Developing Web Services with Apache Axis2

By

Kent Ka Iok Tong

Publisher:	TipTec Development
Author's email:	freemant2000@yahoo.com
Book website:	http://www.agileskills2.org
Notice:	All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.
ISBN:	978-99937-929-1-8
Edition:	Second edition March 2008

Foreword

Learn web services and Apache Axis2 easily

If you'd like to learn how to create web services (in particular, using Apache Axis2) and make some sense of various standards like SOAP, WSDL, MTOM, WS-Addressing, WS-Security, WS-Policy, XML Encryption and XML Signature, then this book is for you. Why?

- It has a tutorial style that walks you through in a step-by-step manner.
- It is concise. There is no lengthy, abstract description.
- Many diagrams are used to show the flow of processing and high level concepts so that you get a whole picture of what's happening.
- The first 46 pages are freely available on http://www.agileskills2.org. You can judge it yourself.

Unique contents in this book

This book covers the following topics not found in other books on Axis:

- How to work with Axis2 1.3.
- How to use Eclipse Europa (WTP 2.0) with Axis2.
- How to invoke asynchronous operations using WS-Addressing.
- How to encrypt and sign SOAP messages using Rampart.
- How to send user authentication information using Rampart.
- How to send and receive binary files using MTOM.
- How to integrate Axis2 with Spring.

Target audience and prerequisites

This book is suitable for those who would like to learn how to develop web services in Java.

In order to understand what's in the book, you need to know Java and to have edited XML files. However, you do NOT need to know the more advanced XML concepts (e.g., XML schema, XML namespace), servlet, Tomcat or PKI.

Acknowledgments

I'd like to thank:

- The Axis developers for creating Axis.
- The WSS4J developers for creating WSS4J.
- Anne Thomas Manes, an expert in web services, for reviewing the book (first edition).
- Helena Lei for proofreading this book.
- Eugenia Chan Peng U for doing book cover and layout design.

Table of Contents

Chapter 1

Designing the interface for a simple web service

What's in this chapter?

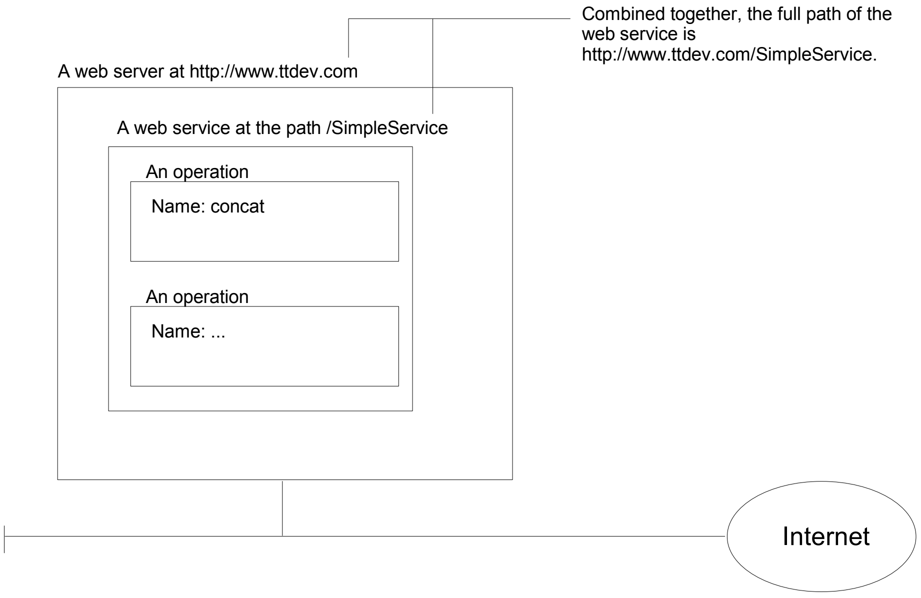
In this chapter you'll learn how to design the interface for a simple web service.

Providing cross platform operations across the Internet

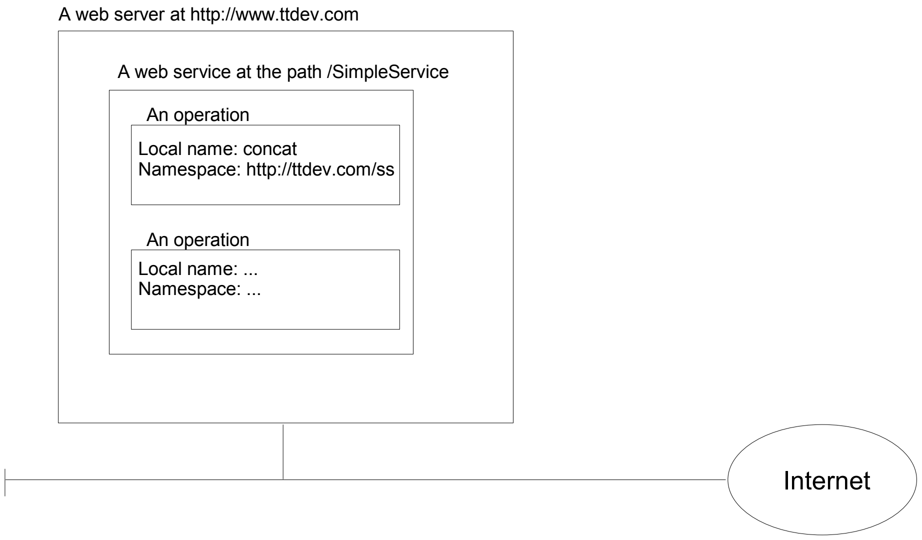
Suppose that you'd like to provide a service to the public or to some business partners: They can send you two strings and you will concatenate them and return the string. Of course, in the real world you provide a more useful service.

There are several major requirements: First, the users may be using different languages (Java, C# and etc.) and using different platforms (Windows, Linux and etc.). Your service must be accessible by different languages and platforms. Second, they will call your service across the Internet and there may be firewalls in between. Your service must be able to go through firewalls.

Given these requirements, the best solution is to provide a so-called "web service". For example, you may make a web service accessible on the host www.ttdev.com and accessible as /SimpleService (see the diagram below), so the full URL is http://www.ttdev.com/SimpleService. This is called the "endpoint" of the web service. Your web service may support one or more operations. One operation may be named "concat":


Combined together, the full path of the web service is http://www.ttdev.com/SimpleService.
A web server at http://www.ttdev.com
A web service at the path /SimpleService
An operation
Name: concat
An operation
Name: ...
Internet

However, you hope to provide a globally unique name to each operation so that you can have your "concat" operation while another person may have his "concat" operation. So, in addition to the name, you may declare that the "concat" name above is in the "namespace" of http://ttdev.com/ss (see the diagram below). A namespace is just like a Java package, but it is not in a dot format like com.ttdev.foo; it is in the format of a URL. So, the full name of the operation will be "concat" in namespace http://ttdev.com/ss. The name "concat" is called the "local name". The full name is called a "QName (qualified name)":


A web server at http://www.ttdev.com
A web service at the path /SimpleService
An operation
Local name: concat
Namespace: http://ttdev.com/ss
An operation
Local name: ...
Namespace: ...
Internet

You may wonder what this http://ttdev.com/ss namespace means. The answer is that it has no particular meaning. Even though it is a URL, it does NOT mean that you can use a browser to access this URL to get a web page (if you do, you may get a file not found error). The only important thing is that it must be globally unique. As I have registered the domain name ttdev.com, it must be globally unique.

Note that the namespace is a completely different concept from the endpoint. The endpoint really is the location, while the namespace is just a unique id. I could easily move the web service to another web server and thus it will have a different endpoint, but the namespaces of its operations will remain unchanged.

RPC style web service

Your concat operation may take two parameters. One is named "s1" and is a string. The other is named "s2" and is also a string. The return value is also a string:

An operation

```
Local name: concat
Namespace: http://ttdev.com/ss
Parameters:
   s1: string
   s2: string
Return:
   string
```

However, what does the above "string" type mean? Is it the Java string type? No, you can't say that because it must be language neutral. Fortunately, the XML schema specification defines some basic data types including a string type. Each of these data types has a QName as its id. For example:

Data type	***Local name***	***namespace***
string	string	http://www.w3.org/2001/XMLSchema
integer	int	http://www.w3.org/2001/XMLSchema
...	...	...

So, the interface of your operation should be written as:

An operation

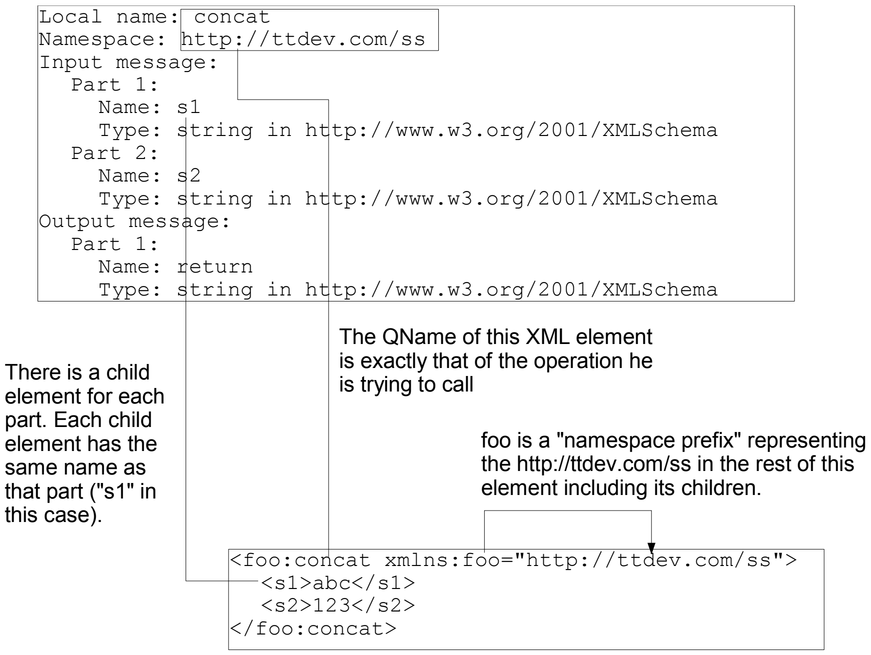
```
Local name: concat
Namespace: http://ttdev.com/ss
Parameters:
   s1: string in http://www.w3.org/2001/XMLSchema
   s2: string in http://www.w3.org/2001/XMLSchema
Return:
   string in http://www.w3.org/2001/XMLSchema
```

Actually, in web services, a method call is called an "input message" and a parameter is called a "part". The return value is called an "output message" and may contain multiple parts. So, it is more correct to say:

An operation

```
Local name: concat
Namespace: http://ttdev.com/ss
Input message:
   Part 1:
     Name: s1
     Type: string in http://www.w3.org/2001/XMLSchema
   Part 2:
     Name: s2
     Type: string in http://www.w3.org/2001/XMLSchema
Output message:
   Part 1:
     Name: return
     Type: string in http://www.w3.org/2001/XMLSchema
```

When someone calls this operation, he can send you an XML element as the input message like:



```
Local name: concat
Namespace: http://ttdev.com/ss
Input message:
  Part 1:
    Name: s1
    Type: string in http://www.w3.org/2001/XMLSchema
  Part 2:
    Name: s2
    Type: string in http://www.w3.org/2001/XMLSchema
Output message:
  Part 1:
    Name: return
    Type: string in http://www.w3.org/2001/XMLSchema
```

