identifies whether the node accepts secure HTTP. You can also specify the maximum wait time for an HTTP Reply and properties for security.

The HTTP Reply node returns a response from the message flow to an HTTP client. This node generates the response to the HTTP client and waits for confirmation that it was sent. With the HTTP Reply node, you can specify the time to wait for an acknowledgment from the client. You also have the option of generating the HTTP header from the HTTP header in the input message.

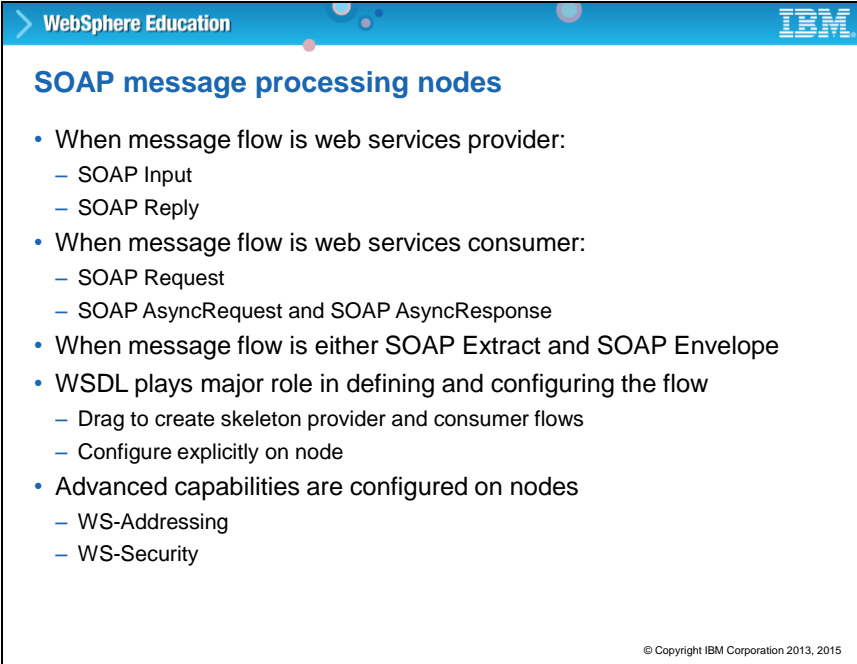
An HTTP Request node can call a web service in the middle of a flow. The **Web service URL** property identifies the URL for the web service. The **Request timeout** property is the time in seconds that the node waits for a response from the web service. The **HTTP(S) proxy location** property identifies the proxy server to which requests are sent.

The Integration Toolkit Tutorials Gallery includes examples of message flows that use the HTTP nodes.



Topic 3: SOAP nodes

SOAP nodes include HTTP function, but they offer much more. This topic describes how to use the SOAP nodes in a message flow.



The slide is titled "SOAP message processing nodes" and is part of a WebSphere Education presentation. It contains a bulleted list of information about SOAP nodes. The list includes: When message flow is web services provider (SOAP Input, SOAP Reply); When message flow is web services consumer (SOAP Request, SOAP AsyncRequest and SOAP AsyncResponse); When message flow is either SOAP Extract and SOAP Envelope; WSDL plays major role in defining and configuring the flow (Drag to create skeleton provider and consumer flows, Configure explicitly on node); and Advanced capabilities are configured on nodes (WS-Addressing, WS-Security). The IBM logo is in the top right corner, and a copyright notice is at the bottom right.

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

© Copyright IBM Corporation 2013, 2015



SOAP message processing nodes

Typically, SOAP nodes are used when working with SOAP-based web services. The slide describes the SOAP nodes available for sending web service requests and receiving web service responses.

A developer can choose to implement SOAP request/response flows synchronously or asynchronously. Calling a web service synchronously with a SOAP Request node means that the flow waits for a response from the web service before processing continues. Calling a web service asynchronously means that the SOAP Async Request node sends a web service request. The request does not block the message flow by waiting for the associated web service response because the response is received at the SOAP Async Response node, which is in a separate flow. This method handles multiple requests in parallel.

You can configure the SOAP nodes manually or by using the WSDL that describes the service.

Web Services Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies. In Integration Bus, policy sets and bindings define and configure your WS-Security and WS-RM requirements for the SOAP node.

 WebSphere Education 

SOAP Input and SOAP Reply nodes

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

© Copyright IBM Corporation 2013, 2015

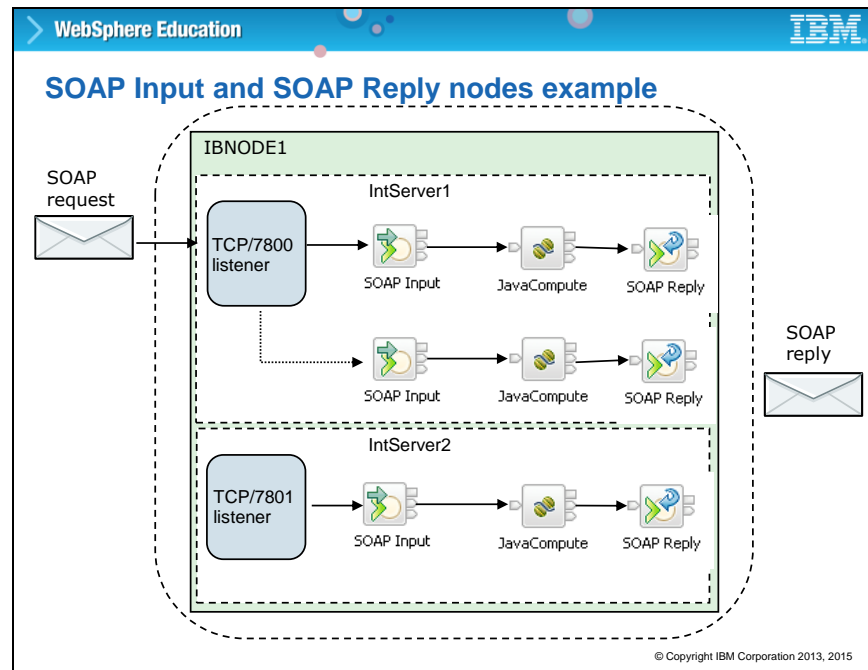
SOAP Input and SOAP Reply nodes

The SOAPInput and SOAPReply nodes are similar to the HTTPInput and HTTPReply nodes and are used in a message flow that implements a web service. The SOAPInput node listens for incoming web service requests, and the SOAPReply sends responses back to the client.

SOAP nodes automatically handle SOAP with Attachment (SwA) and Message Transmission Optimization Mechanism (MTOM) messages.

Integration Bus also supports a SOAP domain. You can use the SOAP parser with the SOAP nodes in your message flow to create a common WSDL-based logical tree format for working with web services, independent of the physical bitstream format.

Every integration server that contains a running message flow with a SOAP Input node is allocated a TCP/IP port to handle and accept incoming HTTP requests.



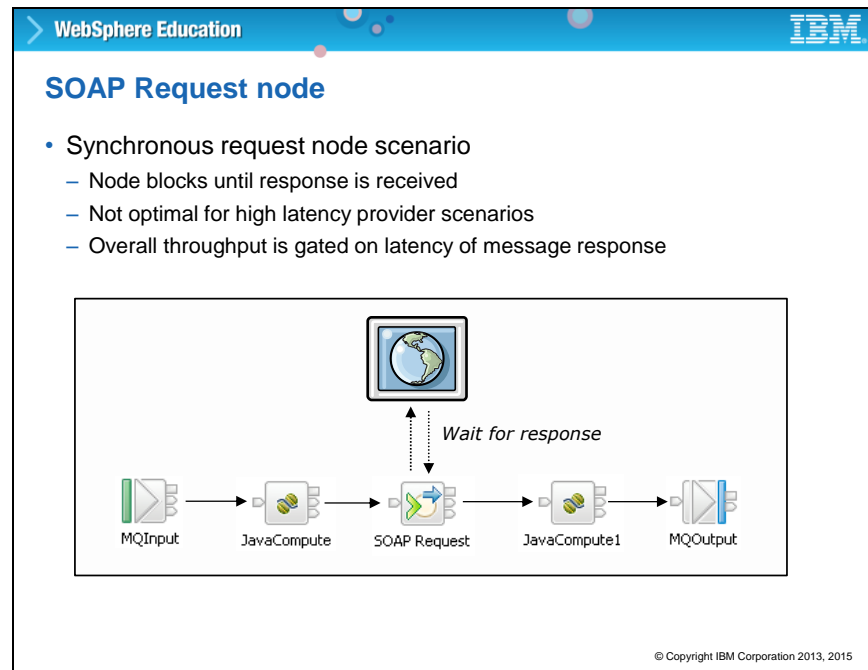
SOAP Input and SOAP Reply nodes example

This slide shows an example of message flows that contain SOAP Input nodes and SOAP Reply nodes.

By default, SOAP Input, SOAP Reply, and SOAP AsyncResponse nodes use an integration server embedded HTTP listener to manage HTTP requests, instead of the integration node HTTP listener. The listener is associated with an HTTP Connector object and an HTTPS Connector object. The HTTP Connector and HTTPS Connector objects control the runtime properties that affect the handling of HTTP messages.

