

SOAP nodes in IBM WebSphere Message Broker V6.1, Part 3: Configuration details

Skill Level: Intermediate

[Rob Henley \(rhenley@uk.ibm.com\)](mailto:rhenley@uk.ibm.com)
Software Developer
IBM

[Matthew Golby-Kirk \(mgk@uk.ibm.com\)](mailto:mgk@uk.ibm.com)
Software Developer
IBM

04 Dec 2008

So far in this four-part article series, you've learned about the basic use of SOAP nodes and the new logical tree format used by the SOAP domain. This article, Part 3 in the series, describes the detailed configuration of the nodes using Web Services Description Language (WSDL). You should have a general familiarity with SOAP-based Web services and WSDL to follow along with this article series. **Note:** This article relates to IBM® WebSphere® Message Broker V6.1 Fix Pack 6.1.0.2. Some details could differ slightly from the 6.1 GA version.

WSDL configuration

WSDL is the accepted definition language for Web services, so the SOAP nodes are configured using WSDL. WSDL definitions can optionally be split into multiple files. The typical arrangement is that a top-level service definition file imports a binding file, the binding file imports an interface file, and this interface file imports or includes schema definition files. (See the [WSDL topology](#) section for more details.) If you *do* use multifile WSDL, you must use the service definition to configure the node, allowing endpoint properties, such as URL, to be set. (If you try to configure a node using just a binding definition, you'll get the error `Service port: you must select a binding that has at least one port.`) The overall use of WSDL is described in the section [WSDL life cycle in the broker](#).

Deployable WSDL

The SOAP nodes are configured using *deployable WSDL*. This is WSDL that has been imported into, or generated in, a message set and that then appears under the category Deployable WSDL in the tooling. The schema definitions for deployable WSDL are held exclusively in message definitions within the message set.

The immediate concern is how to get the WSDL definition required to configure the SOAP nodes. There are two cases: Either you already have a WSDL definition or you need to create one.

Use an existing WSDL

If you have an existing WSDL definition, you can do one of the following:

- Import the WSDL into an existing message set by clicking **New > Message Definition File From > WSDL File**. This option is available on the regular File menu, or via the right-click pop-up menu from a message set selected in the Broker Development pane. **Note:** This is *not* the same as selecting **File > Import**, which only lets you import the WSDL files into your workspace and does not result in deployable WSDL.
- Use the Quick Start wizard *Start from WSDL and/or XSD files*, which lets you create a message flow and message set from scratch.

Note: Your WSDL must include a binding definition. Some products, such as IBM WebSphere Integration Developer, let you work with WSDL definitions that only have a `portType`. You need to add a binding and service to such a definition before you can import it into the broker to create deployable WSDL.

Create a new WSDL

Alternatively, if you have a message set that defines the payload of your Web service messages, you can generate a new WSDL definition from your message set. You generate WSDL by right-clicking your message set, and then selecting **Generate > WSDL Definition > Generate a new WSDL definition from existing message definitions**.

If you want to make the WSDL only externally available, you can now select **Export to an external directory**. However, if you intend to use the WSDL to configure SOAP nodes, then select **Create in a workspace directory**. **Note:** You should select the message set folder beneath your message set project. For example, if your message set project is called `myMessageSet`, select **myMessageSet/myMessageSet**. (Otherwise the generated WSDL doesn't appear under the Deployable WSDL category.)

If you also need to make the same WSDL available to an external toolkit (for

instance for use with a code generator), then you should first generate the WSDL in your message set, and then export it as a separate step. Export WSDL by right-clicking, and then selecting **Generate > WSDL Definition > Export an existing WSDL definition to another directory**. You can then select the required file format and WSDL style.

Note: This is *not* the same as selecting **File > Export**, which only lets you copy individual selected files from your workspace and paste them to the file system, and may not result in well-formed stand-alone WSDL.

Finally, there's also a wizard for creating a new WSDL definition from scratch. (You access this by selecting **New > Other > Web Services > WSDL**.) However, if you want to use this, be aware that the resulting WSDL is not deployable WSDL, because inline schema definitions are added to the `<types>` section. If you want to create a new WSDL definition this way, then after you've created it you have to import it as a separate step, as described earlier, to create deployable WSDL.

