

Encoding of SOAP Operations

SOAP uses an XML format to marshal data that is transported between software applications. SOAP was planned to be used for legacy systems and modern object-oriented systems as well. Consequently, SOAP offers more than one encoding method to convert data from a software object into XML and vice versa (see the [page of W3C](#)).

There are two ways, in which it maps high level data types like arrays, integers, floats and so on to a serialized XML format: **SOAP encoding** (also called Section 5 encoding) and **Literal encoding**. Literal encoding means that the body contents conform to a specific XML Schema. SOAP encoding uses a set of rules based on the XML Schema data types to encode the data, but the message does not conform to a particular schema.

In addition to the SOAP encoding styles, messages can be of two styles: **RPC (Remote Procedure Call) style** or **Document style**. See [SOAP Encoding Styles](#) for a tabular overview and more details about all encoding styles.

The following encodings are commonly used and supported by the Bridge:

- **SOAP Remote Procedure Call (RPC) encoding**
 - SOAP encoding
 - RPC style messages
- **SOAP document-style encoding**, which is also known as message-style or document-literal encoding.
 - literal encoding
 - document style messages

The Bridge supports SOAP RPC encoding as default. This default is used when no stereotype is set at the port type operation.

On this Page:

- [Implementing Document-style Encoding with the E2E Bridge](#)
- [Differences between Document-style and RPC Style on the E2E Bridge](#)
- [Effect of Flag wsdlPerService on the Encoding Styles](#)
 - `wsdlPerService = true`
 - `wsdlPerService = false`
 - [Defining the XML Namespace](#)

Implementing Document-style Encoding with the E2E Bridge

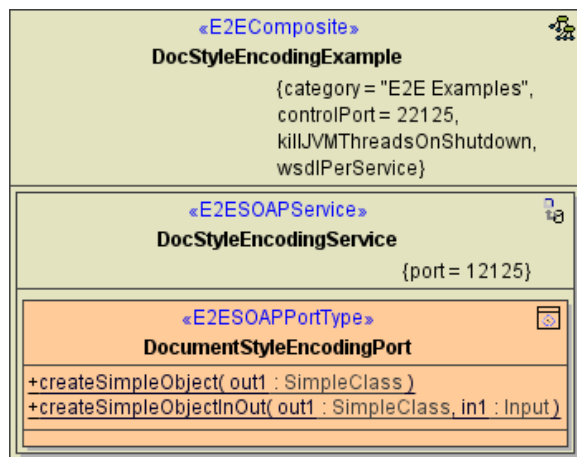
Example File (Builder project Basic Modeling/Frontend):



<your example path>\Basic Modeling\Frontend\uml\documentStyleEncoding.xml

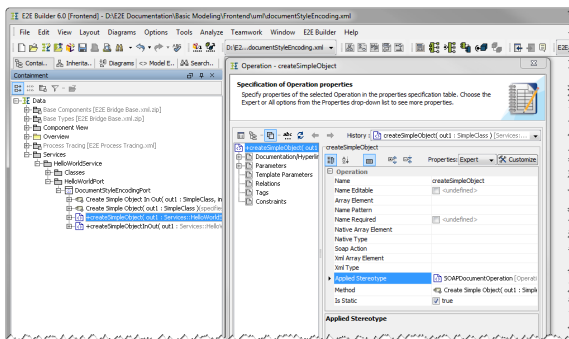
The **DocumentStyleEncodingPort** contains two SOAP operations.

Figure: Hello World Example SOAP Service



From the modeling point of view, the whole difference between RPC and document style operations is given by choosing the stereotype **<<SOAPDocumentOperation>>** on a port type operation:

Figure: Defining Stereotype for Document-style Encoding



This stereotype can also be set on backend port type operations when accessing an external Web Service.

The stereotype **<<SOAPDocumentOperation>>** needs to be set on a port type operation, if it is required by the connecting SOAP client. In a SOAP document-style call, the SOAP stack sends an entire XML document to the Bridge. The message can contain any sort of XML data that is appropriate to the deployed web service.

The parameter names of the **<<SOAPDocumentOperation>>** port type operation will correspond to XSD element names. The parameters require the accordant object flow states in the implementing activity diagram (mostly, the required classes are generated by importing the XSD file with the E2E Builder). Document-style encoded operations mostly have zero or one input and output parameters. The parameter names correspond to the root element of the input respectively output message that is transferred in the SOAP body.

Differences between Document-style and RPC Style on the E2E Bridge

The request/response stream of RPC and document style operation calls differs considerably. The following table shows examples of two such requests, both requests transporting the same information.

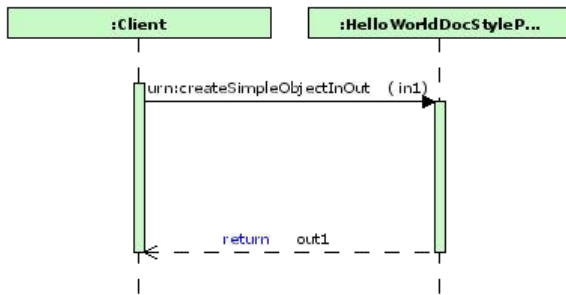
Document Style / Literal Encoding Request	RPC Style / SOAP Encoding Request
<pre> <env:Body> <in1 myKey="Hello World!"></in1> </env:Body> </env:Envelope> </pre>	<pre> <env:Body> <ns3:createSimpleObjectInOut> <in1 xsi:type="ns2:Input"> <myKey xsi:type="xsd:string">Hello World!</myKey></in1> </ns3:createSimpleObjectInOut> </env:Body></env:Envelope> </pre>

In the left-hand-side example, the SOAP envelope body contains plain (= literal) XML. Each child element of the SOAP body corresponds to an operation parameter serialized to plain XML. On the right hand side, the first and only child of the SOAP body element is the SOAP operation containing the input parameters serialized according to the SOAP encoding rules. The main differences between the two formats are as follows:

- Literal XML messages do not contain explicit type information making it more compact than SOAP encoded requests.
- The operation is explicitly given for RPC style calls. For document style requests, the SOAP operation must be derived by the SOAP action.