In the example, the integration server listener handles the SOAP requests. Each integration server in the integration node uses a unique listener port. Each connector has its own assigned port, which is allocated from a range of numbers, as required. The default range for the HTTP Connector is 7800 - 7842; the default range for the HTTPS Connector is 7843 - 7884.

For example, the first integration server to start an embedded listener by using the HTTP Connector is allocated port 7800. The second integration server to start an embedded listener by using the HTTP Connector is allocated port 7801.



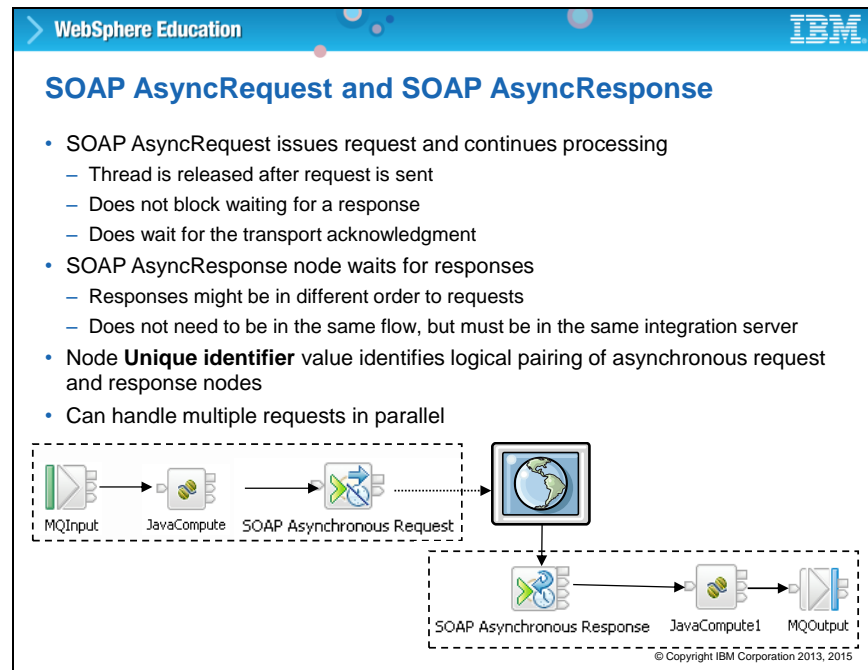
SOAP Request node

Similar to the HTTP Request node, the SOAP Request node is used to start a web service from within a message flow.

If the web service provider takes a long time to return the message, then the message flow is blocked until a response is received. This blocking can affect the overall throughput of messages through the flow.

Synchronous requests in a message flow (not only SOAP Request but also HTTP Request, MQ Get, Aggregation nodes) can always lead to latency problems.

Integration Bus provides an asynchronous alternative for SOAP requests.



SOAP AsyncRequest and SOAP AsyncResponse

You can use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a web service asynchronously. This asynchronous function enables multiple outbound requests to be made almost in parallel because the outbound request is not blocked while it waits for the response.

The SOAPAsyncRequest node is linked as a pair with a SOAPAsyncResponse node by using a unique identifier, and optionally WS-Addressing, to correlate response messages with the original request.

Asynchronous SOAP nodes need special configuration.

WebSphere Education

Configuring SOAP AsyncRequest nodes

SOAP Asynchronous Request Node Properties - SOAP Asynchronous

Description ✖ Unique identifier: A value must be set for this property.

Basic

Unique identifier* Unique identifier value is used to pair nodes
e.g. Asynchronous_NodePair_1

Operation mode

WSDL Properties

WSDL file name*

Port type*

Binding*

Binding operation*

Service port*

Target namespace:

Transport

© Copyright IBM Corporation 2013, 2015

Configuring SOAP AsyncRequest nodes

This slide shows the SOAP AsyncRequest node **Basic** properties.

The **Unique identifier** property on SOAPAsyncRequest node links to a SOAPAsyncResponse node. When using HTTP transport, this identifier is used as a unique URL fragment to identify incoming response messages for the SOAPAsyncResponse node.

The **WSDL file name** property is the location of the WSDL file that you want to use to configure the node. Enter the full path to the WSDL file, or click **Browse** to select a WSDL file from the workspace.

The **Port type** property lists all the port types that the WSDL file defines. By default, the first port type that is found in the WSDL file that has an associated HTTP or JMS binding is selected.



The **Binding property** lists all the SOAP bindings associated with the selected port type. Only HTTP or JMS transport is supported. Bindings are listed in the order that they are displayed in the WSDL file. By default, the first binding that implements the operation and has an associated service port is selected.

The **Binding operation** property contains all the operations defined by the selected binding. The first operation in the list is selected by default.

The **Service port** property lists all the service ports that point to the selected binding. The first service port for the binding is selected by default.

The **Target namespace** property displays the namespace of the selected WSDL file.

The **Transport** property is set automatically when the **Binding** property is selected. The value of this property shows the transport used by the selected WSDL binding. Use the properties on the **HTTP Transport** and **JMS Transport** properties tabs to configure the transport properties.



WSDL importer

- Creates a message model from WSDL
 - Models all messages that can exist inside SOAP <Envelope>
 - Imports the predefined message definitions for the <Envelope> itself
- Accepts a range of WSDL formats
 - Single and multifile formats
 - RPC-encoded, RPC-literal, and document-literal WSDL styles
 - WSDL V1.1, SOAP V1.1, and SOAP V1.2
- Validates WSDL against WS-I Basic Profile
- Line command: `mqsicreatemsgdefsfromwsdl` or `mqsicreatemsgdefs`
- To implement the WSDL binding and endpoint details in a message flow, drag WSDL into the Message Flow editor
- Creates and configures subflows
 - To call a web service or to expose the flow as a web service
 - To wrap or unwrap the SOAP envelope

© Copyright IBM Corporation 2013, 2015

WSDL importer

When you import a WSDL document into the Integration Toolkit, you can build a web service or web service client that is based on the operations and message model that is defined in the WSDL document.

The WSDL importer creates a message model from WSDL. It models all messages that can be used inside SOAP <Envelope> and imports the IBM supplied message definitions for the <Envelope> itself.

The WSDL file is validated to ensure that it is WS-I compliant. If the WSDL file uses a SOAP/JMS transport URI, it is not WS-I compliant, but by default no error is shown. You can enable strict WS-I validation and display a warning when a SOAP/JMS transport is used by clearing the **BP 1.1: Allow SOAP/JMS as transport URI** check box in the **Integration Development > WSDL > Validation** preferences in the Integration Toolkit.

You can import WSDL in the Integration Toolkit or by using the `mqsicreatemsgdefsfromwsdl` or `mqsicreatemsgdefs` line commands.

You can implement the WSDL binding and endpoint details in a message flow by dragging the WSDL onto a blank message flow canvas.

WebSphere Education
IBM

Import WSDL into workspace

- Import a WSDL document into an Integration Bus application, library, or integration service:
 - From the Integration Toolkit menu, click **New > Message Model**.
 - Click **SOAP XML** as the message model type.
 - In the wizard, click **I already have WSDL for my data**.

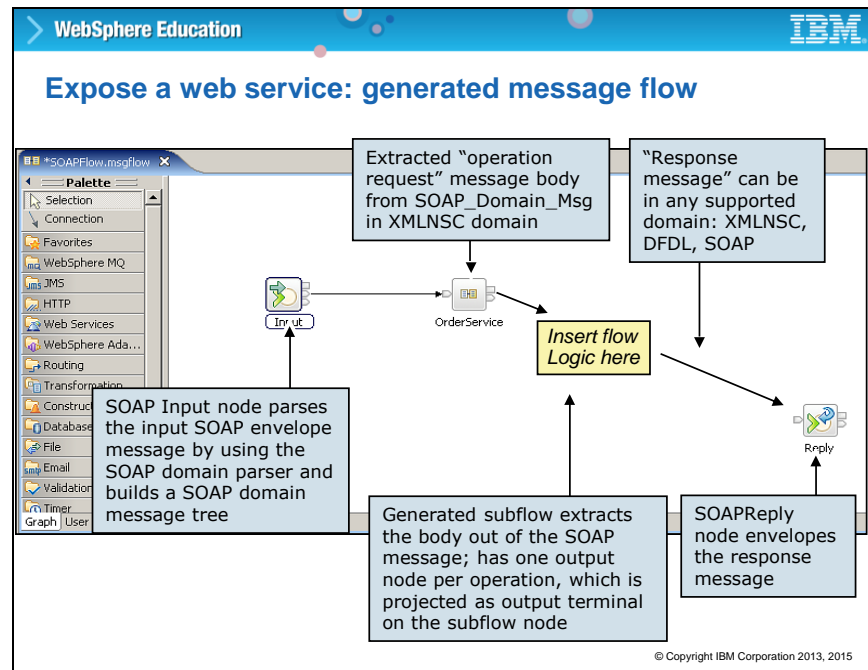
As an option, you can specify an XML schema file for the WSDL document XML schema types.