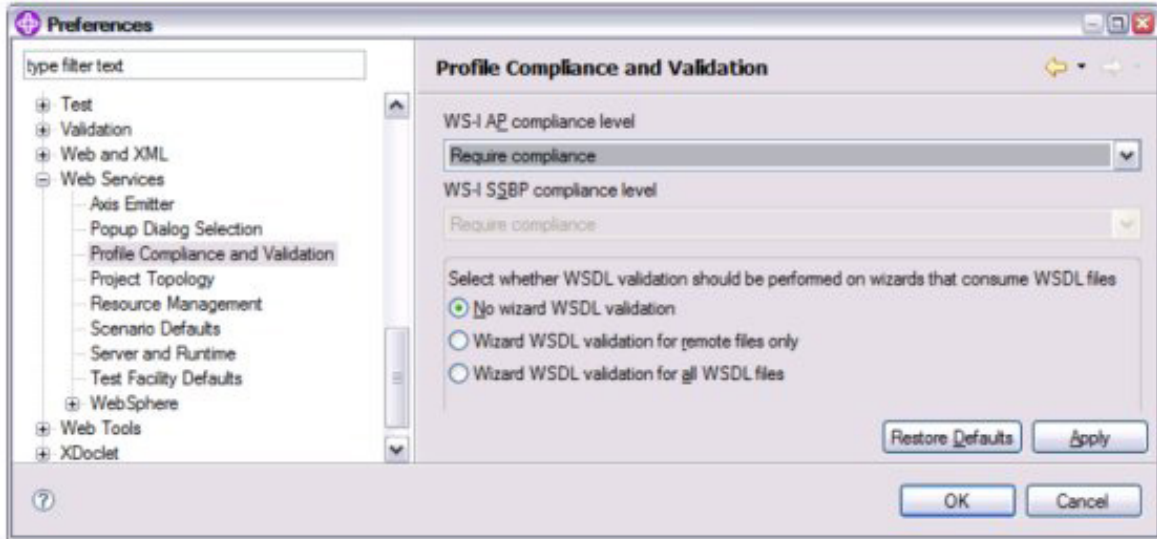
WSDL validation: development time

A WSDL validator checks WSDL against the WS-I Basic Profile. A WSDL definition is far less likely to cause interoperability problems if it's WS-I compliant.

The validator can be invoked automatically when the WSDL importer or WSDL generator is run, or you can invoke it manually via the tooling (for example, the pop-up menu on a .wsdl file). Typically you only need to validate the WSDL definition once, but you should rerun the validation if you modify the WSDL file. (If you see the error `Referenced file contains errors` for a file in a multipart WSDL, then you should run the validator manually on the referenced file(s) to see the detailed messages.)

You can specify when the validator should be run and which compliance level should be used under the tooling Preferences (**Window > Preferences**), as shown in Figure 1.

Figure 1. WSDL validation preferences

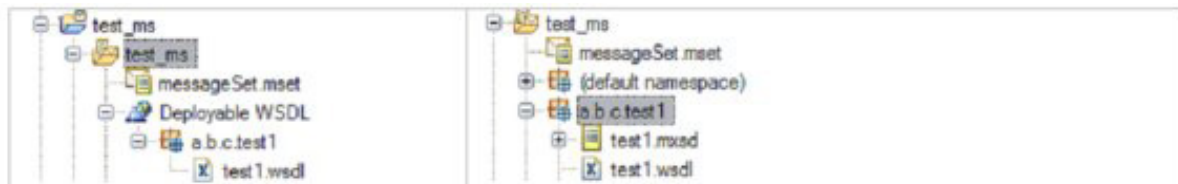


By default the WS-I Attachments Profile (WS-I AP) is selected, but if this is set to **Ignore compliance**, then you can choose to **Require** or **Suggest** compliance with the less comprehensive Simple SOAP Binding Profile (WS-I SSBP). (See [Resources](#) for more information on SSBP.) You should generally leave the default, which allows the validator to check the conformance of MIME (SOAP with Attachments) bindings in the WSDL, in addition to regular SOAP bindings.