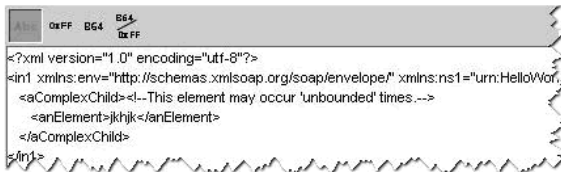
By default, the SOAP action is given by the operation name (prefixed with **urn:** to make it to an URI), but it can be overridden using the tagged value **soapAction**. This SOAP action is used as UML message name instead of the operation name when displaying sequence diagrams for document style requests in the E2E Model Debugger (see sequence diagram depicted beneath). Additionally, if sequence diagrams depict document style/literal encoding requests, the input and output messages are not rendered as UML objects but are shown in a separate text window (see the following picture below).

Figure: SOAP Action in E2E Model Debugger



Clicking on input parameter **in1** of operation **urn:createSimpleObjectInOut** will display the message in a separate pane.

Figure: Document Style Encoded Message



Effect of Flag `wsdlPerService` on the Encoding Styles

Flag **`wsdlPerService`** in the E2E composite controls the generation of WSDLs at compile time.

If true (default=false), each xUML service gets its own WSDL file. Additionally, all XML Schema elements and types having the same namespace are put into one schema file. These schema files are imported into the WSDLs to be shared among them. In this case it is also possible to mix RPC/soap-encoded services with Document/literal services.

`wsdlPerService = true`

	RPC Style Encoding	Document-style Encoding
Definitions of Types	<ul style="list-style-type: none"> Types are defined directly in the WSDL file. The WSDL only contains types, that are referenced in the service interface. 	<ul style="list-style-type: none"> Types are <u>not</u> defined directly in the WSDL file, but in XSD files that are imported to the WSDL. The Compiler generates one XSD file per namespace. The generated XSD files contain all imported types. That means, that imported XSDs are passed through as they are. If importing XSDs with equal namespaces (e.g. <code>noNamespace.xsd</code>) and if consuming services import multiple WSDLs, it could be that types get overwritten.
Definitions of Services	<ul style="list-style-type: none"> The Compiler generates one WSDL per service. Each service WSDL gets its own URL in the E2E Bridge. 	<ul style="list-style-type: none"> The Compiler generates one WSDL per service. Each service WSDL gets its own URL in the E2E Bridge.
Best Practice	Use this configuration for: <ul style="list-style-type: none"> test services, because operation signatures can directly be depicted basic services, e.g. Bridge-internal communication 	<ul style="list-style-type: none"> Decide to use namespaces, or not, but do not mix these two different approaches in the same project. Use this configuration for services that are visible externally, because this approach is very common.

wsdlPerService = false

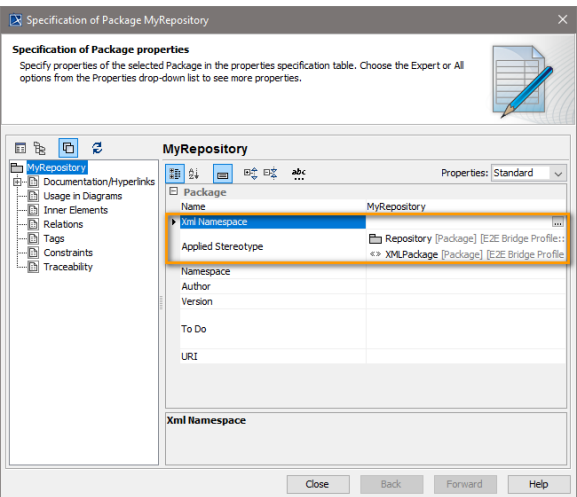
	RPC Style Encoding / Document-style Encoding
Definitions of Types	<ul style="list-style-type: none">Types are defined directly in the WSDL file.The WSDL only contains types, that are referenced in the service interface.
Definitions of Services	<ul style="list-style-type: none">The Compiler generates one WSDL file, that contains all service definitions.The Compiler generates multiple URLs for the one WSDL file to fulfill the URL convention (<code><service url?wsdl></code>).
Best Practice	<ul style="list-style-type: none">Use this configuration, if only used types should be visible in the interface.Apart from that: avoid this configuration.

Defining the XML Namespace

The Compiler uses namespaces in the following order:

1. Namespace defined on the class.
2. Namespace defined on the owning package (going up until the top of the hierarchy).
3. No namespace.

It is recommended to define the XML Namespace on the package, sub namespaces are then generated automatically. But it is also possible to set the namespace in each sub package or even on classes.



Specification of Package MyRepository

Specify properties of the selected Package in the properties specification table. Choose the Expert or All options from the Properties drop-down list to see more properties.

MyRepository

Package

Applied Stereotype: XMLPackage [Package] [E2E Bridge Profile]

Namespace

XML Namespace

To define a namespace on packages with stereotype `<<XMLPackage>>`, use tagged value `xmlNamespace`.



Specification of Class OutputClass

Specify properties of the selected Class in the properties specification table. Choose the Expert or All options from the Properties drop-down list to see more properties.

OutputClass

Class

XML

Is Ordered: ☐ undefined

Is Mixed: ☐ undefined

XML Element Name

XML Namespace