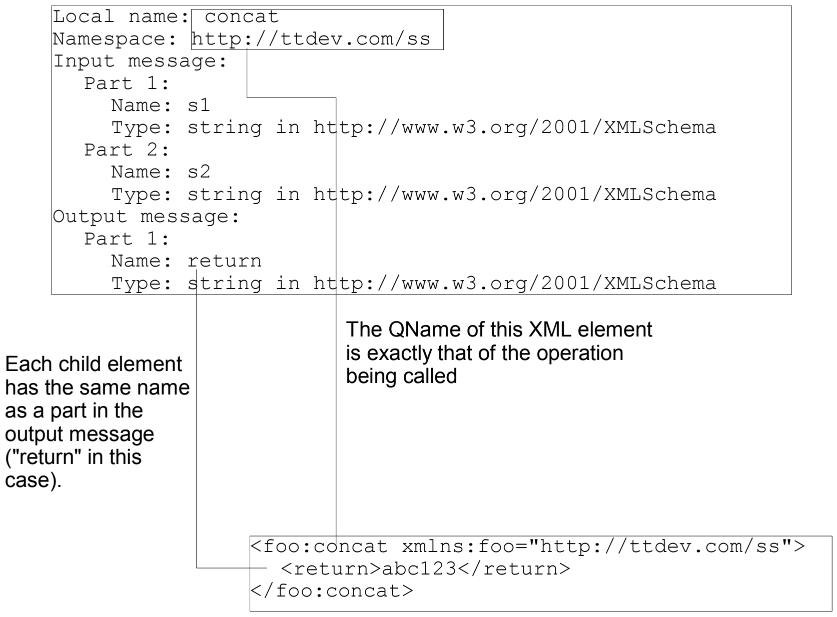
There is a child element for each part. Each child element has the same name as that part ("s1" in this case).

The QName of this XML element is exactly that of the operation he is trying to call

foo is a "namespace prefix" representing the http://ttdev.com/ss in the rest of this element including its children.

```
<foo:concat xmlns:foo="http://ttdev.com/ss">
  <s1>abc</s1>
  <s2>123</s2>
</foo:concat>
```

When you return, the output message may be like:



```
Local name: concat
Namespace: http://ttdev.com/ss
Input message:
  Part 1:
    Name: s1
    Type: string in http://www.w3.org/2001/XMLSchema
  Part 2:
    Name: s2
    Type: string in http://www.w3.org/2001/XMLSchema
Output message:
  Part 1:
    Name: return
    Type: string in http://www.w3.org/2001/XMLSchema
```

Each child element has the same name as a part in the output message ("return" in this case).

The QName of this XML element is exactly that of the operation being called

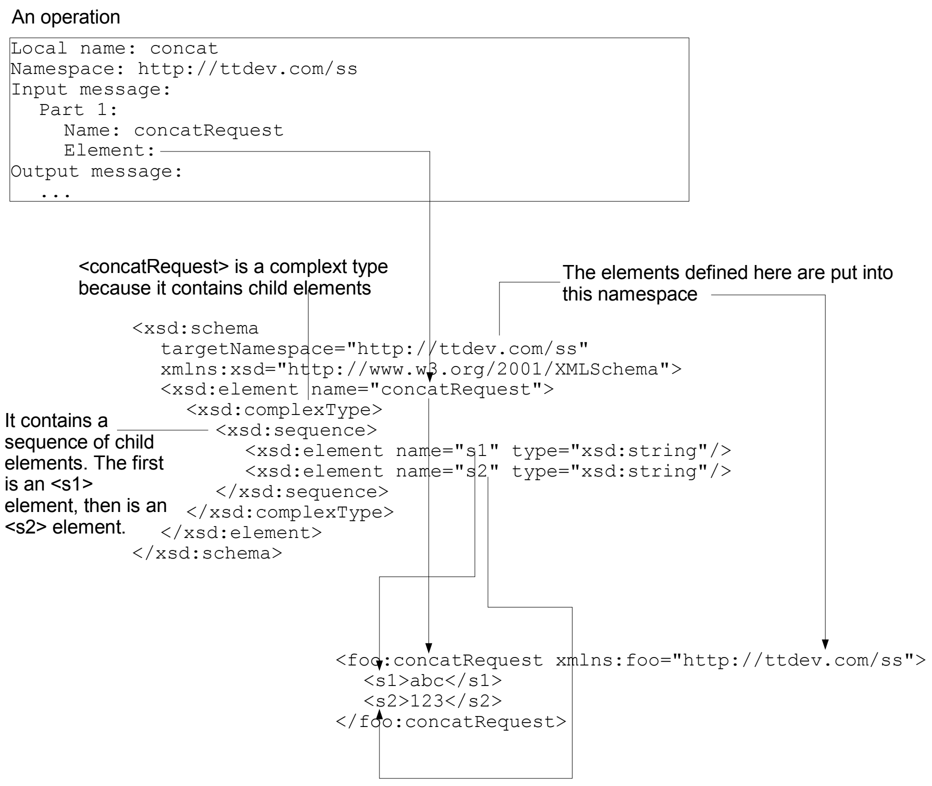
```
<foo:concat xmlns:foo="http://ttdev.com/ss">
  <return>abc123</return>
</foo:concat>
```

This kind of web service is called "RPC style" web service (RPC stands for

"Remote Procedure Call"). That is, the operation QName and the names of the parts are used to create the input and output messages.

Document style web service

The above way is not the only way you design the interface of your web service. For example, you may say that its input message only contains a single part (see the diagram below) which is an element defined in a schema. In that schema, it is defined as an element named "concatRequest" that contains two child elements <s1> and <s2>:

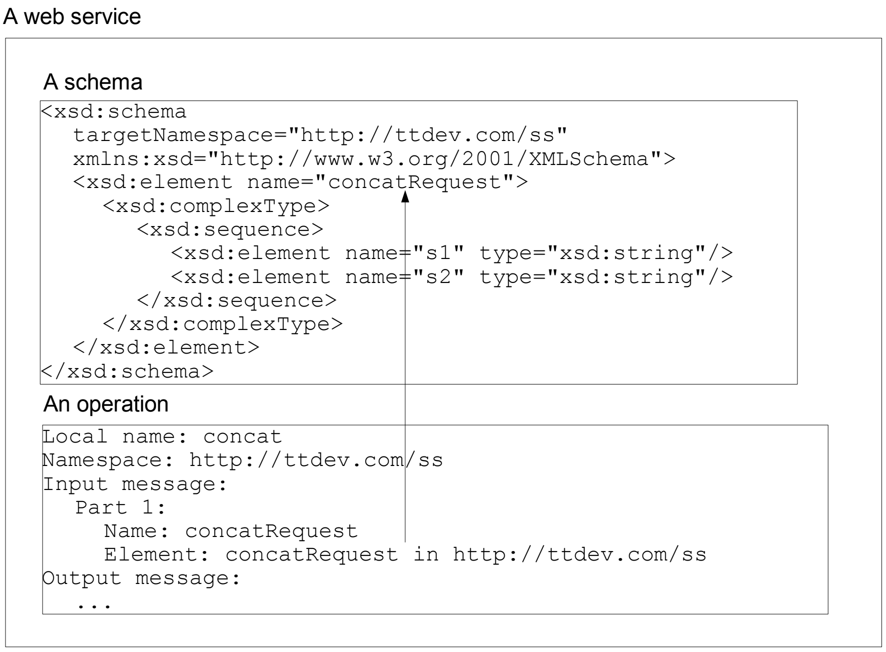

An operation
<concatRequest> is a complex type because it contains child elements
The elements defined here are put into this namespace
It contains a sequence of child elements. The first is an <s1> element, then is an <s2> element.

```
Local name: concat
Namespace: http://ttdev.com/ss
Input message:
  Part 1:
    Name: concatRequest
    Element:
Output message:
  ...
```

```
<xsd:schema
  targetNamespace="http://ttdev.com/ss"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="concatRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="s1" type="xsd:string"/>
        <xsd:element name="s2" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```
<foo:concatRequest xmlns:foo="http://ttdev.com/ss">
  <s1>abc</s1>
  <s2>123</s2>
</foo:concatRequest>
```

Note that the schema is included in the interface of your web service:



```
<xsd:schema
  targetNamespace="http://ttdev.com/ss"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="concatRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="s1" type="xsd:string"/>
        <xsd:element name="s2" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

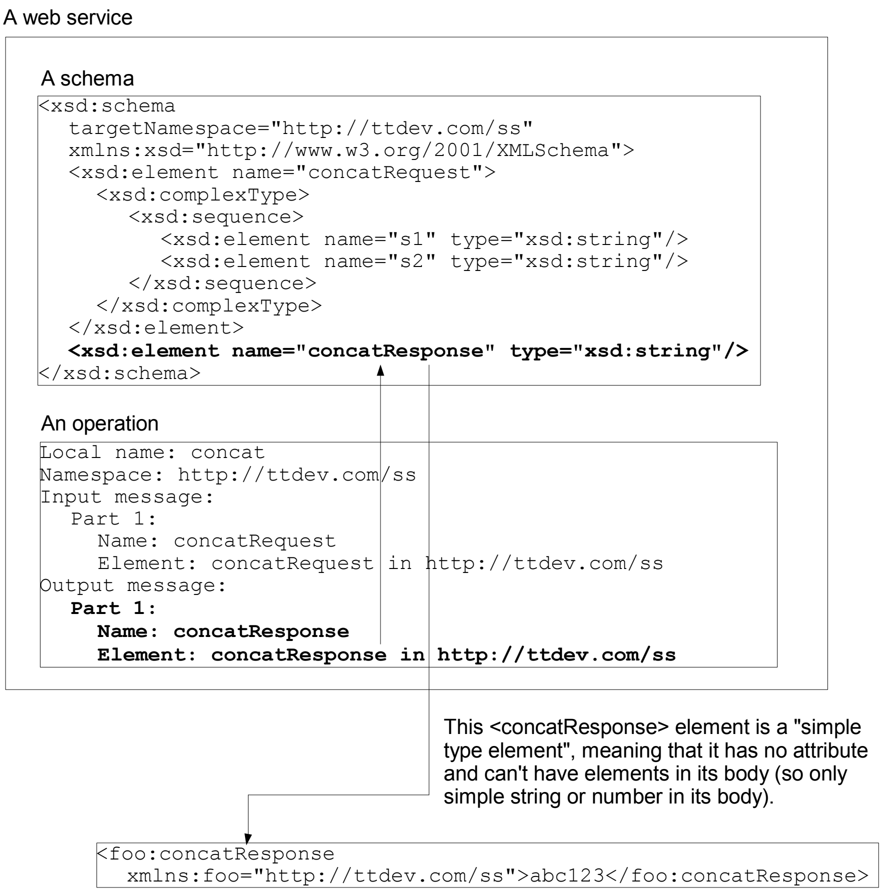
```
Local name: concat
Namespace: http://ttdev.com/ss
Input message:
  Part 1:
    Name: concatRequest
    Element: concatRequest in http://ttdev.com/ss
Output message:
  ...
```

As you can see above, a part may be declared as a particular element (<concatRequest> defined in your schema) or as any element having a particular type (string defined in XML schema specification). In either case it is identified using a QName.

When someone calls this operation, he will send you a <concatRequest> element as the input message like:

```
<foo:concatRequest xmlns:foo="http://ttdev.com/ss">
  <s1>abc</s1>
  <s2>123</s2>
</foo:concatRequest>
```

Similarly, for the output message, you may specify that it contains only one part and that part is a <concatResponse> element:



```
<xsd:schema
  targetNamespace="http://ttdev.com/ss"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="concatRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="s1" type="xsd:string"/>
        <xsd:element name="s2" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="concatResponse" type="xsd:string"/>
</xsd:schema>
```

```
Local name: concat
Namespace: http://ttdev.com/ss
Input message:
  Part 1:
    Name: concatRequest
    Element: concatRequest in http://ttdev.com/ss
Output message:
  Part 1:
    Name: concatResponse
    Element: concatResponse in http://ttdev.com/ss
```

This <concatResponse> element is a "simple type element", meaning that it has no attribute and can't have elements in its body (so only simple string or number in its body).