Deployable WSDL and flow configuration

Having imported or generated your WSDL, you should have a message set containing deployable WSDL. Usually this appears under the category Deployable WSDL (left side in Figure 2), but if **Hide Categories** is checked, then note that the Deployable WSDL category is not shown (right side in Figure 2). (Hide Categories is off by default, but can be selected via the pop-up menu of the message set.)

Figure 2. Deployable WSDL



The deployable WSDL can then be used to configure SOAP nodes (SOAPInput, SOAPRequest, or SOAPAsyncRequest) in a message flow, either by dragging the WSDL from the resources pane directly onto the node in the Message Flow Editor (see Figure 3) or by browsing for the WSDL on the Basic tab of the node Properties (Figure 4).

Figure 3. Drag configuration of a SOAPInput node

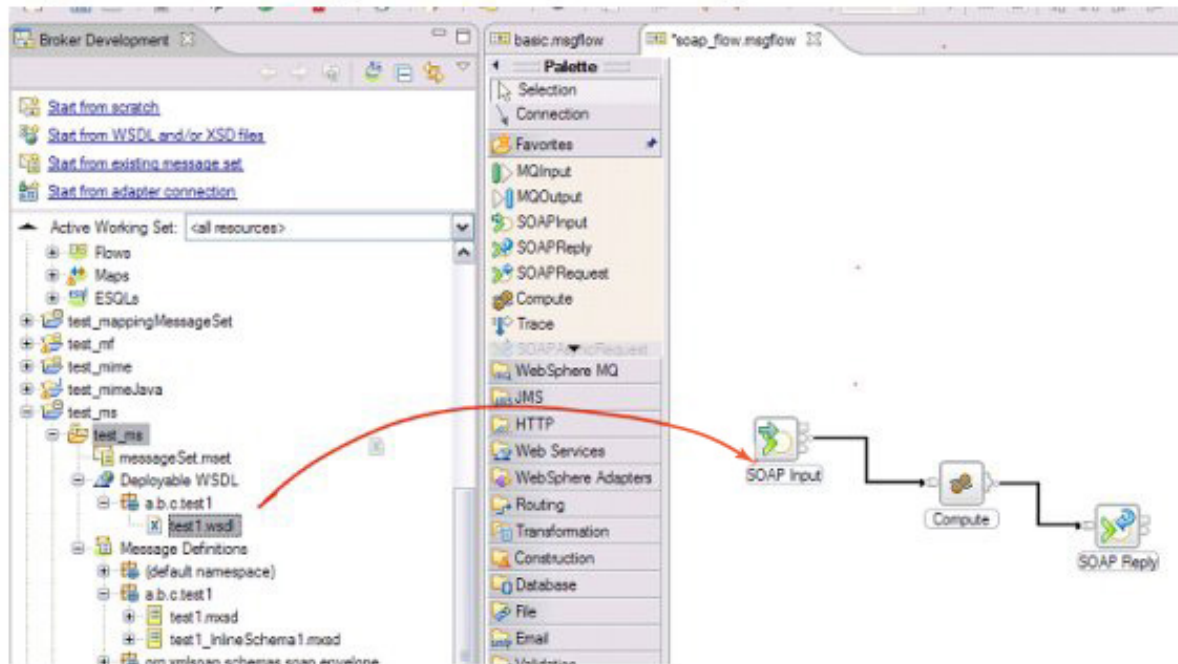
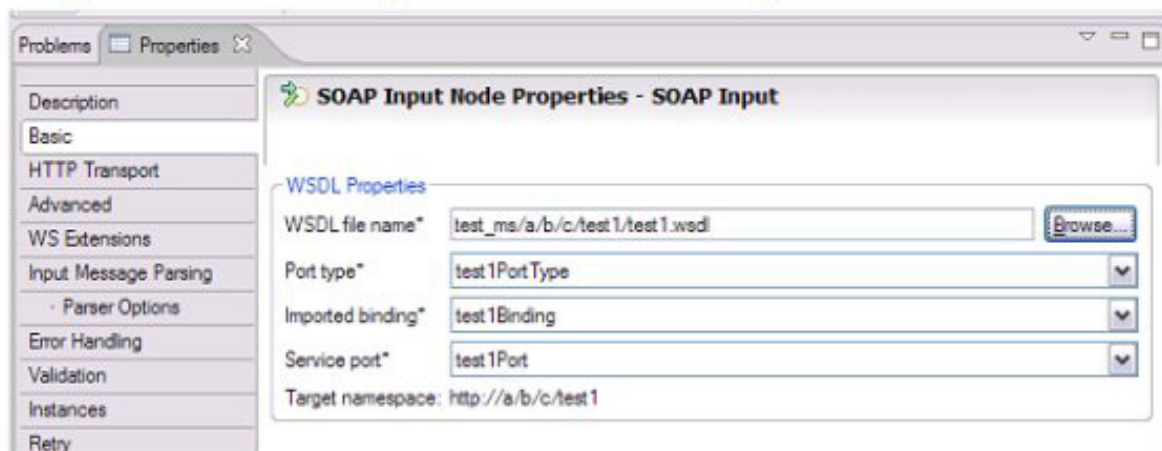