The image contains two screenshots of the 'New Message Model' wizard in IBM WebSphere. The left screenshot shows the 'Create a new message model file' step where 'SOAP XML' is selected under the 'XML' category. The right screenshot shows the 'SOAP XML' step where 'I already have WSDL for my data' is selected among several options.

Import WSDL into workspace

As an option, you can import a WSDL document into an Integration Toolkit application, library, or integration service with the **Message Model Import wizard**. Use the **SOAP XML** option as a starting point for a new message model. In the SOAP XML message model options, click **I already have a WSDL for my data** to import the WSDL document from your computer.

The WSDL document has two uses in a library or application: configuring web service operations, and working with SOAP web service message models.



Expose a web service generated message flow

After you import the WSDL into the Toolkit, you can drag the WSDL onto the message flow canvas to create a web service message flow skeleton. In the wizard you specify whether the flow should be exposed as a web service, or whether the WSDL should be used to start a remote web service.

Depending on which option you select, the wizard automatically selects the binding operations and the service port that should be used. If multiple bindings and ports are available, these binding types are all listed; you select the appropriate one.

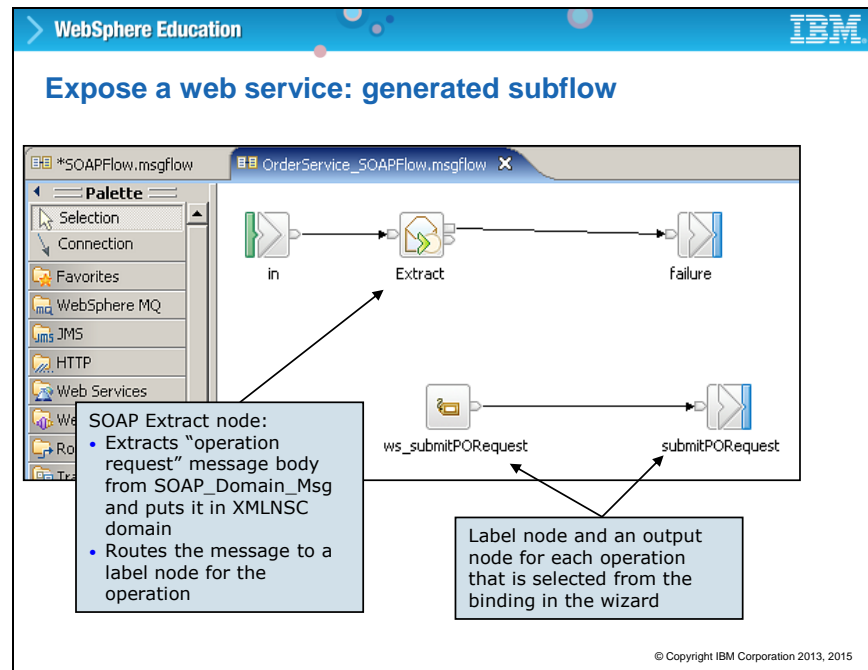
The example that is shown on this slide shows a message flow that is exposed as a web service.

The first node is a SOAP input node that is named Input by default. The properties of the Input node are populated automatically by using the values that are derived from the imported WSDL, and the selections that were made in earlier stages of the wizard. The SOAP input node creates a SOAP domain message that is based on the SOAP messages that arrive at this input node.

The second node, called OrderService, calls a subflow. This subflow contains a terminal for each of the operations that you selected in the wizard. This subflow extracts the message body and removes the SOAP envelope. The output from this subflow is the payload of the message in the XMLNSC domain.

The final node is a SOAP Reply node, which sends a response message that corresponds to the SOAP Input node.

User flow logic is placed between the generated subflow and the final Reply node.



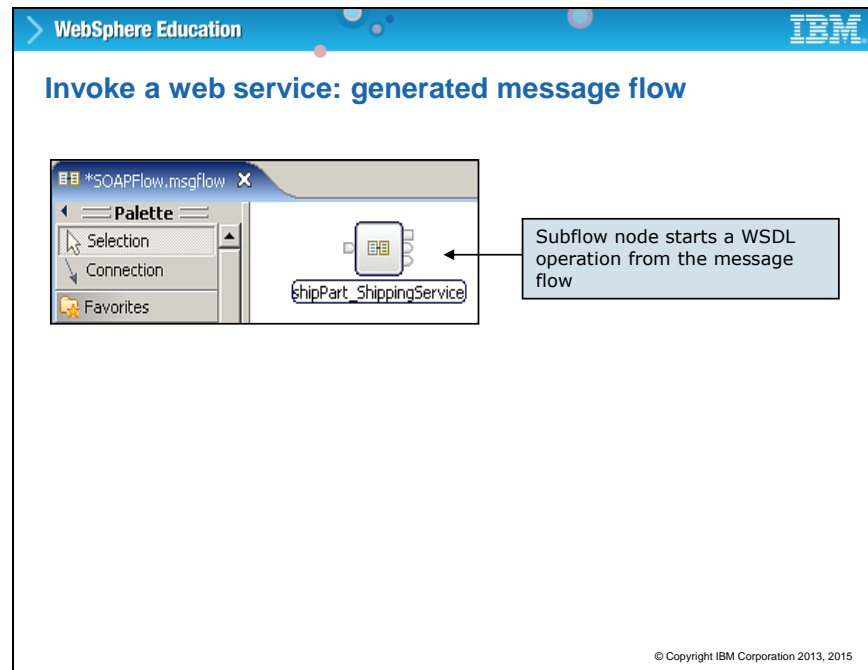
Expose a web service generated subflow

This slide shows the generated subflow that is used to handle the incoming SOAP message. The subflow uses a RouteToLabel node to handle each operation. The example on this figure specifies only one operation so it has one Label node.

The Label node then passes the message to a corresponding Output node, which in turn corresponds to the appropriate output terminal on the subflow node.

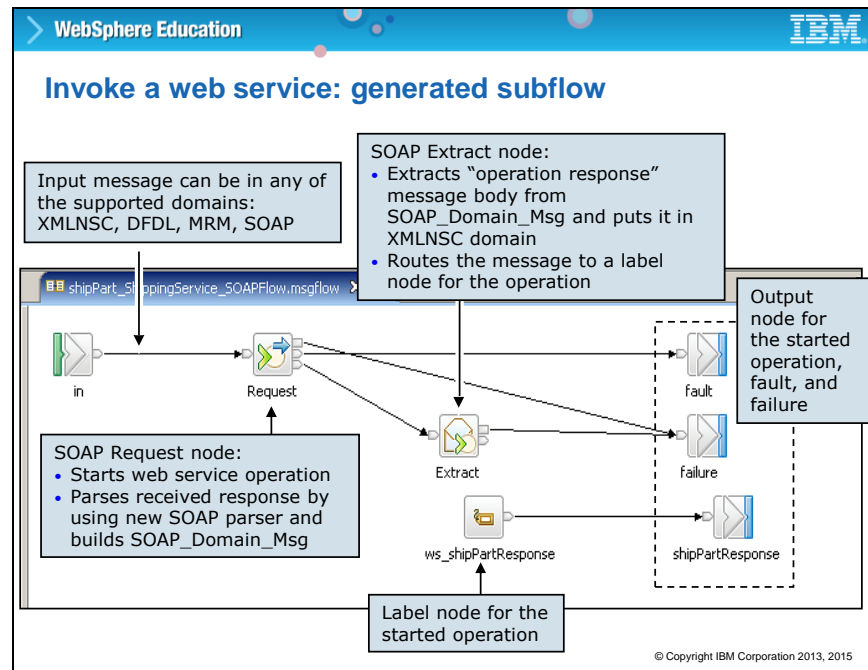
The node that is named Extract is a SOAPExtract node that removes the SOAP envelope and places the resulting payload of the incoming message into the XMLNSC domain.

For the client scenario, another flow skeleton is generated.



Invoke a web service generated message flow

This slide shows the client scenario, where the message flow calls a web service. The subflow and details are provided on the next slide.

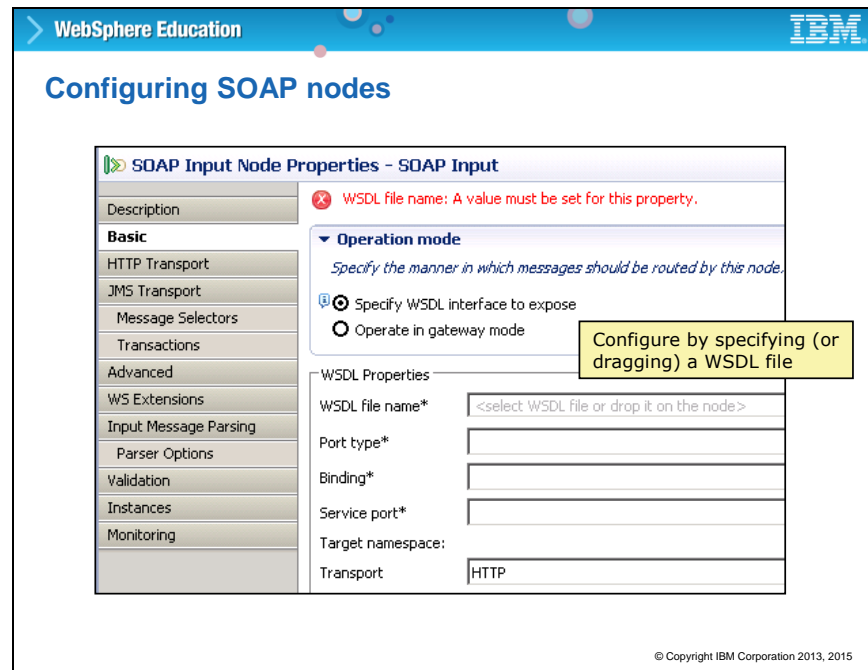


Invoke a web service generated subflow

Expanding the subflow shows a SOAP Request node. The default name of this node is Request.