```
<foo:concatResponse
  xmlns:foo="http://ttdev.com/ss">abc123</foo:concatResponse>
```

This kind of web service is called "document style" web service. That is, the input message will contain a single part only which is well defined in a schema. The same is true of the output message.

If you go back to check the input message for the RPC style service, it should be revised as:

```
<foo:concat>
  xmlns:foo="http://ttdev.com/ss"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance">
  <s1 xsi:type="xsd:string">abc</s1>
  <s2 xsi:type="xsd:string">123</s2>
</foo:concat>
```

This attribute is used to explicitly state the XML data type of the body of an element ("abc" here). This is useful when the element (<s1>) itself is not defined in a schema. This "type" attribute is defined in the http://www.w3.org/2001/XMLSchema-Instance namespace, so you need to introduce a prefix for it:

This is because <foo:concat>, <s1> and <s2> are not defined in any schema and therefore you must explicitly state the XML element types of the content of <s1> and <s2>.

Now, let's compare the input messages of the RPC style web service and the document style web service:

RPC style

```
<foo:concat>
  xmlns:foo="http://ttdev.com/ss"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance">
  <s1 xsi:type="xsd:string">abc</s1>
  <s2 xsi:type="xsd:string">123</s2>
</foo:concat>
```

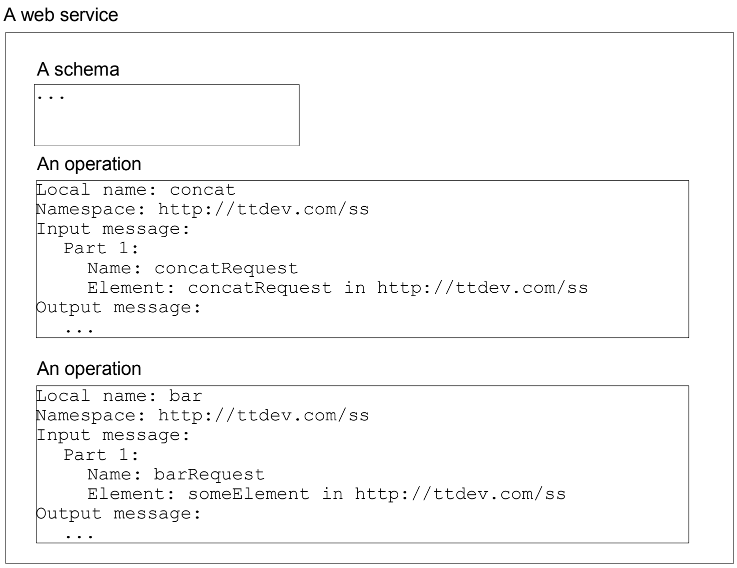
Document style

```
<foo:concatRequest xmlns:foo="http://ttdev.com/ss">
  <s1>abc</s1>
  <s2>123</s2>
</foo:concatRequest>
```

Not much difference, right? The significant difference is that the former can't be validated with a schema while the latter can. Therefore, document style web service is becoming the dominant style. According to an organization called "WS-I (web services interoperability organization)", you should use document style web services only.

Determining the operation for a document style web service

To call an operation in a document style web service, one will send the single part of the input message only. Note that it does NOT send the operation name in any way. Then if there are more than one operations in the web service (see the diagram below), how can it determine which one is being called? In that case, it will see if the input message is a <concatRequest> or a <someElement> to determine. What if both take a <someElement>? Then it is an error and it won't work:



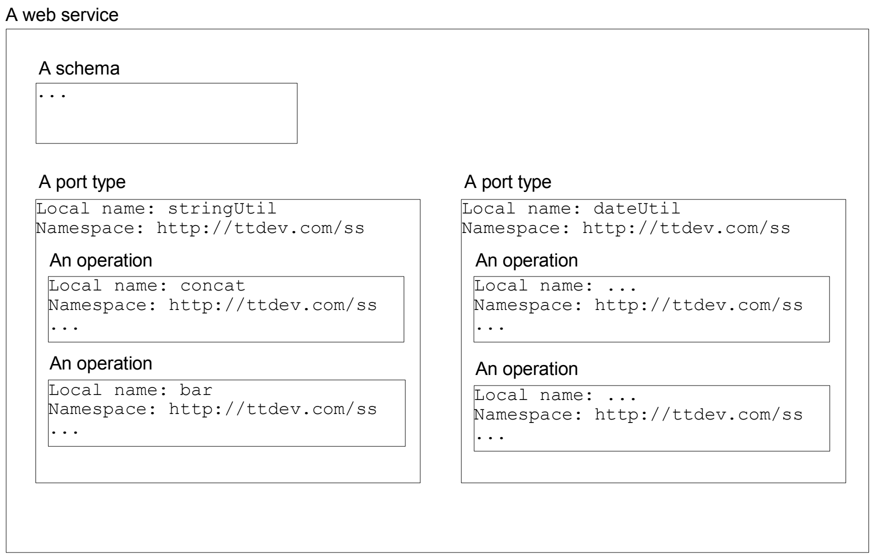
```
...
```

```
Local name: concat
Namespace: http://ttdev.com/ss
Input message:
  Part 1:
    Name: concatRequest
    Element: concatRequest in http://ttdev.com/ss
Output message:
  ...
```

```
Local name: bar
Namespace: http://ttdev.com/ss
Input message:
  Part 1:
    Name: barRequest
    Element: someElement in http://ttdev.com/ss
Output message:
  ...
```

Port type

Actually, a web service doesn't directly contain a list of operations. Instead (see the diagram below), operations are grouped into one or more "port types". A port type is like a Java class and each operation in it is like a static method. For example, in the web service above, you could have a port type named "stringUtil" containing operations for strings, while having another port type named "dateUtil" containing operations for dates. The name of a port type must also be a QName:

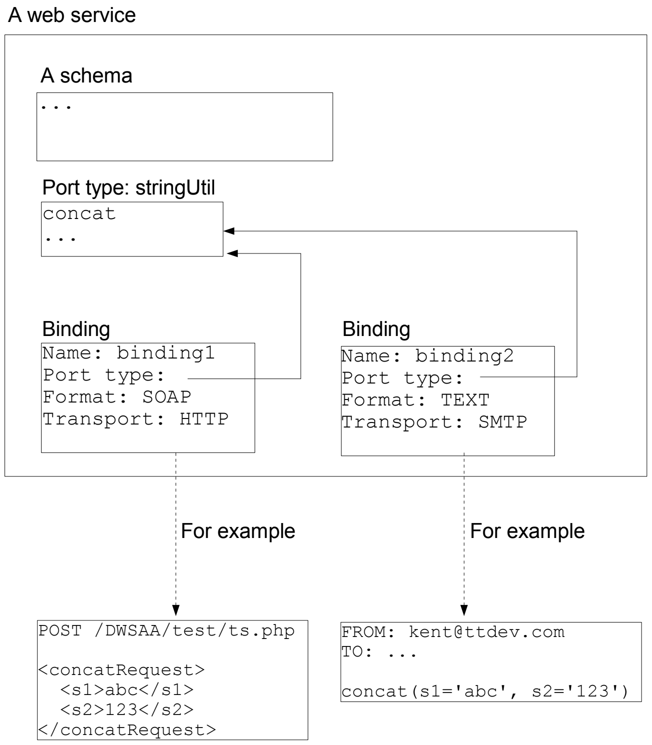


Binding

Actually, a port type may allow you to access it using different message formats. The message format that you have seen is called the "Simple Object Access Protocol (SOAP)" format. It is possible that, say, the stringUtil port type may also support a plain text format:

```
concat(s1='abc', s2='123')
```

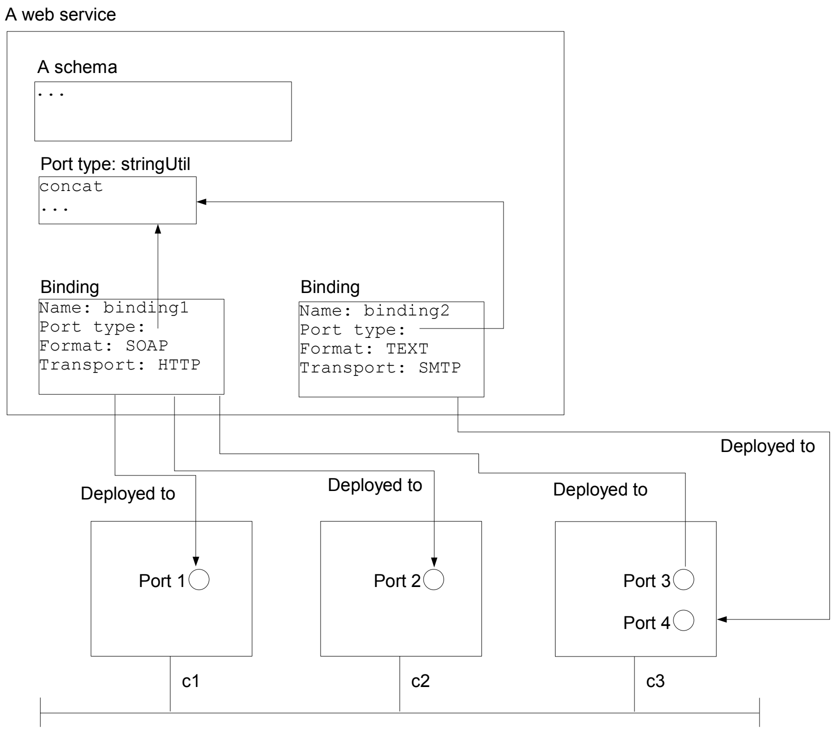
In addition to the message format, a port type may allow the message to be carried (transported) in an HTTP POST request or in an email. Each supported combination is called a "binding":



What bindings should your port type support? SOAP+HTTP is the most common combination. So, you should probably use this binding in practice.

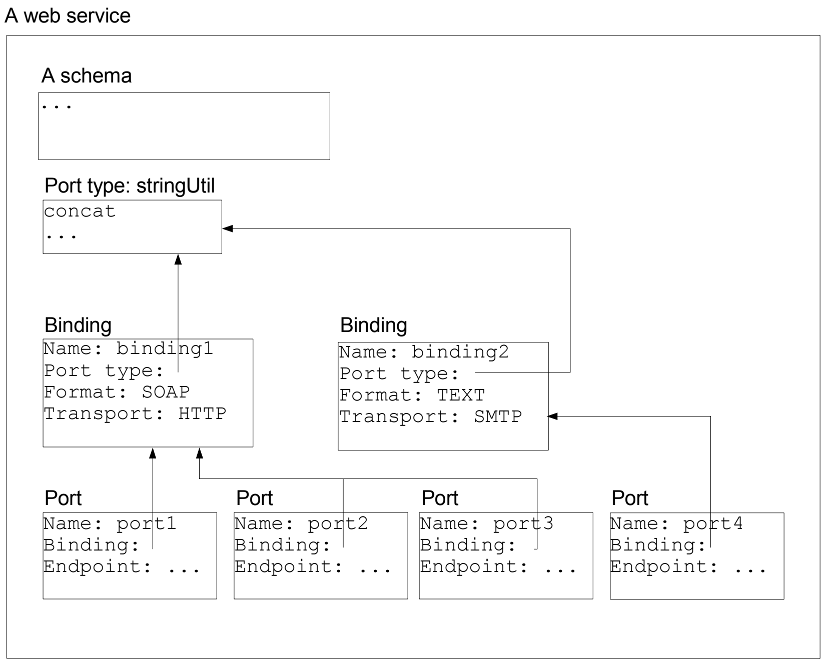
Port

Suppose that there are just too many people using your web service, you decide to make it available on more than one computers. For example (see the diagram below), you may deploy the above binding 1 on computers c1, c2 and c3 and deploy binding 2 on c3. In that case it is said that you have four ports. Three ports are using binding 1 and one using binding 2:



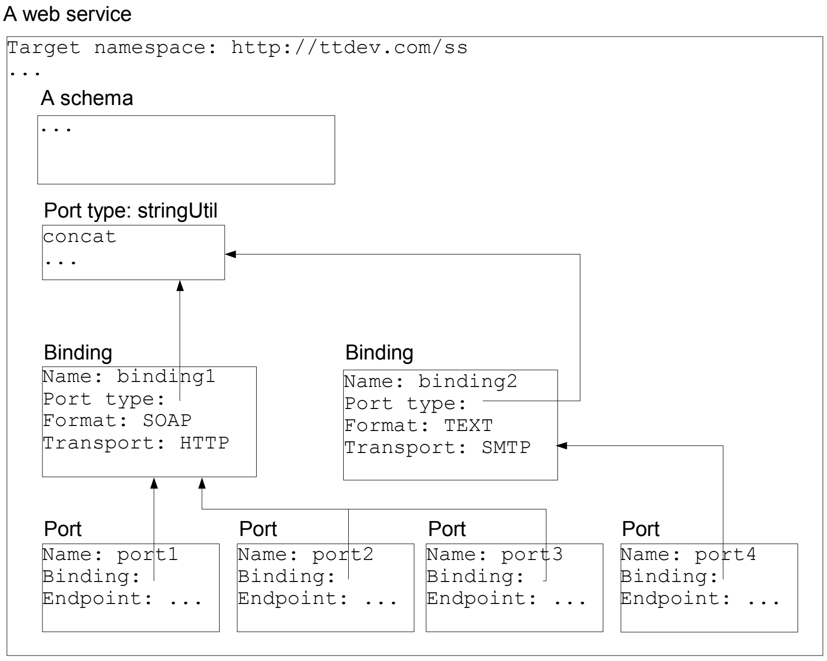
Note that it does NOT mean that the requests received by these three computers will be forwarded to a computer hiding behind for processing. Instead, it means that there is some software implementing the port type installed on these three computers. There is no requirement that the same piece of software is installed onto the different computers. For example, on c1, port 1 may be written in Java, while on c2, port 2 may be written in C#. The important point is that they both support the operations specified in port type stringUtil and the message format and transport specified in the binding 1. Port 4 must also implement the same operations too (same port type) but the message format and transport are different.

To tell others about this arrangement, you include these ports in the interface of the web service:



Target namespace

You have been using the same namespace for the operation names, port type names and etc. in this web service. Do they have to be in the same namespace? By default, this is the case: There is a single namespace for a web service to put the names into. This is called the "target namespace" for the web service:



You've been using http://ttdev.com/ss as the target namespace. Is it a good choice? Basically a namespace is good as long as it is globally unique. So this one should be good. However, people may try to download a web page from this URL. When it doesn't work, they may suspect that your web service is out of order. To avoid this confusion, you may use something called URN (Uniform Resource Name) as the namespace.

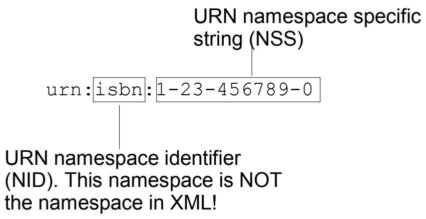
A namespace must be a URI. URI stands for Uniform Resource Identifier. There are two kinds of URI. One is URL such as http://www.foo.com/bar. The other is URN. A URN takes the format of urn:<some-object-type>:<some-object-id>. For example, International ISBN Agency has made a request to the IANA (International Assigned Numbers Association) that it would like to manage the object type named "isbn". After the request has been approved, the International ISBN Agency can declare that a URN urn:isbn:1-23-456789-0 will identify a book whose ISBN is 1-23-456789-0. It can determine the meaning of the object id without consulting IANA at all.

Similarly, you may submit a request to IANA to register your Internet domain name such as foo.com as the object type. Then on approval you can use URNs like urn:foo.com:xyz to identify an object xyz in your company. What xyz means or its format is completely up to you to decide. For example, you may use urn:foo.com:product:123 (so xyz is product:123) to mean the product #123 produced by your company, or urn:foo.com:patent/123 (so xyz is patent/123) to mean a patent coded 123 in your company.

However, this will create a lot of workload on you and on IANA (one registration per company!). As you have already registered the domain name foo.com, it is unlikely that someone will use it in their URN's. So, you may want to go ahead and use foo.com, or, as many people do, foo-com as the object type without registration with IANA and hope that there won't be any collision.

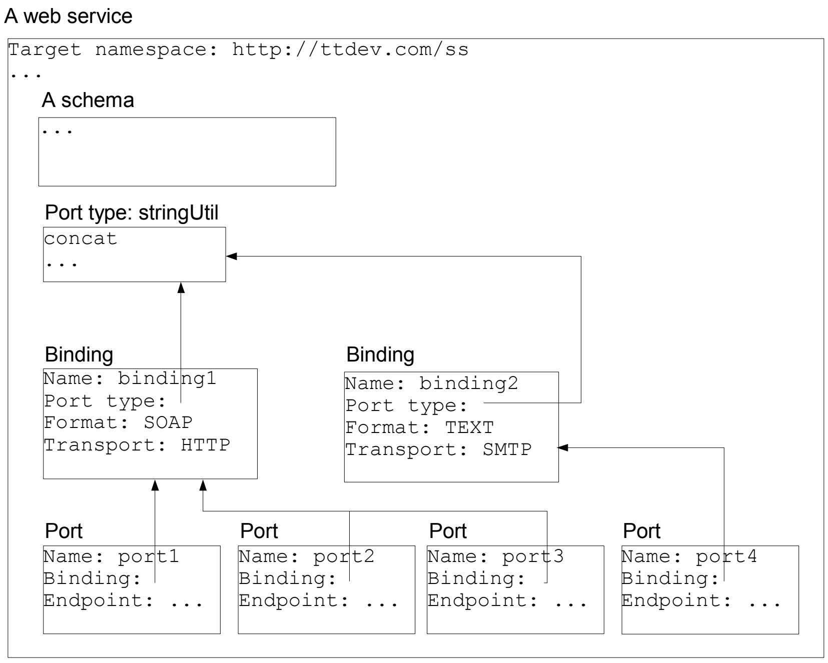
An XML namespace must be a URI. You can use a URL or a URN. Functionally there is no difference at all. For example, you may use say urn:ttdev.com:ss as the target namespace for your web service instead of http://ttdev.com/ss without changing any functionality.

By the way, if you are going to lookup references on URN, do NOT try to find terms like "object type" or "object id". The official terms are:


URN namespace specific string (NSS)
urn:isbn:1-23-456789-0
URN namespace identifier (NID). This namespace is NOT the namespace in XML!

WSDL

By now you have finished designing the interface for your web service:



It fully describes your web service. This description language (terms and concepts) is called "WSDL (Web Services Description Language)".

Summary

A web service is platform neutral, language neutral and can be accessed across the Internet.

A web service has one or more ports. Each port is a binding deployed at a certain network address (endpoint). A binding is a port type using a particular message format and a particular transport protocol. A port type contains one or more operations. An operation has an input message and an output message. Each message has one or more parts. Each part is either a certain element defined in the schema of the web service, or any element belonging to a certain element type in that schema. All this information is fully described in WSDL.

To call a RPC style web service, one will create an XML element with the name of the operation and a child element for each of its input message part. To call a document style web service, one will just send the one and only part of its input message. Because the XML element used to call a RPC style web service is not defined in any schema, for better interoperability, one should create document style web services.

The web service, and each of its ports, bindings, port types and operations, has a QName uniquely identifying it. A QName has a local part and an XML namespace. An XML namespace is a URI that is globally unique. By default the names of all these components are put into the target namespace of the web service.

There are two kinds of URI: URL and URN. URN takes the form of urn:<NID>:<NSS>. You can use either as an XML namespace. The only difference is that a URL is suggesting that it is the location of an object, while a URN is purely an id of the object.

Chapter 2

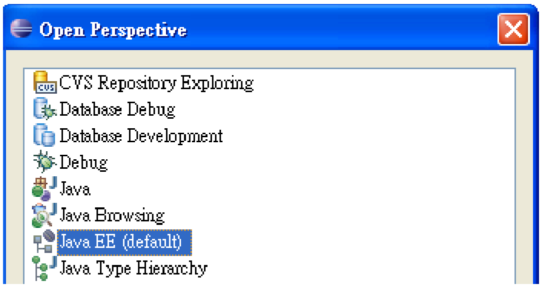
Implementing a web service

What's in this chapter?

In this chapter you'll learn how to implement the web service interface designed in the previous chapter.

Installing Eclipse

You need to make sure you have Eclipse v3.3 (or later) installed and it is the bundle for Java EE (the bundle for Java SE is NOT enough). If not, go to http://www.eclipse.org to download the Eclipse IDE for Java EE Developers (e.g., eclipse-jee-europa-fall-win32.zip). Unzip it into c:\eclipse. Then, create a shortcut to run "c:\eclipse\eclipse -data c:\workspace". This way, it will store your projects under the c:\workspace folder. To see if it's working, run it and make sure you can switch to the Java EE perspective:



BUG ALERT: If you're using Eclipse 3.3.1, there is a serious bug in it: When visually editing WSDL files Eclipse will frequently crash with an OutOfMemoryError. To fix it, modify c:\eclipse\eclipse.ini:

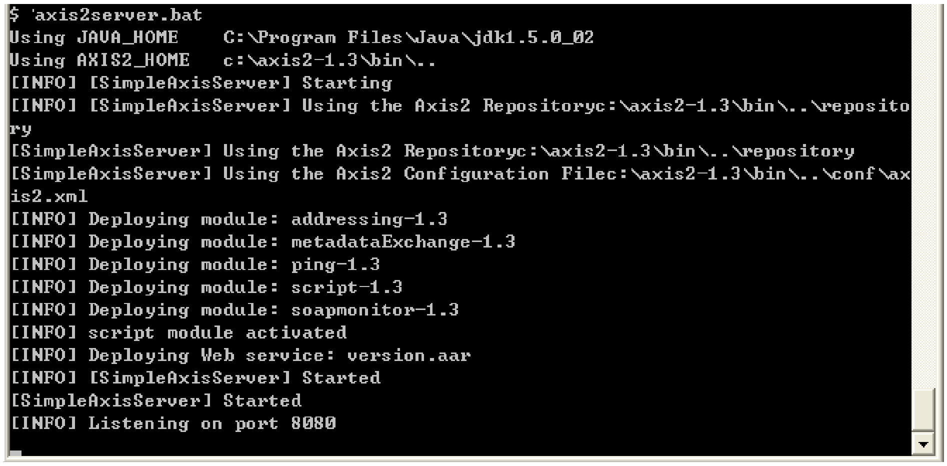
```
-showsplash
org.eclipse.platform
~~--launcher.XXMaxPermSize~~
~~256m~~
-vmargs
-Xms40m
-Xmx256m
-XX:MaxPermSize=256m
```

Delete them

This line must be put after -vmargs

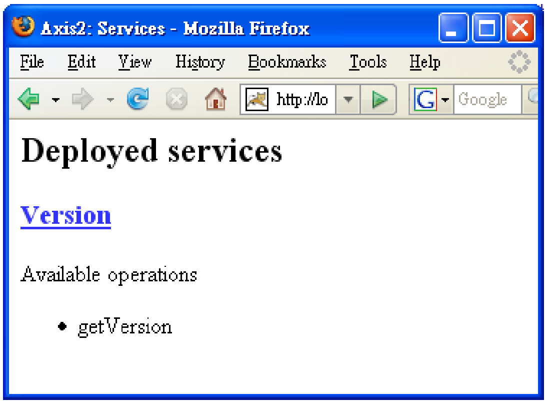
Installing Axis2

Next, go to http://ws.apache.org/axis2 to download the "Standard Binary Distribution" (e.g. axis2-1.3-bin.zip). Unzip it into c:\axis. To run the Axis server, change into c:\axis\bin and run axis2server.bat. You should see:



```
$ 'axis2server.bat
Using JAVA_HOME    C:\Program Files\Java\jdk1.5.0_02
Using AXIS2_HOME   c:\axis2-1.3\bin\..
[INFO] [SimpleAxisServer] Starting
[INFO] [SimpleAxisServer] Using the Axis2 Repositoryc:\axis2-1.3\bin\..\reposito
ry
[SimpleAxisServer] Using the Axis2 Repositoryc:\axis2-1.3\bin\..\repository
[SimpleAxisServer] Using the Axis2 Configuration Filec:\axis2-1.3\bin\..\conf\ax
is2.xml
[INFO] Deploying module: addressing-1.3
[INFO] Deploying module: metadataExchange-1.3
[INFO] Deploying module: ping-1.3
[INFO] Deploying module: script-1.3
[INFO] Deploying module: soapmonitor-1.3
[INFO] script module activated
[INFO] Deploying Web service: version.aar
[INFO] [SimpleAxisServer] Started
[SimpleAxisServer] Started
[INFO] Listening on port 8080
```

Then open a browser and access http://localhost:8080. You should see:


Axis2: Services - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://lo
Google

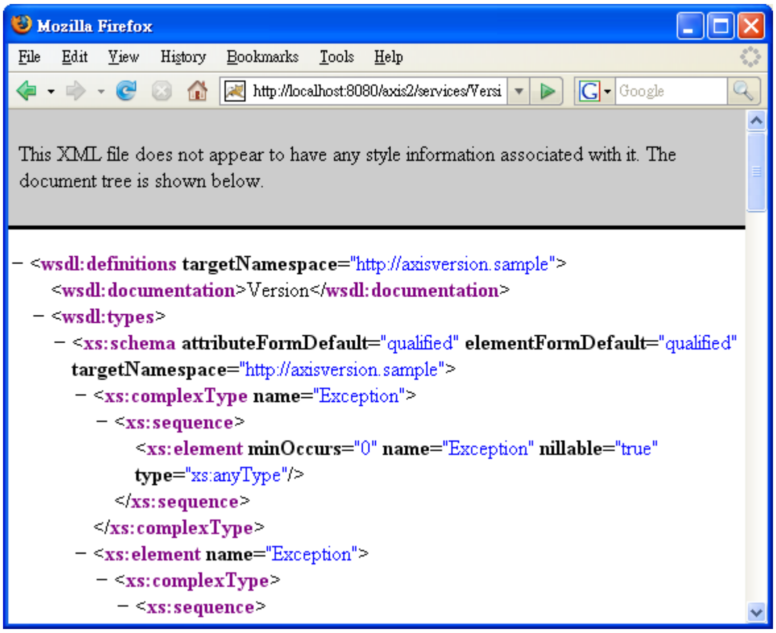
Deployed services

Version

Available operations

- getVersion

It means that there is an existing web service called "Version" available. Click on that "Version" link and you should see its WSDL file:

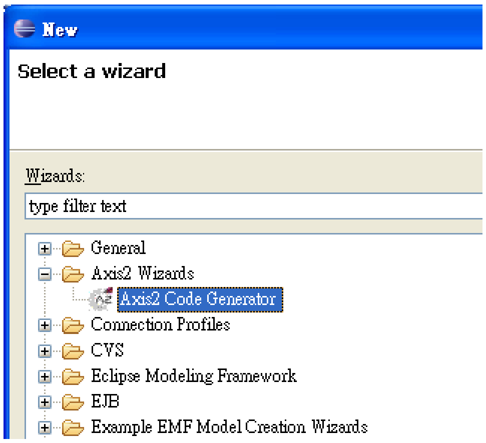

Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/axis2/services/Versi
Google

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <wsdl:definitions targetNamespace="http://axisversion.sample">
    <wsdl:documentation>Version</wsdl:documentation>
  - <wsdl:types>
    - <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
      targetNamespace="http://axisversion.sample">
      - <xs:complexType name="Exception">
        - <xs:sequence>
            <xs:element minOccurs="0" name="Exception" nillable="true"
            type="xs:anyType"/>
          </xs:sequence>
        </xs:complexType>
      - <xs:element name="Exception">
        - <xs:complexType>
          - <xs:sequence>
```

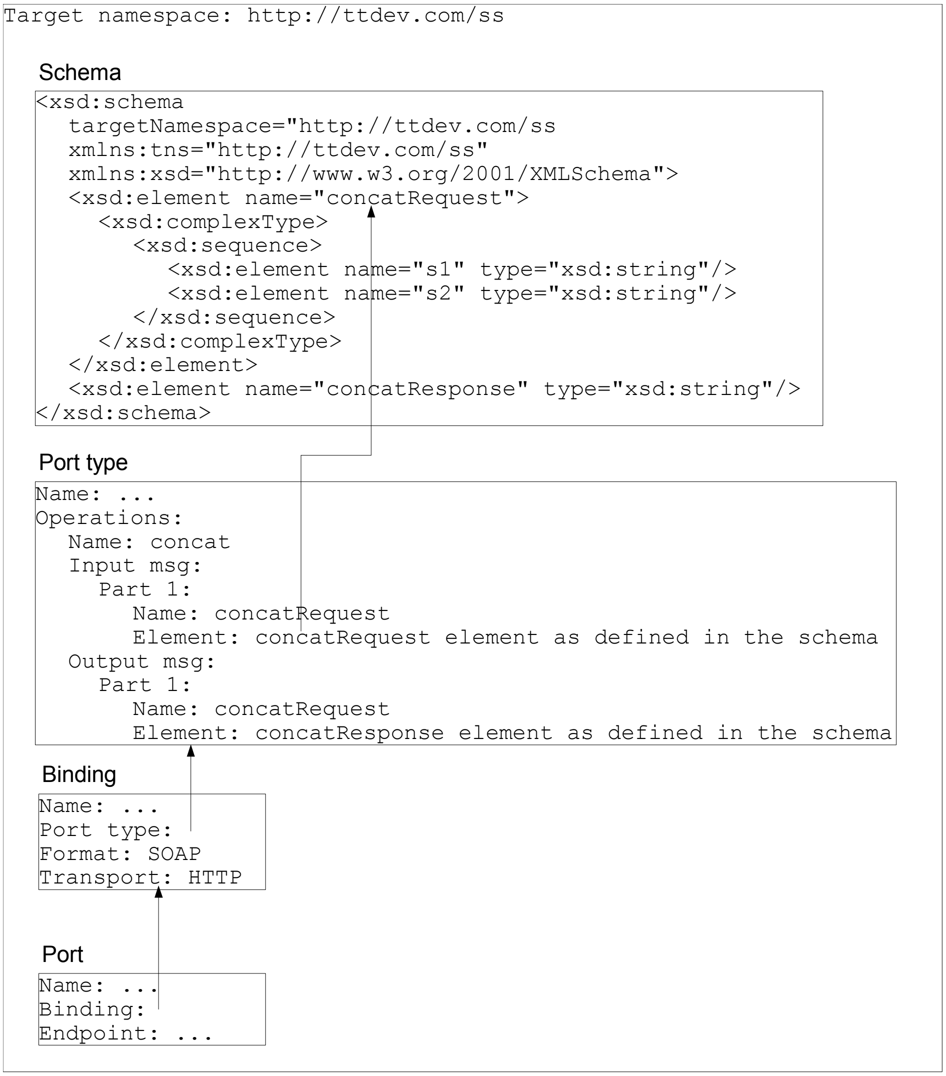
Installing the Axis2 plugin for Eclipse

Go to http://ws.apache.org/axis2/tools/index.html and download the Code Generator Wizard - Eclipse Plug-in. BUG ALERT: v1.4 of the plugin contains a critical bug. Use v1.3 instead! Suppose that it is axis2-eclipse-codegen-wizard.zip. Unzip it into the c:\eclipse\plugins folder. Restart Eclipse if required. To check if it's working, choose "File | New | Other" and you should see the "Axis2 Code Generator":



WSDL file for the web service

Suppose that you'd like to create a web service described in the previous chapter:



```
Target namespace: http://ttdev.com/ss
```

Schema

```
<xsd:schema
  targetNamespace="http://ttdev.com/ss
  xmlns:tns="http://ttdev.com/ss"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="concatRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="s1" type="xsd:string"/>
        <xsd:element name="s2" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="concatResponse" type="xsd:string"/>
</xsd:schema>
```

Port type

```
Name: ...
Operations:
  Name: concat
  Input msg:
    Part 1:
      Name: concatRequest
      Element: concatRequest element as defined in the schema
  Output msg:
    Part 1:
      Name: concatRequest
      Element: concatResponse element as defined in the schema
```

Binding

```
Name: ...
Port type:
Format: SOAP
Transport: HTTP
```

Port

```
Name: ...
Binding:
Endpoint: ...
```

To write it using the real WSDL language, it should be:

The names of the port types, operations, bindings and ports will be put into this namespace