Figure 4. Properties configuration



You must always select a specific WSDL binding. And for a SOAPRequest or SOAPAsyncRequest node, you must always select a specific operation, too.

Often you'll want to create your own message flow and then configure the nodes as just described. However, as a convenience you can also create a new skeleton message flow by dragging a WSDL definition onto a blank Message Flow Editor canvas. When you do this, you also choose the type of flow (service provider or consumer) and the operations to be handled by the flow.

In addition to the WSDL file name itself, other WSDL-derived default values are set automatically on the Basic tab when the node is configured, but can be overridden by the user.

Role of the message set and WSDL Editor

Logically, the WSDL configures your message flow. But it's also important to understand the significance of the message set itself. Whether you import or generate the WSDL, your message set contains both the .wsdl resource and .mxsd resources representing the Web services messages described by that WSDL. At development time, these message definitions support the use of extended SQL (ESQL) content assist and the creation of mappings. At run time they allow schema validation of the detailed message content.

If you edit a deployable WSDL resource using the WSDL Editor, you'll see that you can navigate from the graphical Design view of the WSDL to the underlying XML schema definitions represented by the message set.

Figure 5. WSDL Editor

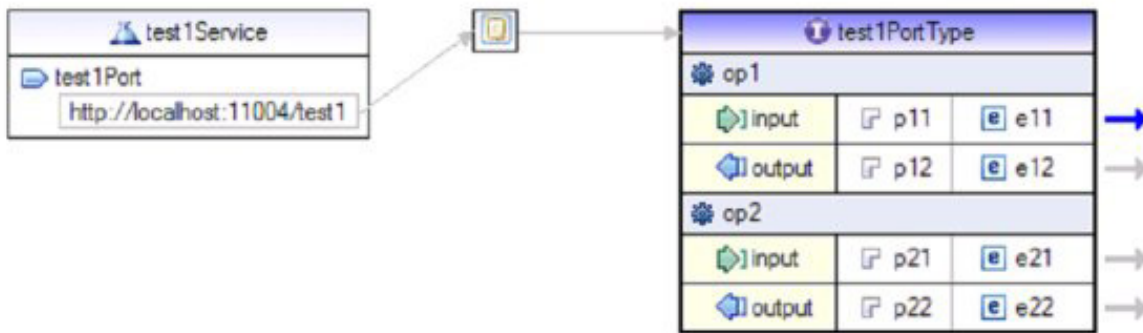


Figure 5 shows a service with two operations: `op1` and `op2`. The input message for `op1` is described by the XML schema element `e11`. Clicking the arrow to the right of `e11` opens the Message Definition Editor on the .mxsd containing that definition.

Terminology

- **W3C:** The World Wide Web Consortium, the body that publishes the Web services standards.
- **SOAP:** The W3C standard XML message format for Web services messages.
- **SOAP with Attachments (SwA):** The W3C standard for Web services that need to incorporate attachments, such as image data, in their messages.
- **SOAP domain:** The broker domain for working with Web services messages. The messages are represented in a message flow using the **SOAP domain logical tree**.
- **Web Services Description Language (WSDL):** The W3C standard for describing a Web service.