On completion, this node passes a message in the SOAP domain to a SOAP Extract node, which removes the SOAP envelope and passes the payload of the SOAP message to the rest of the message flow. The output from the Extract node is in the XMLNSC domain.

In this scenario, one label node and multiple output nodes handle SOAP faults and failures.





Configuring SOAP nodes

You can configure SOAP nodes by using a WSDL file. This approach is suggested because it reduces the possibility of configuration errors.

If you use a corresponding SOAP Reply node in the message flow, its configuration is taken from the associated SOAP Input node. A reference to the originating SOAP node is passed in the LocalEnvironment folder of the message assembly.

The properties of the SOAP Input are organized under the following tabs.

- The **Basic** properties configure the WSDL properties, in particular, the WSDL file name, port type, binding, and service port. If the WSDL contains more than one binding, port Type, or other properties, then the correct one can be defined from a menu. At run time, if the SOAP message contains an operation that is not defined within the WSDL, a SOAP fault is returned.
- The **HTTP Transport** properties specify the URL Selector, whether to use HTTPS, and a Maximum Client wait time.
- You configure the **JMS Transport** properties when the messages are sent by using SOAP over JMS as the transport protocol.
- **WS-Extensions** properties configure WS-Addressing and WS-Security. The WS-Extensions properties are covered later in this unit.

 WebSphere Education 

Generating WSDL from message set

- In the Integration Toolkit, right-click the folder that contains the message set file and then select **Generate > WSDL Definition**
 - Generate a new WSDL definition from existing message definitions
 - Export an existing WSDL definition to another directory in the workspace or file system
- One or more bindings can be requested
 - SOAP/JMS, SOAP/HTTP, JMS TextMessage
- WSDL message types match MRM logical messages
 - Physical representations can require mapping or transformation for bindings
- Supports both the recognized WSDL styles: “rpc” and “document”
- Only “literal” encoding; SOAP encoding not supported
- Validates generated WSDL against WS-I Basic Profile


© Copyright IBM Corporation 2013, 2015

Generating WSDL from message set

If an integration node is to communicate with a web service client, it typically needs to accept SOAP messages. One approach is to use the MRM domain, in which case the message model and the WSDL definition that the web service client uses must describe the same messages.

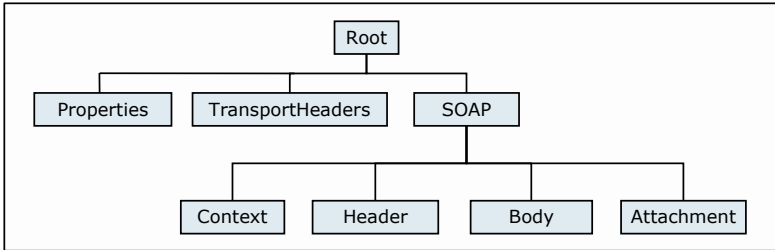
If the integration node has an existing message model, it can be exported to create a corresponding WSDL definition for use by the client. At the same time, your message model needs to be enhanced with appropriate definitions for the SOAP envelope and the WSDL operations (for RPC-style).

If the integration node must interact with an existing web service, you can import a WSDL definition into a message set. The resulting message set contains message definitions that model the SOAP envelope and the content of the corresponding SOAP messages. A flow developer can use these definitions to validate and work with an incoming message; for example, defining a mapping to transform a SOAP request message into a SOAP response message. Various WSDL styles are accepted.

WebSphere Education


Common SOAP message tree

- SOAP domain and parser offer a consistent approach for constructing a flow to handle web services, regardless of the specific bitstream format
- WSDL document triggers SOAP parser events and is used to validate the SOAP Envelope
- Web service message can be SOAP, SwA, or MTOM
- Context contains WSDL-related information such as operation name
- Generated from the WSDL



```

graph TD
    Root[Root] --> Properties[Properties]
    Root --> TransportHeaders[TransportHeaders]
    Root --> SOAP[SOAP]
    SOAP --> Context[Context]
    SOAP --> Header[Header]
    SOAP --> Body[Body]
    SOAP --> Attachment[Attachment]
    
```

© Copyright IBM Corporation 2013, 2015

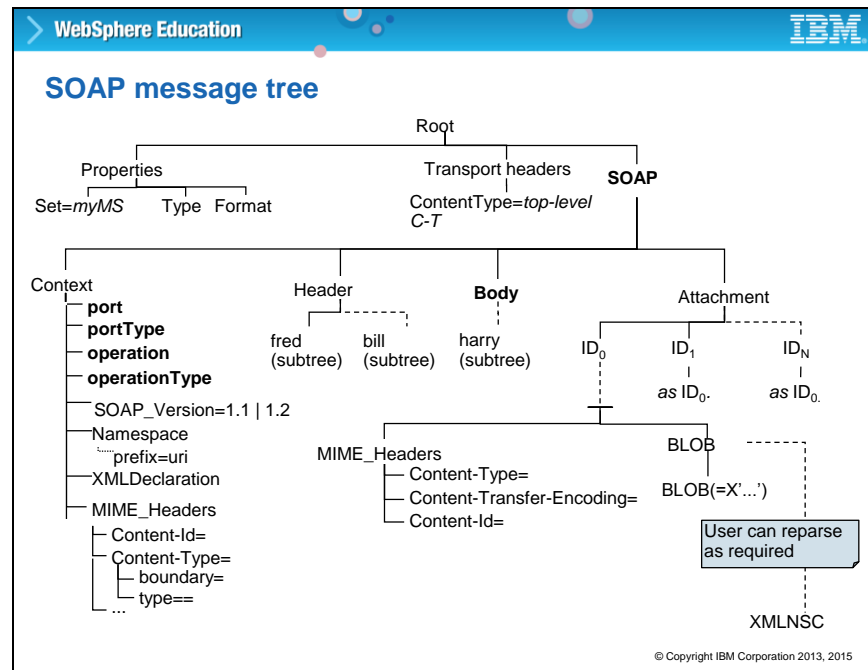
Common SOAP message tree

The SOAP parser represents web service messages with the same logical tree shape irrespective of the specific bitstream format. The SOAPInput node generates a tree whose shape is unaffected by the input message. By contrast, the message assembly that an HTTPInput node produces differs depending on whether the message was MIME format or SOAP format.

That WSDL that is deployed to the integration node is used to validate the SOAP messages that are received for the SOAPInput node. For example, you can ensure that the operation that is received in the incoming SOAP message is defined within the WSDL.

The bitstream format for these runtime messages can be SOAP V1.1 or SOAP V1.2, and optionally wrapped by MIME as an SOAP with Attachments (SwA) or MTOM message.

The next slide shows the tree in more detail.



SOAP message tree

This slide shows a more detailed view of the SOAP message tree.

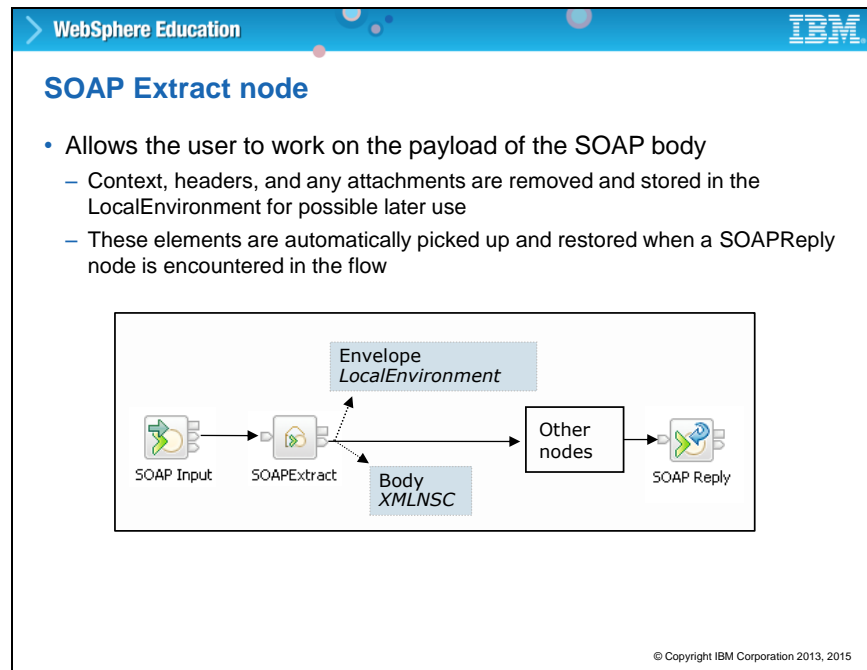
The **SOAP.Context** branch contains the SOAP envelope information. It also shows the SOAP version that is being used. The SOAP parser sets the following information (derived from the WSDL) under **SOAP.Context** on input: **port**, **portType**, **operation**, **operationType**.