All the elements and element types defined in the schema will be put into this namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://ttdev.com/ss"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="SimpleService"
  targetNamespace="http://ttdev.com/ss">
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://ttdev.com/ss"
      xmlns:tns="http://ttdev.com/ss">
      <xsd:element name="concatRequest">
        <xsd:complexType>
          <xsd:sequence>
          <xsd:element name="s1" type="xsd:string"/>
          <xsd:element name="s2" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="concatResponse" type="xsd:string"/>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="concatRequest">
    <wsdl:part name="concatRequest" element="tns:concatRequest" />
  </wsdl:message>
  <wsdl:message name="concatResponse">
    <wsdl:part name="concatResponse" element="tns:concatResponse" />
  </wsdl:message>
  <wsdl:portType name="SimpleService">
    <wsdl:operation name="concat">
      <wsdl:input message="tns:concatRequest" />
      <wsdl:output message="tns:concatResponse" />
    </wsdl:operation>
  </wsdl:portType>
  ...
</wsdl:definitions>
```

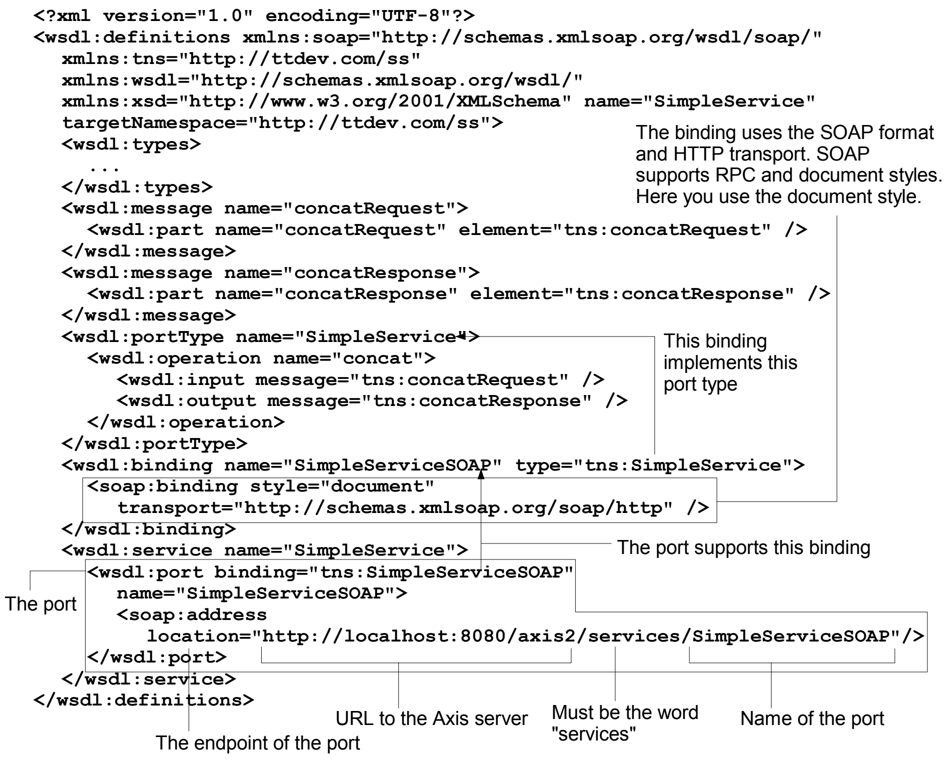
Put the schema into the <types> section

The input message contains a single part. The name of the part is unimportant.

The output message contains a single part. The name of the part is unimportant.

concat operation

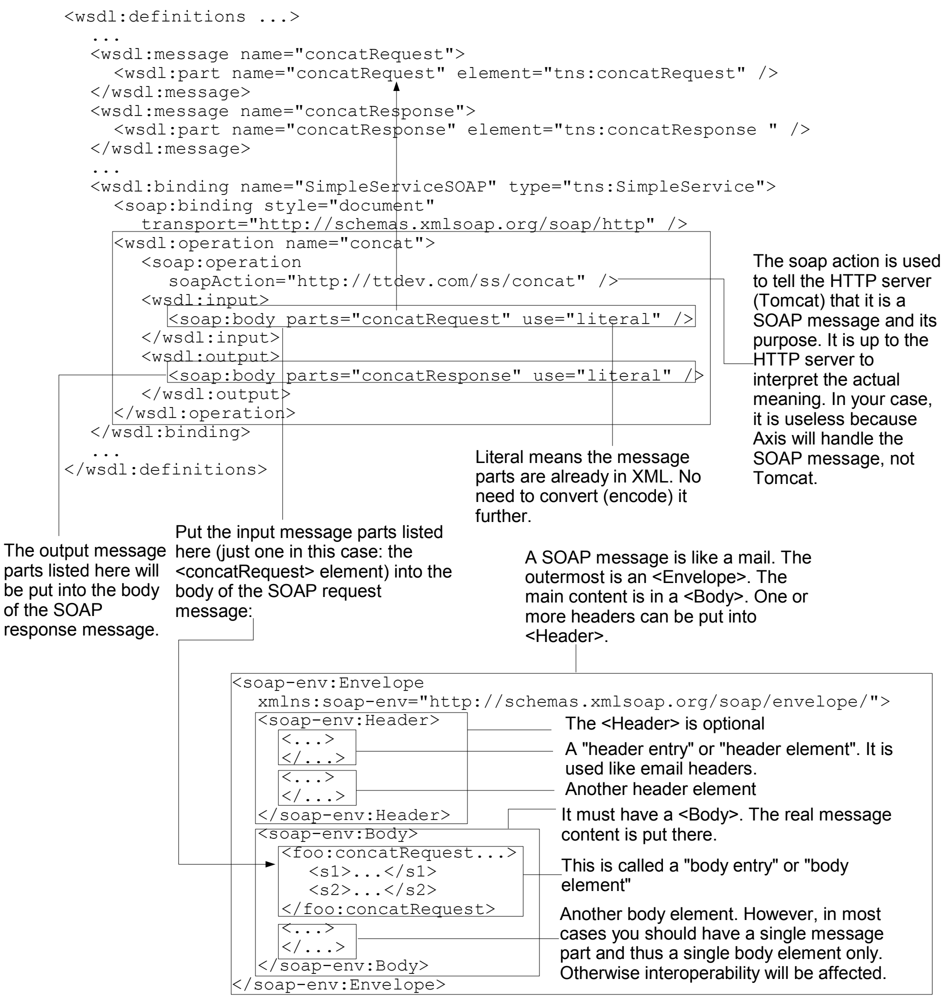
This defines the schema and the port type. To define the binding and the port:



```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://ttdev.com/ss"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="SimpleService"
  targetNamespace="http://ttdev.com/ss">
  <wsdl:types>
    ...
  </wsdl:types>
  <wsdl:message name="concatRequest">
    <wsdl:part name="concatRequest" element="tns:concatRequest" />
  </wsdl:message>
  <wsdl:message name="concatResponse">
    <wsdl:part name="concatResponse" element="tns:concatResponse" />
  </wsdl:message>
  <wsdl:portType name="SimpleService">
    <wsdl:operation name="concat">
      <wsdl:input message="tns:concatRequest" />
      <wsdl:output message="tns:concatResponse" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SimpleServiceSOAP" type="tns:SimpleService">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
  </wsdl:binding>
  <wsdl:service name="SimpleService">
    <wsdl:port binding="tns:SimpleServiceSOAP"
      name="SimpleServiceSOAP">
      <soap:address
        location="http://localhost:8080/axis2/services/SimpleServiceSOAP"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

The binding uses the SOAP format and HTTP transport. SOAP supports RPC and document styles. Here you use the document style.
This binding implements this port type
The port supports this binding
The port
URL to the Axis server
Must be the word "services"
Name of the port
The endpoint of the port

In fact, in a SOAP binding, you need to specify some more details:



```
<wsdl:definitions ...>
   ...
   <wsdl:message name="concatRequest">
     <wsdl:part name="concatRequest" element="tns:concatRequest" />
   </wsdl:message>
   <wsdl:message name="concatResponse">
     <wsdl:part name="concatResponse" element="tns:concatResponse " />
   </wsdl:message>
   ...
   <wsdl:binding name="SimpleServiceSOAP" type="tns:SimpleService">
     <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
     <wsdl:operation name="concat">
        <soap:operation
           soapAction="http://ttdev.com/ss/concat" />
        <wsdl:input>
           <soap:body parts="concatRequest" use="literal" />
        </wsdl:input>
        <wsdl:output>
           <soap:body parts="concatResponse" use="literal" />
        </wsdl:output>
     </wsdl:operation>
   </wsdl:binding>
   ...
</wsdl:definitions>
```

The soap action is used to tell the HTTP server (Tomcat) that it is a SOAP message and its purpose. It is up to the HTTP server to interpret the actual meaning. In your case, it is useless because Axis will handle the SOAP message, not Tomcat.

Literal means the message parts are already in XML. No need to convert (encode) it further.

The output message parts listed here will be put into the body of the SOAP response message.

Put the input message parts listed here (just one in this case: the <concatRequest> element) into the body of the SOAP request message:

A SOAP message is like a mail. The outermost is an <Envelope>. The main content is in a <Body>. One or more headers can be put into <Header>.

```
<soap-env:Envelope
   xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
   <soap-env:Header>
      <...>
      </...>
      <...>
      </...>
   </soap-env:Header>
   <soap-env:Body>
      <foo:concatRequest...>
         <s1>...</s1>
         <s2>...</s2>
      </foo:concatRequest>
      <...>
      </...>
   </soap-env:Body>
</soap-env:Envelope>
```

The <Header> is optional

A "header entry" or "header element". It is used like email headers.

Another header element

It must have a <Body>. The real message content is put there.

This is called a "body entry" or "body element"

Another body element. However, in most cases you should have a single message part and thus a single body element only. Otherwise interoperability will be affected.

RPC version of the web service

If the web service was a RPC style service, then the WSDL file would be like:

```
<wsdl:definitions ...>
  <wsdl:types>
    <xsd:schema ...>
      ~~<xsd:element name="concatRequest">~~
        ~~<xsd:complexType>~~
          ~~<xsd:sequence>~~
          ~~<xsd:element name="s1" type="xsd:string"/>~~
          ~~<xsd:element name="s2" type="xsd:string"/>~~
          ~~</xsd:sequence>~~
        ~~</xsd:complexType>~~
      ~~</xsd:element>~~
      ~~<xsd:element name="concatResponse" type="xsd:string"/>~~
```

Don't need these any more

```
    </xsd:schema>
  <wsdl:types/>
  <wsdl:message name="concatRequest">
    <wsdl:part name="s1" type="xsd:string" />
    <wsdl:part name="s2" type="xsd:string" />
```

The input message has two parts. Each part is of element type xsd:string (not elements).

```
  </wsdl:message>
  <wsdl:message name="concatResponse">
    <wsdl:part name="return" type="xsd:string" />
```

The output message has one part. It is of element type xsd:string (not elements).

```
  </wsdl:message>
  <wsdl:portType name="SimpleService">
    <wsdl:operation name="concat">
      <wsdl:input message="tns:concatRequest" />
      <wsdl:output message="tns:concatResponse" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SimpleServiceSOAP" type="tns:SimpleService">
    <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http" />
```

RPC style

```
    <wsdl:operation name="concat">
      <soap:operation
        soapAction="http://ttdev.com/ss/concat" />
      <wsdl:input>
        <soap:body parts="s1 s2" use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body parts="return" use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  ...
</wsdl:definitions>
```

Two message parts are listed. So, they will be included into the <Body> (but not directly). As it is a RPC style service, the caller must create an element with the QName of the operation and then add each message part listed here as a child element. So it should still have a single element in the <Body>:

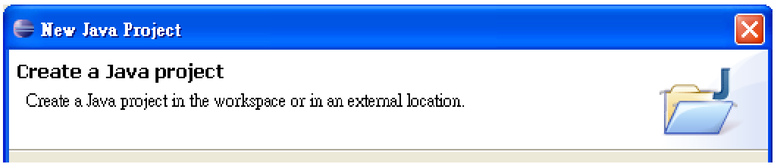
```
<soap-env:Envelope
  xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Header>
    ...
  </soap-env:Header>
  <soap-env:Body>
    <foo:concat ...>
      <s1>...</s1>
      <s2>...</s2>
    </foo:concat>
  </soap-env:Body>
</soap-env:Envelope>
```

No schema to validate it

As RPC style is not good for interoperability, you'll continue to use the document style version.