- **Deployable WSDL:** The broker WSDL representation used to configure SOAP domain nodes. All the schema definitions for deployable WSDL are held in broker message definitions. Deployable WSDL can be created by importing regular WSDL definitions. Likewise, deployable WSDL can be exported as regular WSDL for external consumption.
- **Multipurpose Internet Mail Extensions (MIME):** A message format for multipart messages and the underlying format for SwA and MTOM.
- **WS-Addressing:** A W3C specification that describes how to specify identification and addressing information for messages. It provides a correlation mechanism and lets more than two services interact.
- **MQ:** IBM WebSphere MQ.

You can use the WSDL Editor and Message Definition Editor to modify your WSDL definition if necessary. (A word of warning: You should treat the graphical view of the editor as a viewer and use the source view if you want to make updates. If you modify the WSDL using the graphical view, then you can add inline schema definitions, which are not allowed in deployable WSDL.)

If you change the `targetNamespace` of your WSDL, the name of any port, `portType`, binding or operation, or the style (`rpc` or `document`), then you must re-apply the WSDL to any SOAP nodes that use it. Otherwise you can't add the affected message flows to a BAR file (you'll see a message saying that your WSDL has errors).

SOAPInput and WSDL

The expectation is that a message flow beginning with a SOAPInput node can handle all the operations in the binding, which was selected when the node was configured. Each `portType` is bound to a specific endpoint by a WSDL port. This endpoint information is used to dispatch requests to the node, and should, therefore, be unique. (If multiple SOAPInput nodes are associated with the same endpoint, then the broker can't tell which node to dispatch to.)

Each SOAPInput node is configured by a single WSDL file. If you need to implement several versions of the same WSDL, you can configure multiple SOAPInput nodes in the same flow as long as each node uses a different Uniform Resource Identifier (URI). The SOAPInput nodes can share a single SOAPReply node if required; this may improve the readability of the flow. Similarly, each SOAPInput node uses either HTTP or HTTPS, so ideally you shouldn't mix Secure Sockets Layer (SSL)-secured operations and non-SSL operations in the same `portType`. But if this is unavoidable, you can configure two separate SOAPInput nodes with the same

WSDL, defining one to use HTTP and the other to use HTTPS.

A SOAPInput node determines the specific operation from the message payload, that is, the name and namespace of the first child of the SOAP body. For a rpc-style WSDL, the payload is guaranteed to be unique to an operation because it's the WSDL operation name itself. But it's possible to construct a document-style WSDL with the same payload for more than one operation. (The WSDL validator warns you at development time that the operations aren't unique.) In this case you must ensure that the WSDL defines a unique SOAPAction or WS-Addressing Action for each operation; otherwise a SOAPInput node can't distinguish between the operations.

Deployment

Your message flow should be deployed along with the message set containing the WSDL that configured the SOAP nodes. (In fact, you can deploy the message set first if you want to, but it's easier to add your message flow and message set to the same Broker Archive file.) At run time, the message set is then used to validate the Web service messages that are sent and received by the message flow.

If you want to, you can modify properties for individual nodes using the BAR file editor (**Configure** tab). This is particularly relevant for security configuration.

WSDL topology

As mentioned earlier, a WSDL definition consists of four logical sections:

- services
- portTypes
- bindings
- XML schema definitions

These logical sections can be written as separate files (related using import and include statements) or as a single file. Some products have difficulty importing the multifile format, although support is improving. If you use the multifile format, you have the option of specifying a different `targetNamespace` for each file. This is often inadvisable because it makes the WSDL harder to maintain (for instance, each file is written to a separate directory in a broker message set) and may impair its consumability by other tools, whilst conferring no particular benefits.

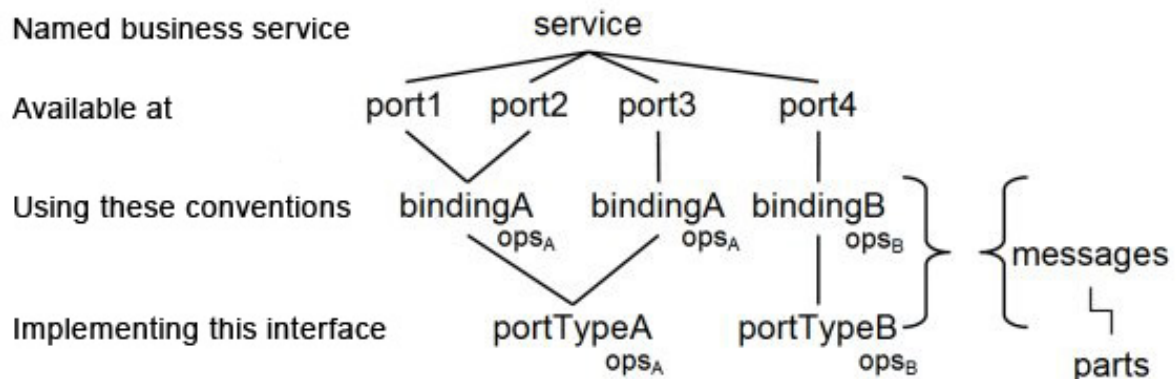
In terms of SOAP node configuration, a top-level service definition is always required.