The **Content-Id** within the **Attachment** section is copied directly under the **Attachment** part, allowing the specific message part to be easily identified and parsed.

The SOAP payload (under **Envelope.Body**) is saved under **SOAP.Body**.

The SOAP header blocks (under **Envelope.Header**) are saved under **SOAP.Header**.

With the **SOAPExtract** and **SOAPEnvelope** nodes you can work on the payload of the SOAP body.

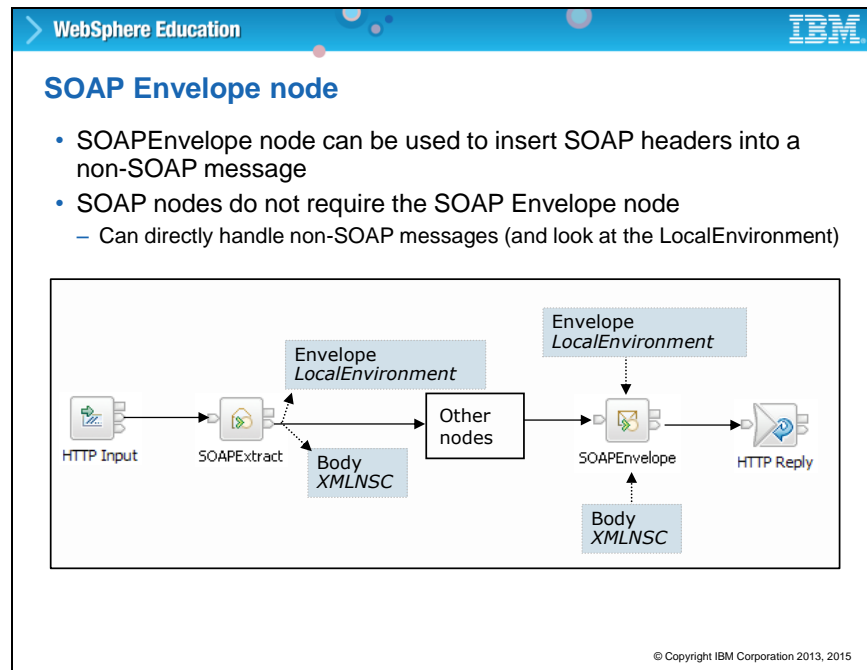


SOAP Extract node

The SOAP Extract node can detach the SOAP envelope to a standard location in the LocalEnvironment tree. Alternatively, you can specify an explicit location by using an XPath expression. Any existing SOAP envelope at the chosen location is replaced.

The SOAP Extract node can also route the SOAP message to a Label node within the message flow that the SOAP operation identifies within the message. The SOAP Operation is identified within the SOAP body tag.

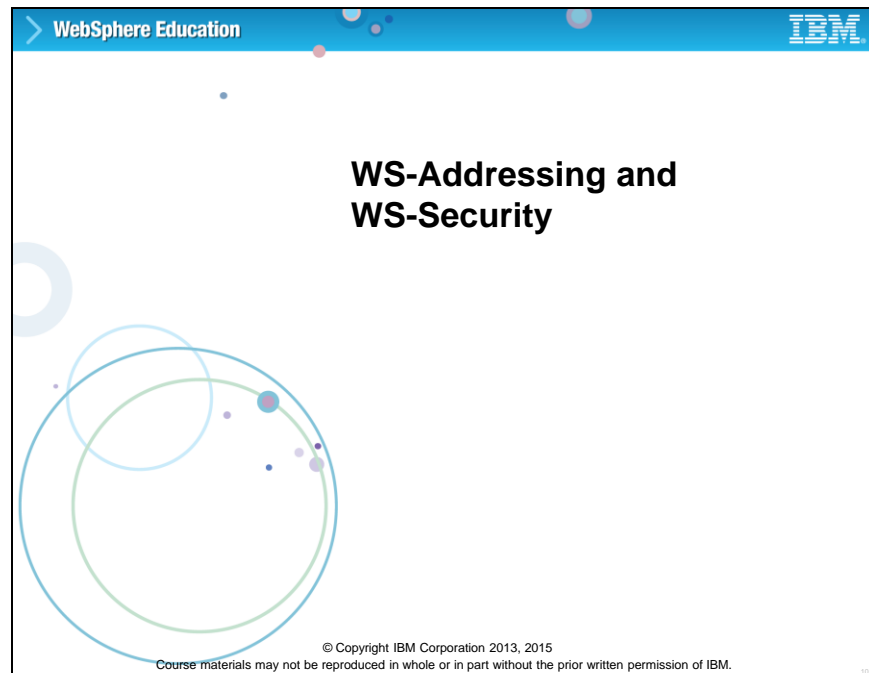
Both functions are optional; they are contained within one node because they are often used together.



SOAP Envelope node



You use the SOAP Envelope node to add a SOAP envelope back onto an existing message that was extracted by using the SOAP Extract node.

The default behavior of the SOAP Envelope node is to attach the SOAP envelope from a standard location in the LocalEnvironment tree. You can specify an explicit location by using an XPath expression. You can also use the node in a flow without a corresponding SOAP Extract node. The node can optionally create a default SOAP envelope.



Topic 4: WS-Addressing and WS-Security

In this topic, you learn how to implement WS-Addressing and WS-Security in Integration Bus.

 WebSphere Education 

WS-Addressing

- WS-Addressing is a standardized way of including the addressing data in the SOAP message itself
 - Endpoint Reference (EPR) is the information that is needed to address a web service endpoint.
 - Message Addressing Properties (MAPs) contains a list of addressing properties such as ReplyTo, FaultTo, MessageID, and Action
- Integration Bus automatically supports WS-Addressing
 - SOAP nodes: Use **WS-Addressing** check box
 - WSA headers flow in LocalEnvironment
 - Large number of overrides and status information is contained within the LocalEnvironment

© Copyright IBM Corporation 2013, 2015

WS-Addressing

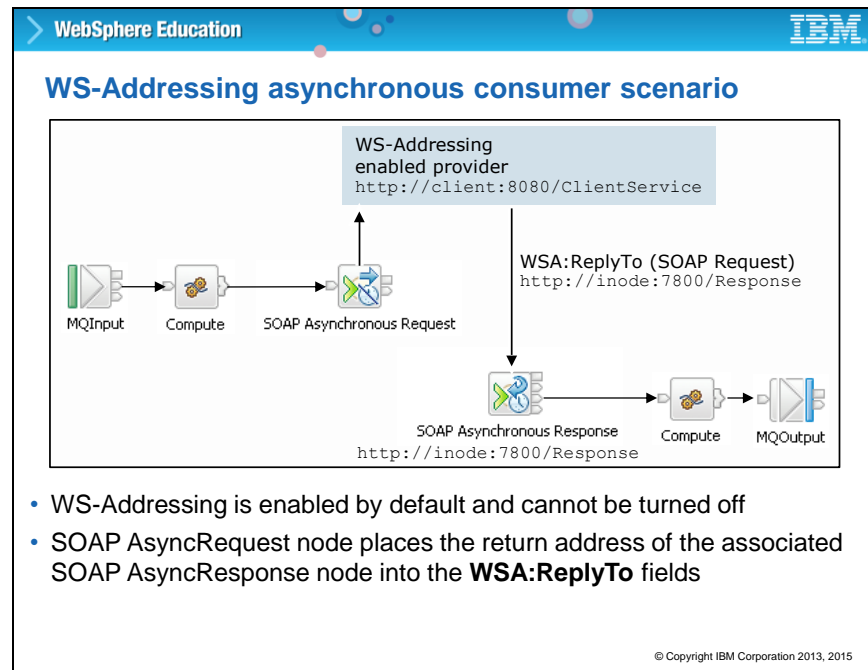
Web Services Addressing (WS-Addressing) is a W3C specification that aids interoperability between web services by defining a standard way to address web services and provide addressing information in messages.

The WS-Addressing specification introduces two primary concepts: endpoint references, and message addressing properties.

EPRs provide a standard mechanism to encapsulate information about specific endpoints. EPRs can be propagated to other parties and then used to target the web service endpoint that they represent.

Message addressing properties are a set of defined WS-Addressing properties that can be represented as elements in SOAP headers. Message addressing properties can provide either a standard way of conveying information, such as the endpoint to which message replies should be directed, or information about the relationship that the message has with other messages.

WS-Addressing is supported for SOAP Input, SOAP Reply, SOAP Request, and SOAP AsyncRequest nodes.