Creating the WSDL file visually

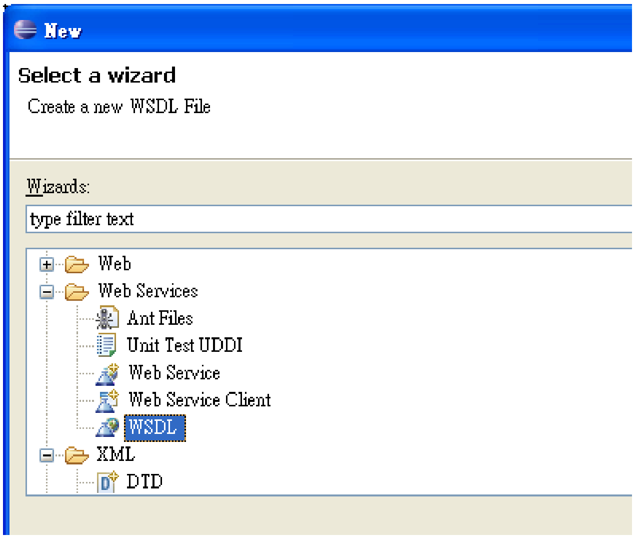
It may be error prone to manually create such a WSDL file. Instead, you may use the Eclipse to do it. First, create a new Java project named SimpleService in Eclipse:


New Java Project

Create a Java project

Create a Java project in the workspace or in an external location.

Make sure you use separate folders for sources and class files. Then go ahead and complete the creation of the project. Next, right click the project and choose "New | Other" and then "Web Services | WSDL":


New

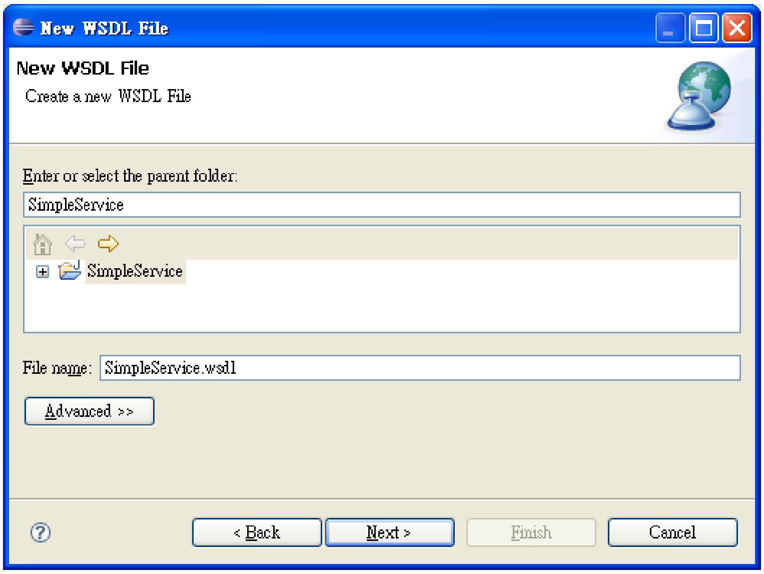
Select a wizard

Create a new WSDL File

Wizards:

type filter text

If you don't see this option, it means that you haven't installed the Java EE version of Eclipse. If it is working, click "Next" and enter SimpleService.wsdl as the filename:



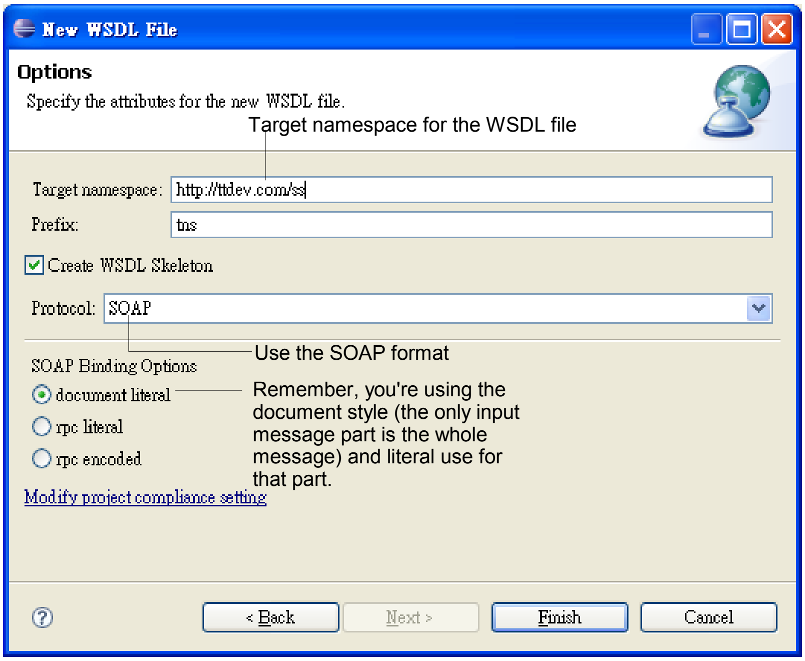
New WSDL File

Create a new WSDL File

Enter or select the parent folder: SimpleService

File name: SimpleService.wsdl

Click "Next". Then input as shown below:



Options

Specify the attributes for the new WSDL file.

Target namespace: http://ttdev.com/ss

Prefix: tns

[✓] Create WSDL Skeleton

Protocol: SOAP

SOAP Binding Options

[✓ document literal]

rpc literal

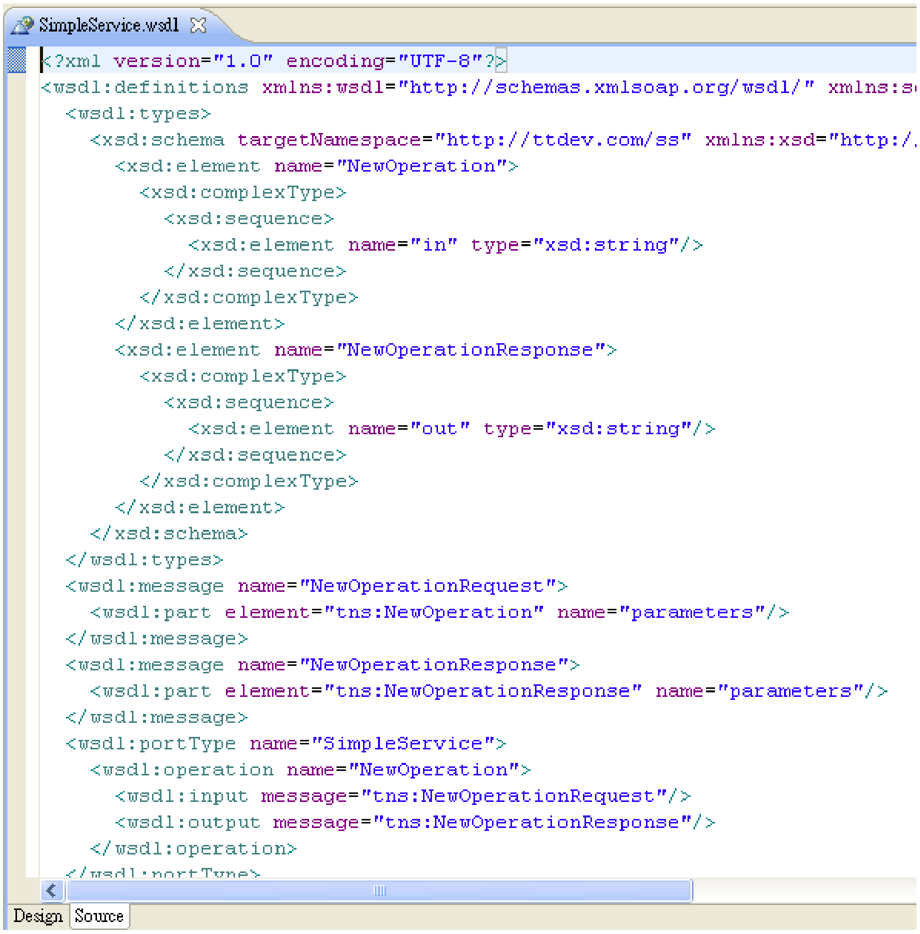
rpc encoded

Target namespace for the WSDL file

Use the SOAP format

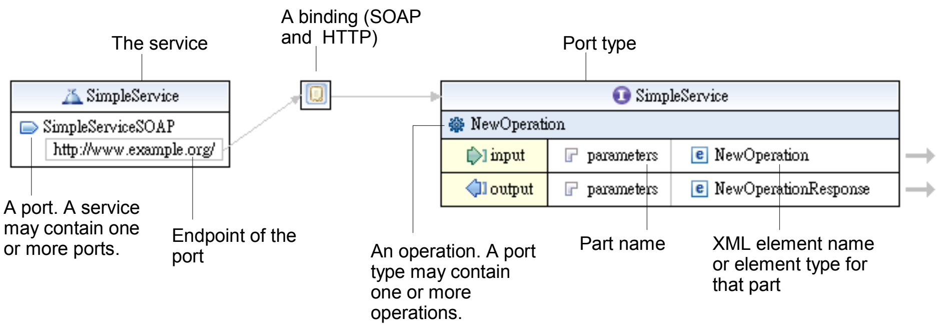
Remember, you're using the document style (the only input message part is the whole message) and literal use for that part.