Typically a WSDL definition has a single service definition with a single `portType` (the logical interface provided by the service) and one or more `binding` (the physical instances of the logical interface). But it's also possible for a WSDL definition to include multiple `portTypes`, either as variations of a single interface or as multiple distinct interfaces.

A binding defines a `use`, which may be `document` (the default) or `rpc`. If the `use` is `document`, then the SOAP payload is described by an XML schema element in the WSDL. If the `use` is `rpc`, then the SOAP payload is the WSDL operation name in a specified namespace.

The bindings are related to the service by port definitions. The bindings and `portTypes` are related to the XML schema by the WSDL message and part definitions. Figure 6 shows the relationship of the various parts of a WSDL definition.

Figure 6. WSDL organization



- `port1`, `port2`, `port3`, and `port4` offer the same named service at different locations.
- `port2` provides the same service as `port1` (same logical interface and wire format).
- `port3` provides the same logical service as `port1` and `port2` (same logical interface), but with a different binding (for example, it might be SOAP/MQ instead of SOAP/HTTP).
- `port4` has a different `portType`, implying that a different set of operations (`opsB`) is supported. This can mean that `port4` offers a variation of the same logical interface. For example, you might want to offer a richer service implementation over your intranet than the one available publicly. Alternatively, it can mean that `port4` offers a completely distinct interface.

Note: Each port should generally be configured with a different location. A `SOAPInput` node implements a specific `portType` and binding, but requests are dispatched to it based on the port location.

WSDL life cycle in the broker

The following sections describe the life cycle of WSDL from import or generation through deployment.

WSDL import or generation

WSDL is imported into a message set, or a new WSDL is generated from a message set. In both cases, the result is a message set ready for use with the SOAP domain, meaning that the message set contains the definitions needed to describe the SOAP messages expected at run time.

The following message definitions (mxsds) are added to the message set:

- (On import) definitions corresponding to both inlined and externally referenced `<schema>` definitions
- A SOAP envelope definition for the version of SOAP used by the WSDL
- For rpc-style WSDL (rpc-literal or rpc-encoded), definitions for the rpc-style operations and part names
- For rpc-encoded WSDL, a SOAP encoding definition for the appropriate SOAP version
- A definition of the SOAP domain tree itself (`SOAP_Domain_Msg`)

On *import*, the WSDL importer makes a deep copy of the WSDL into the message set:

- The WSDL files are copied into the message set under a directory reflecting the WSDL target namespace, defined by the `targetNamespace` attribute on the WSDL `<definitions>` element.
- The original file names are preserved unless it's necessary to change them to avoid a name clash.
- All `<schema>` definitions are replaced with `<xsd:import>` references to the newly created message definition files.
- The WSDL is validated against the WS-I Basic Profile using the WSDL Validator.

On *generation*, the WSDL generator creates a new WSDL definition in the message set. This is complementary to the import case above.