WS-Addressing asynchronous consumer scenario

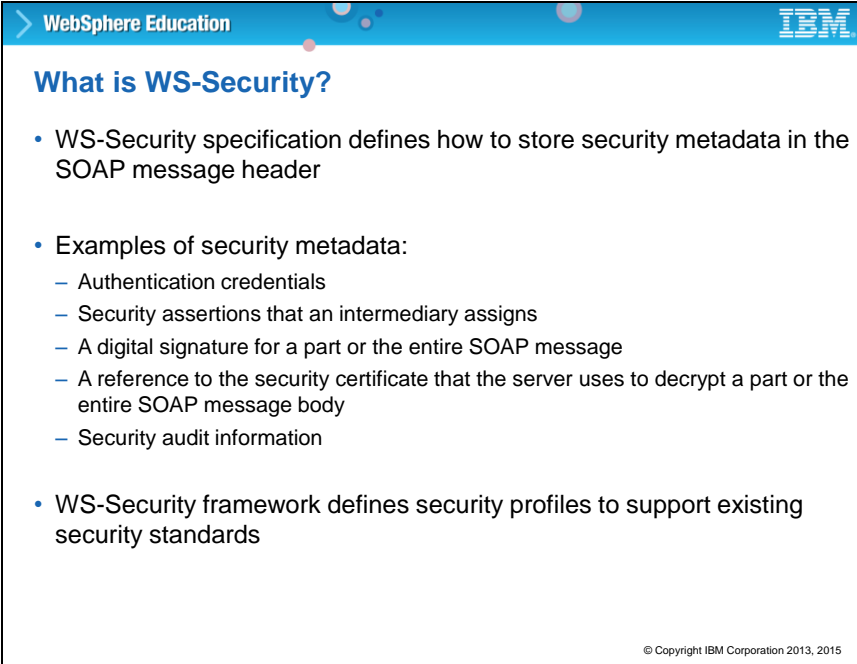
In this scenario, the message flow uses the SOAP Asynchronous Request node to start a web service. The response to the original web service request is handled separately from the request. In this case, the use of WS-Addressing is mandatory, and the property is automatically set on the SOAP AsyncRequest node.

The SOAP AsyncRequest node generates the required WS-Addressing headers. However, the "Reply To" header is the reply address of the associated SOAP Asynchronous response node.

Because the web service provider is enabled for WS-Addressing, the reply recognizes the "Reply To" header, and is sent back to the SOAP Async Response node. The second part of the message flow then completes as normal.

The SOAP Fault conditions are understood, and if it is specified, faults are sent to the address in the "Fault To" header.

For the SOAP Request, SOAP Input and SOAP Reply nodes, WS-Addressing is optional. However, it is mandatory for the SOAP Async nodes.



The slide is titled "What is WS-Security?" and is part of a "WebSphere Education" presentation, as indicated by the header. It contains a bulleted list of information about WS-Security. The first bullet point states that the WS-Security specification defines how to store security metadata in the SOAP message header. The second bullet point, "Examples of security metadata:", is followed by a list of five items: authentication credentials, security assertions assigned by an intermediary, a digital signature for a part or the entire SOAP message, a reference to the security certificate used by the server for decryption, and security audit information. The third bullet point states that the WS-Security framework defines security profiles to support existing security standards. A small copyright notice for IBM Corporation is located at the bottom right of the slide.

WebSphere Education

What is WS-Security?

- WS-Security specification defines how to store security metadata in the SOAP message header
- Examples of security metadata:
 - Authentication credentials
 - Security assertions that an intermediary assigns
 - A digital signature for a part or the entire SOAP message
 - A reference to the security certificate that the server uses to decrypt a part or the entire SOAP message body
 - Security audit information
- WS-Security framework defines security profiles to support existing security standards

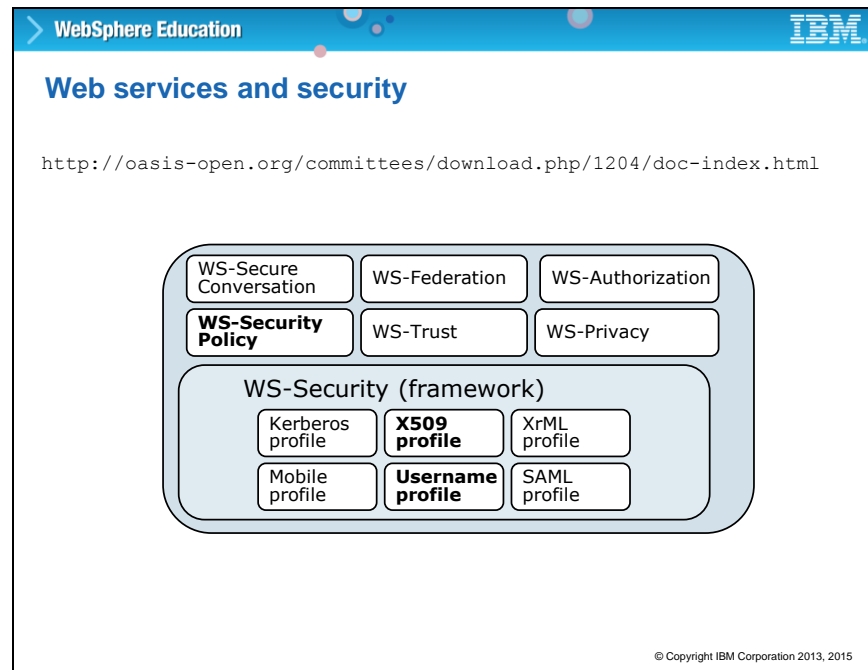
© Copyright IBM Corporation 2013, 2015

What is WS-Security?

Web Services Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

Integration Bus focuses on the part of the WS-Security specification that is known as WS-SecurityPolicy. This policy is implemented by using the properties of the SOAP Input or SOAP Request nodes.


Key areas that the Integration Bus encompasses include authentication, message-level protection, and message part protection.




Web services and security

What is often thought of as a single specification, WS-Security, is a large set of interrelated specifications. The figure shows the components that constitute WS-Security. It is important to note that the WS-Security specification is large, and many software vendors do not implement the entire specification. A software component normally implements the security features that are appropriate for the type of security usage that is required.

WS-Security defines a <Security> element that can be placed in the header section of a SOAP message.



WebSphere Education 

Transport level security and message security

- Transport level security (SSL and HTTPS)
 - Protects the stream of data that is being passed from one endpoint to another
 - Typically all of the data is encrypted; does not discriminate on a per message basis (everything is encrypted) with the same key
 - When passing messages by using another intermediary, if the intermediary must “see” any part of the message, it can access all of it
- Message level security (WS-Security)
 - Message-based security provides finer granularity
 - Security is applied on a per message basis
 - Parts of a message can be (multiply) encrypted and signed on a “need-to-know” basis
 - WS-Security can be used with insecure transports

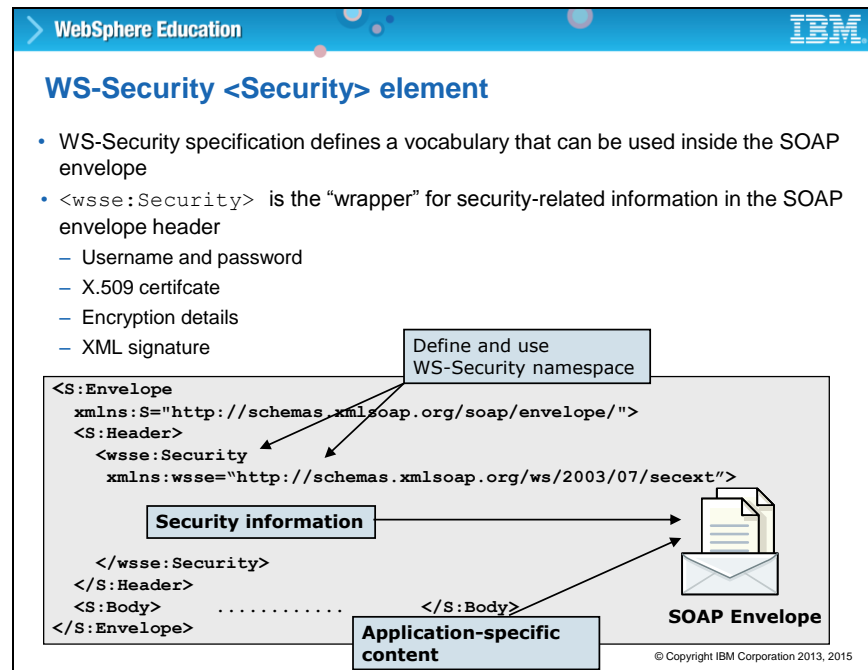
© Copyright IBM Corporation 2013, 2015

Transport level security and message security

A primary security task is to keep the SOAP message secure through intermediate systems until it reaches the ultimate recipient.

At the transport level, you can use SSL and HTTPS.

At the message level, you can use WS-Security. WS-Security is based on securing SOAP messages through an XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens. The WS-Security specification defines the core facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message.



WS-Security <Security> element

A SOAP message consists of an envelope that contains the whole message. Within the envelope is the actual message, which consists of header information and a SOAP body. The body is the real payload, that is, the message that an application program typically needs to access.

WS-Security defines a new <Security> element that can be placed in the header section of a SOAP message. It can contain a user name/password, X.509 certificate, encryption details, or XML signature.

The example shows a SOAP message with a skeleton showing where WS-Security should be placed. The element `wsse:Security` is used to contain the security information.

WebSphere Education

WS-Security authentication in IBM Integration Bus

- User name/password (Username Token) is fully supported when using either LDAP or a WS-Trust v1.3 STS to authenticate the identity
- X.509 Certificate (Binary Token) by using Java key and trust store configured on the integration node or integration server
- Kerberos Token Profile by using the JVM/Host Kerberos infrastructure, krb5.conf, and keytab files

```

<S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  <S:Header>
    <wsse:Security
      xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/07/secext">
      <wsse:UsernameToken wsu:ID="myToken">
        <wsse:Username>jonathan</wsse:Username>
        <wsse:Password>passw0rd</wsse:Password>
      </wsse:UsernameToken>
      </wsse:Security>
    </S:Header>
    <S:Body>
      Application-specific content
    </S:Body>
  </S:Envelope>

```

Security information

Application-specific content

© Copyright IBM Corporation 2013, 2015


WS-Security authentication in IBM Integration Bus

WS-Security defines two types of security tokens: a user name token and binary security token.

A user name token consists of a user name and optionally, password information. You can include a user name token directly as an element in the **Security header** within the SOAP message, rather than the **HTTP header**. Binary tokens, such as X.509 certificates, Kerberos tickets, Lightweight Third-Party Authentication (LTPA) tokens, or other non-XML formats, require a special encoding for inclusion.

In this example, a user name token is specified. The user name and password are specified in clear text. This text can be encrypted by using message part encryption, which is described later.

The Integration Bus administrator must enable security and specify a security profile. If these steps are not done, all user names and passwords are accepted.

WebSphere Education 

WS-Security message level protection

- Entire body within the SOAP message can be encrypted and signed
 - Using different certificates, if required
- Signed (XML signature): Message can be read but not changed
- Encrypted (XML encryption): Message cannot be read or changed

```

<S:Envelope>
  <S:Header>
    <wsse:Security S:mustUnderstand="1"
      xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext">
      <wsse:BinarySecurityToken EncodingType="wsse:Base64Binary">
        MIIDQTCC4ZzO7tIgerPlaid1q ... [truncated]
      </wsse:BinarySecurityToken>
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        ...signature data...
      </ds:Signature>
    </wsse:Security>
  </S:Header>
  <S:Body>
    <m:OrderAircraft quantity="1" type="777" config="Atlantic"
      xmlns:m="http://www.boeing.com/AircraftOrderSubmission"/>
  </S:Body>
</S:Envelope>

```

Security information


Application-specific content

© Copyright IBM Corporation 2013, 2015

WS-Security message level protection

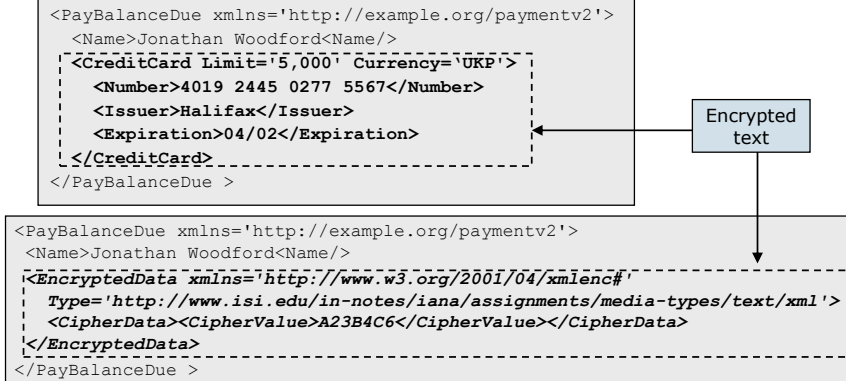
Integration Bus supports WS-Security message protection for the entire message or part of the message. The message can be encrypted, digitally signed, or both.

The example shows a payload that is digitally signed. However, it is not encrypted, and is still readable by intermediaries. By using this technique, you can allow intermediaries to access parts of the message, but inhibit access to other parts of the message.

WebSphere Education 

WS-Security message part protection

- Allows header and body to be encrypted and signed
- Different certificates can be used for different parts of the message
- Element and namespace (QNAME) support



The diagram illustrates the process of WS-Security message part protection. It shows two versions of a SOAP message body. The top version is the original plaintext, and the bottom version is the encrypted version. A dashed box in the top version highlights the `<CreditCard>` element, which is then mapped to an `Encrypted text` box. This box points to the `<EncryptedData>` element in the bottom version of the message body.

```
<PayBalanceDue xmlns='http://example.org/paymentv2'>
  <Name>Jonathan Woodford</Name>
  <CreditCard Limit='5,000' Currency='UKP'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Halifax</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PayBalanceDue >
```

```
<PayBalanceDue xmlns='http://example.org/paymentv2'>
  <Name>Jonathan Woodford</Name>
  <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
    Type='http://www.isi.edu/in-notes/iana/assignments/media-types/text/xml'>
    <CipherData><CipherValue>A23B4C6</CipherValue></CipherData>
  </EncryptedData>
</PayBalanceDue >
```

© Copyright IBM Corporation 2013, 2015

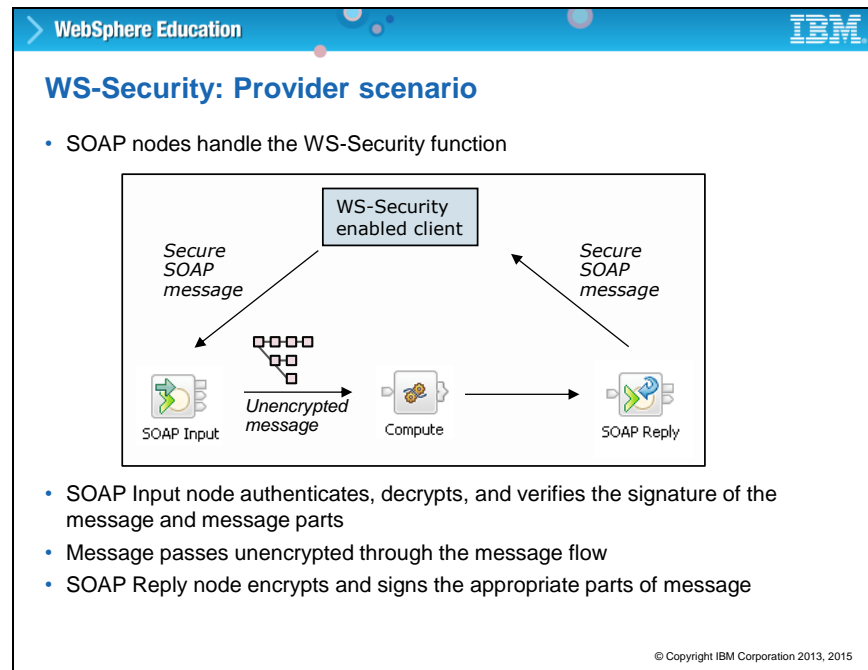
WS-Security message part protection

You protect the message parts by specifying elements and namespaces within the header or body of the SOAP message that you want to sign or encrypt.

Because message part protection can specify the header, it can be used with user name tokens to protect the user name and password.

The example shows a credit card transaction, where the credit card details are encrypted. The section at the bottom of the figure contains the message that is sent over the transmission channel. The sensitive credit card data is shown as encrypted.

Message and message part protection typically requires that the administrator work with the developer that is familiar with the format and structure of the message data.



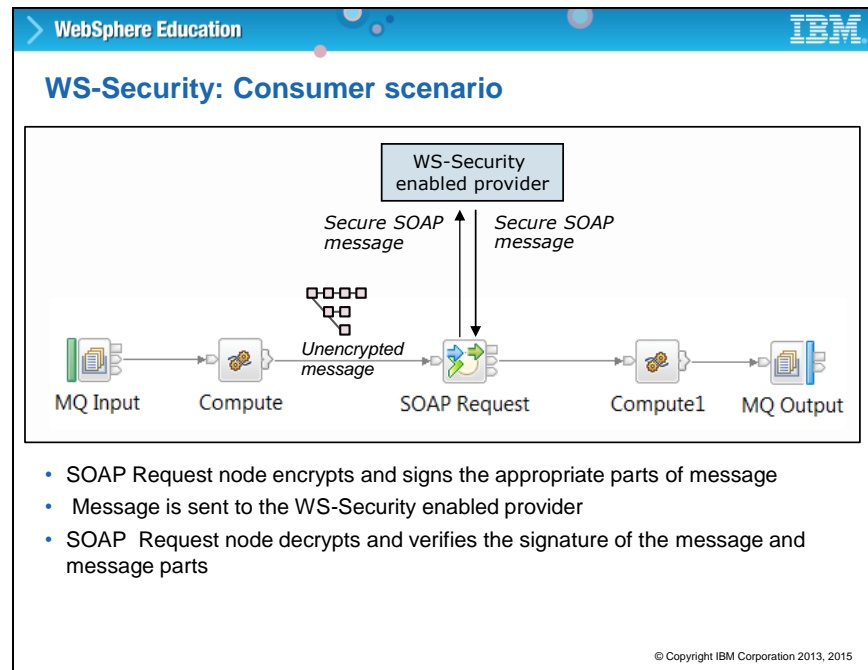
WS-Security: Provider scenario

This slide shows a scenario where the message flow represents the web service. A web service client starts the message flow by sending it a SOAP message. In this case, the SOAP message is secured by using WS-Security. The SOAP nodes handle the WS-Security function. No logic or action is required in the rest of the flow.