WSDL export

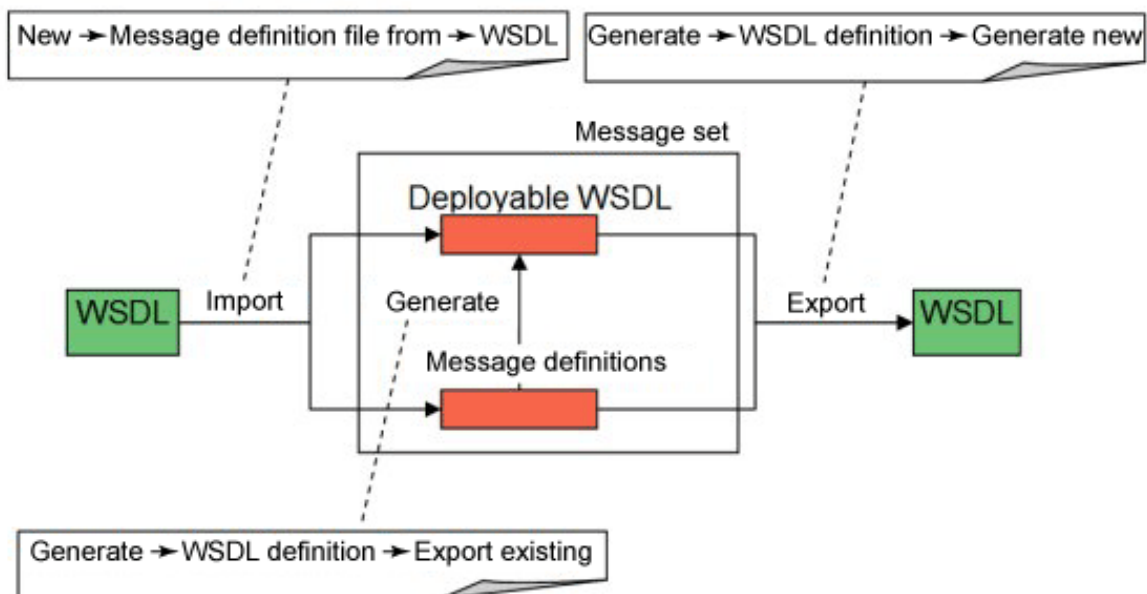
WSDL can be exported from a message set, allowing you to make a previously generated WSDL available to an external consumer or to export a previously imported WSDL. This is actually selected via the WSDL generation wizard.

Figure 7 summarises import, generation, and export, and shows how WSDL can be either:

1. Imported into a message set.
2. Generated from an existing message set (and subsequently exported if it's also required for use by an external application).

In both cases, the resulting deployable WSDL in the message set can be used to configure the SOAP nodes.

Figure 7. WSDL import, generation, and export



WSDL configuration

WSDL is used to configure SOAP nodes in a message flow, allowing you to create a message flow with a direct correlation to the WSDL that describes the messages it needs to handle.

BAR file creation

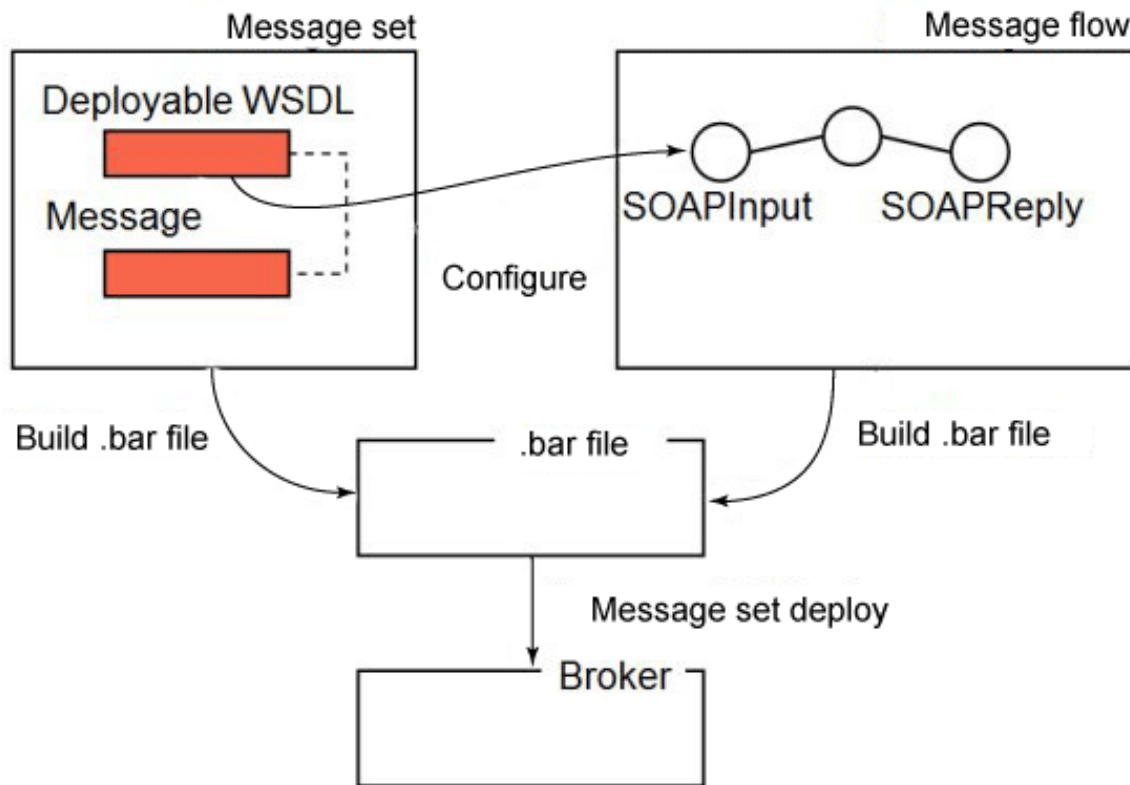
The message set containing the WSDL is added to a .bar file. Typically you add the message set and its associated message flow to the same .bar file. You can deploy the WSDL on its own if you want to, so long as it's deployed *before* the message flow that depends on it.

BAR file deployment

The .bar file is deployed to a broker, enabling the message flow to check messages against the WSDL at run time and to provide WSDL-derived information in the logical tree.

Figure 8 summarises configuration, BAR file creation, and deployment, showing how the message set and message flow are deployed to the broker in a broker archive file.

Figure 8. Configuration and deployment



Conclusion

In this article, Part 3 of the [series](#), you've seen a detailed description of the configuration of the SOAP nodes and use of WSDL. Part 4 will describe the resulting runtime behavior.

Resources

Learn

- Read the other articles in this series:
 - Part 1: [SOAP node basics](#) (developerWorks, Jun 2008)
 - Part 2: [The SOAP domain logical tree](#) (developerWorks, Jul 2008)
- Learn more about the [IBM WebSphere Message Broker](#) and [what's new](#) (developerWorks, Jan 2008).
- Learn more about the [WS-Addressing](#) and [WS-Security](#) specifications.
- Visit the [W3C site](#), which hosts the specifications for SOAP, SwA, and MTOM, as well as WSDL and the WSDL 1.1 Binding Extension for SOAP 1.2.
- Get specifications for WS-Addressing from the [W3C](#). There are two main versions, usually referred to as the [Submission version](#) and the Final version, which is described across 3 documents: [Core](#), [SOAP Binding](#), and [Metadata](#). WebSphere Message Broker supports both versions, but defaults to the Final version.
- [Get offers the specifications for the Basic profile and attachments profile](#) from the WS-I.
- The [SOA and Web services zone](#) on IBM developerWorks hosts hundreds of informative articles and introductory, intermediate, and advanced tutorials on how to develop Web services applications.
- Play in the [IBM SOA Sandbox!](#) Increase your SOA skills through practical, hands-on experience with the IBM SOA entry points.
- The [IBM SOA Web site](#) offers an overview of SOA and how IBM can help you get there.
- Stay current with [developerWorks technical events and webcasts](#).
- Browse for books on these and other technical topics at the [Safari bookstore](#).
- Check out a quick [Web services on demand demo](#).
- Get an [RSS feed for this series](#). (Find out more about [RSS](#).)

Get products and technologies

- Download a [trial version of WebSphere Message Broker V6.1](#).
- Download [IBM product evaluation versions](#) and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- [Participate in the discussion forum for this content.](#)
- Get involved in the developerWorks community by participating in [developerWorks blogs](#).

About the authors

Rob Henley

Rob Henley is a software developer on the WebSphere Message Broker development team at the IBM Hursley Software Lab in the UK. He works on the design and implementation of support for Web services in WebSphere Message Broker.

Matthew Golby-Kirk

Matthew Golby-Kirk is a software developer working on the WebSphere Message Broker development team at the IBM Hursley Software Lab in the UK. He works on the design and implementation of the HTTP and Web services support, along with the ESQL language run time in WebSphere Message Broker.

Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of [IBM trademarks](#).

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.