The role of the SOAP Input node is to verify all signed parts and decrypt the message as defined within the configuration. It does so by using the security configuration that it is given. The role of the SOAP Reply node is to apply any required WS-Security function. If the WS-Security configuration is applied to the whole message, then the entire message is decrypted for use by the message flow.

If the WS-Security configuration defines parts of the message, then the SOAP Input node decrypts these parts for use by the message flow. Any parts of the message that Integration Bus does not have permission to access because it does not contain the correct certificates remain encrypted.

When the message reaches the SOAP Reply node, the message must be signed and encrypted, according to the definition of the WS-Security configuration for the message flow.



WS-Security: Consumer scenario

This slide shows a message flow that calls a web service from the middle of a flow by using the SOAP Request node.

In this scenario, the SOAP Request node encrypts and signs the appropriate parts of the message. The message is then sent to the WS-Security enabled provider. When the response is received from the provider, the SOAP Request node decrypts and verifies the signature of the message and message parts.

WebSphere Education

IBM

Policy sets

- WS-Security within Integration Bus is configured through policy sets and policy set bindings
- Policy set is a collection of configuration information that defines what is required for Integration Bus WS-Security
 - Express what security you want to apply to the message: authentication, integrity, confidentiality
 - Which parts of the message must be signed and encrypted
- Policy set bindings define how Integration Bus WS-Security is configured
 - What keys can be used and which security policies are required
 - Contains the physical security credentials
- Policy Set editor is provided in the Integration Toolkit

© Copyright IBM Corporation 2013, 2015

Policy sets

Policy sets and bindings define and configure WS-Security requirements, supported by Integration Bus, for the SOAP nodes.



A *policy set* is a container for the WS-SecurityPolicy type.

A *policy set binding* is associated with a policy set and contains information that is specific to the environment and operating system, such as information about keys.

You administer policy sets and bindings from the Integration Toolkit or by using the `mqsicreateconfigurableservice` command. In the Integration Toolkit, you can add, delete, display, and edit policy sets and bindings for an integration node.


Any changes to policy sets or bindings are saved directly to the associated integration node. You must stop and then restart the message flow for the new configuration information to take effect.

A default policy set and policy set binding is provided with Integration Bus.




Administrative versus development responsibilities

- Policy set configuration is handled primarily as an administrative task
 - The administrator can handle most of the process
 - Authentication, body encryption, and signature



- For message part protection configuration consists of development and administration tasks
 - Nodes allow the developer to specify elements that require message part protection applied
 - Administrator decides how the part of the message is to be encrypted or signed



© Copyright IBM Corporation 2013, 2015

Administrative versus development responsibilities

The implementation of the security requirements for a particular application is split between the application developer and the system administrator.

Most of the implementation is done as an administration task. For message-level protection, the administrator can do all the necessary tasks to implement WS-Security. If you need to implement message part protection, you can specify which parts of the message should be protected.

The policy set editor allows the administrator to specify how each of these message parts is to be encrypted or signed. However, administrators might not be familiar with the message tree contained within the SOAP header and SOAP body, so they need the developer to specify the elements in the message tree that must be protected.

WebSphere Education IBM

Defining message part protection on the node

SOAP node Properties in the Integration Toolkit

Alias	XPath Expression
part_encrypt	\$Root/soapenv:Envelope/soapenv:Body/echoOperation

Policy Set Editor in the Integration Toolkit

Name	Security Type	SOAP Message
part_encrypt	Encryption	Request

- XPath expressions on the SOAP nodes allow the specification of the parts of the message that must be protected
- LocalEnvironment can override


© Copyright IBM Corporation 2013, 2015


Defining message part protection on the node

This slide shows how the properties of the SOAP nodes are used to specify message part protection on parts of the SOAP message.

On the SOAP node **WS Extensions** property tab, an alias is added for each message part that is subject to security. This alias can be contained in the header of the message or the body of the message. You use an XPath expression to specify the parts of the message that must be protected.

The alias provides a link between the configuration in the SOAP nodes and the runtime components that the system administrator specifies. The Integration Bus administrator defines the Policy Set, which resolves the alias to either encrypt or sign the part of the message that the XPath expression references. Specifying the policy set is an administrative function. You cannot define or reference it with the SOAP node properties.



WebSphere Education 

Assigning policy sets with flows and nodes

- Different web services can have different requirements on WS-Security
 - Different services can use different certificates
 - Different messages (for differing uses) can require different parts of the message to be encrypted
- A single policy set and policy set binding is not sufficient for all nodes in all flows
- Policy set can be assigned at the flow or node level, depending on the required granularity

© Copyright IBM Corporation 2013, 2015

Assigning policy sets with flows and nodes

It is possible to define more than one policy set and policy set binding for each instance of an integration node.

The message flows deployed to that integration node can have different requirements for web services security. Each service can use different certificates and have different requirements for message part processing. Therefore, it is necessary to be able to assign a policy set and binding to a specific set of web services or message flows.

The policy set can be assigned at the flow level, or if the requirements are more granular, at the node level.

WebSphere Education

Assigning policy sets and bindings in the BAR file editor

- Select the message flow in the **Manage** tab to show the **Consumer Policy Set** and **Provider Policy Set** properties
- Click the **Edit** button to select a previously defined Policy Set and Policy Set Binding
- Can also set on a SOAP Input, SOAP Request, and SOAP AsyncRequest node for more control

© Copyright IBM Corporation 2013, 2015

Assigning policy sets and bindings in the BAR file editor

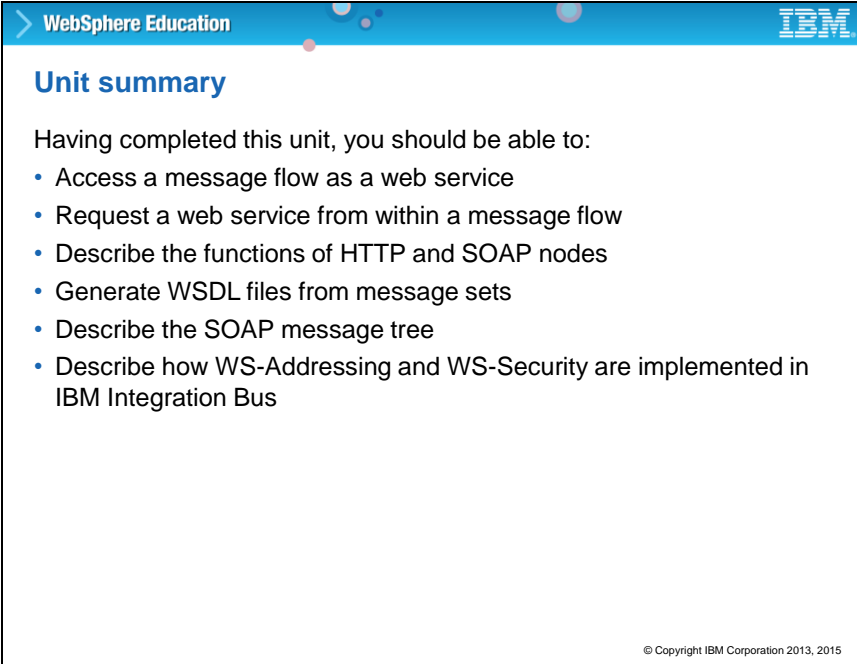
Policy sets are associated with a message flow or a message processing node in the BAR file.

For convenience, you can specify settings for provider and consumer at the message flow level. The provider setting applies to all SOAP Input and SOAP Reply nodes in the message flow. The consumer setting applies to all SOAP Request, SOAP AsyncRequest, and SOAP AsyncResponse nodes.


Individual policy set and binding assignments can be applied at the node level in the BAR file editor, and these take precedence over the flow-level provider and consumer settings.

The default setting is **none**, meaning that no policy set and bindings are to be used.

At run time, it is the responsibility of the administrator to ensure that the required policy sets are available to the integration node. If the integration node cannot find the associated policy set or bindings, an error is reported.



The slide features a blue header bar with the text 'WebSphere Education' on the left and the IBM logo on the right. The main content area is white with a blue title 'Unit summary'. Below the title, it states 'Having completed this unit, you should be able to:' followed by a bulleted list of six items. At the bottom right, there is a small copyright notice: '© Copyright IBM Corporation 2013, 2015'.

> WebSphere Education 

Unit summary

Having completed this unit, you should be able to:

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

© Copyright IBM Corporation 2013, 2015

Unit summary

You can use IBM Integration Bus nodes and services to connect to other web services providers and consumers. This unit described the use of web services transport and WSDL generation from message definitions. Handling messages with SOAP formats was also covered.

Having completed this unit, you should be able to:

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