Click "Finish". Then you will see something like:



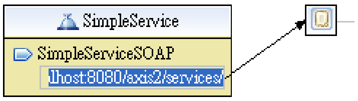
```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:s
  <wsdl:types>
    <xsd:schema targetNamespace="http://ttdev.com/ss" xmlns:xsd="http:/
      <xsd:element name="NewOperation">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="in" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="NewOperationResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="out" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="NewOperationRequest">
    <wsdl:part element="tns:NewOperation" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="NewOperationResponse">
    <wsdl:part element="tns:NewOperationResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="SimpleService">
    <wsdl:operation name="NewOperation">
      <wsdl:input message="tns:NewOperationRequest"/>
      <wsdl:output message="tns:NewOperationResponse"/>
    </wsdl:operation>
  </wsdl:portType>
```

This is the WSDL code. To edit it visually, click the "Design" tab at the bottom of the editor window. Then you'll see:

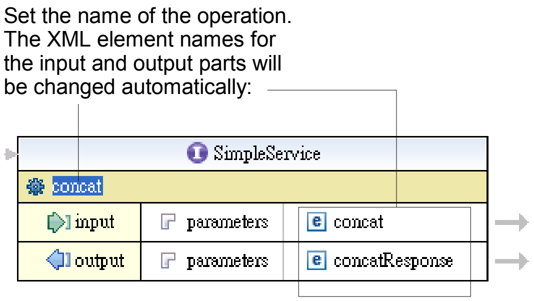


Double click on the endpoint to change it to http://localhost:8080/axis2/services/

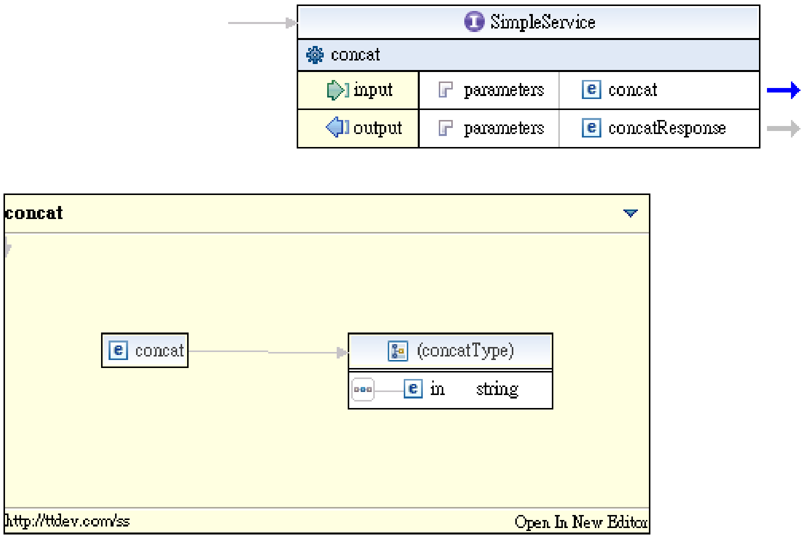
SimpleService:



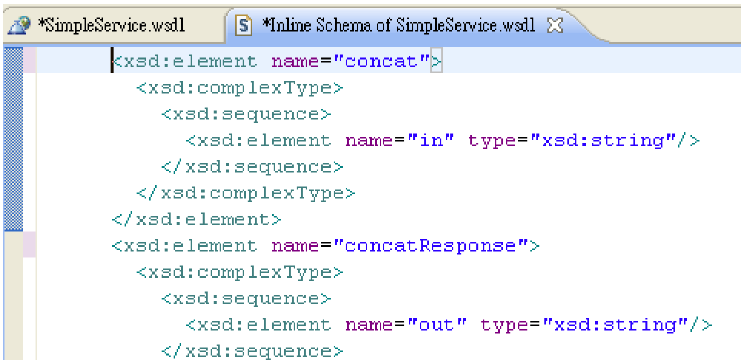
Double click on the name of operation and change it to "concat":


Set the name of the operation. The XML element names for the input and output parts will be changed automatically:
SimpleService
concat
input
parameters
concat
output
parameters
concatResponse

For the moment, the input part is an <concat> element. You'd like to change it to <concatRequest>. But for now, put the cursor on the arrow to its right first. The arrow will turn into blue color. Wait a couple of seconds then a preview window will appear showing the definition of the <concat> element:

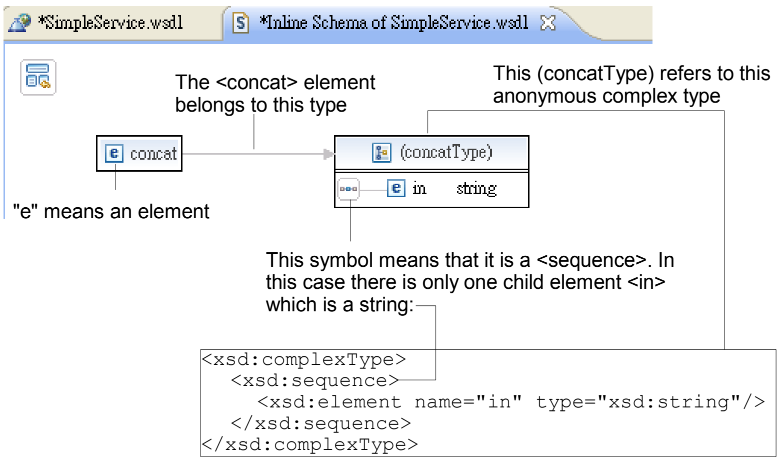


Clicking anywhere else will make that preview window disappear. To edit the schema definition, click on the blue arrow. A new editor window will appear:



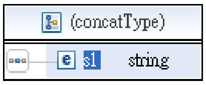
```
<xsd:element name="concat">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="in" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="concatResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="out" type="xsd:string"/>
    </xsd:sequence>
```

To edit it visually, click the "Design" tab at the bottom, you'll see:

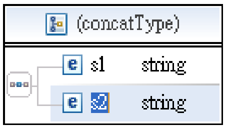

*SimpleService.wsdl
*Inline Schema of SimpleService.wsdl
The <concat> element belongs to this type
This (concatType) refers to this anonymous complex type
concat
(concatType)
in string
"e" means an element
This symbol means that it is a <sequence>. In this case there is only one child element <in> which is a string:

```
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name="in" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

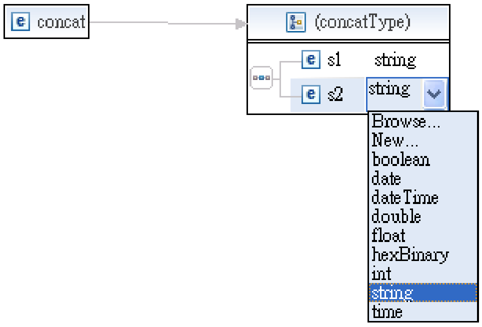
Double click on "in" and change it to "s1":



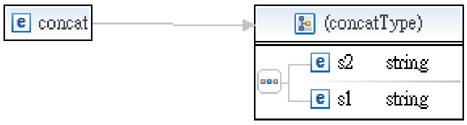
Right click it and choose "Add Element" and set the name to "s2":



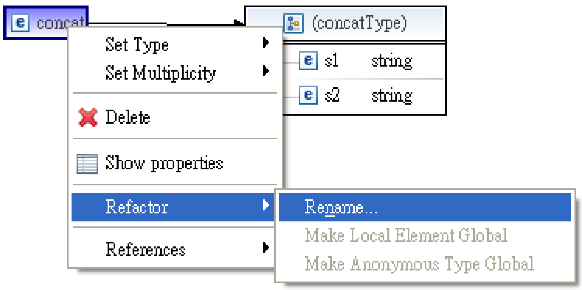
By default the type is already set to string. If you wanted it to be say an int instead, you would double click on the type and it would become a combo box and then you could choose "int":



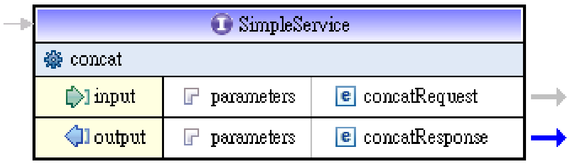
If you wanted s2 to appear before s1 in the sequence, you could drag it and drop it before s1:

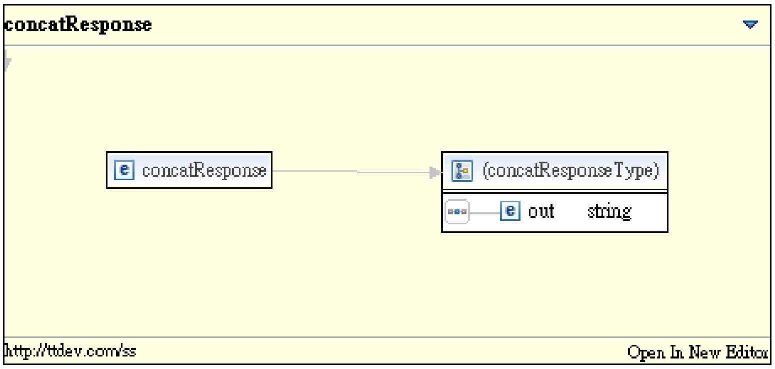


But for now, make sure it is s1 first and then s2. Next, right click on the <concat> element and choose "Refactor | Rename", then change its name to concatRequest:



You're done with the <concatRequest> element. Now return to the WSDL editor to work on the response message. For the moment, the <concatResponse> is like:





That is, it is an element that contains a sequence of <out> element:

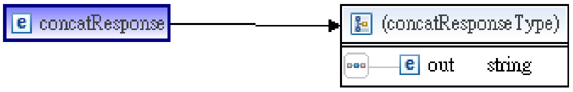
```
<foo:concatResponse>
   <foo:out>abc</foo:out>
</foo:concatResponse>
```

However, in your design, the response is simple type element, not a complex type element:

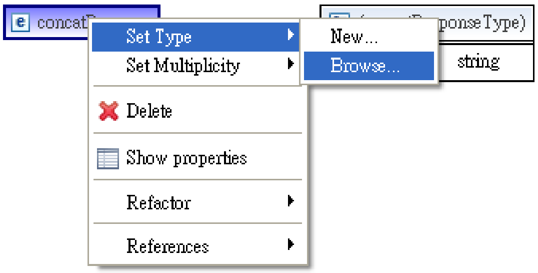
Its body contains a string instead of other elements

```
<foo:concatResponse
   xmlns:foo="http://ttdev.com/ss">abc123</foo:concatResponse>
```

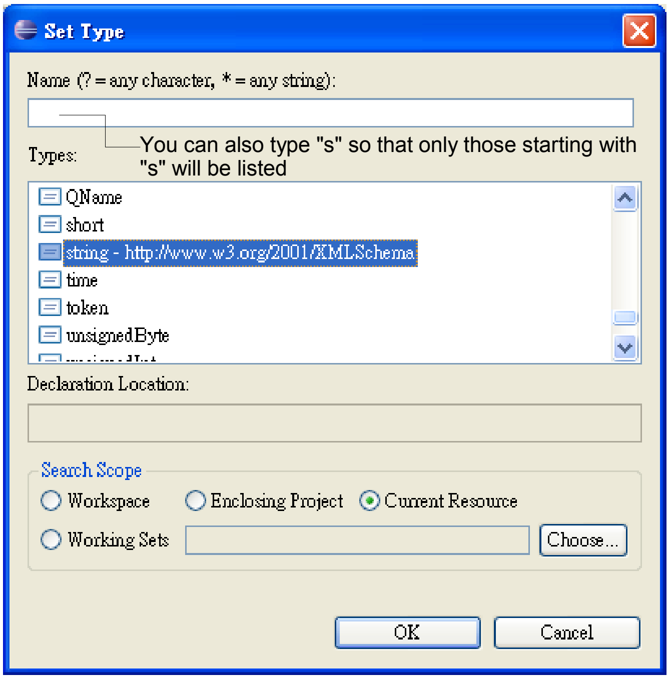
To do that, go into the schema editor to edit the <concatResponse> element:



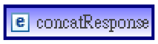
Right click it and choose "Set Type | Browse":



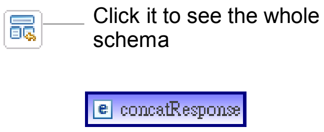
Choose "string":


Set Type
Name (? = any character, * = any string):
Types:
You can also type "s" so that only those starting with "s" will be listed
QName
short
string - http://www.w3.org/2001/XMLSchema
time
token
unsignedByte
Declaration Location:
Search Scope
Workspace
Enclosing Project
Current Resource
Working Sets
Choose...
OK
Cancel

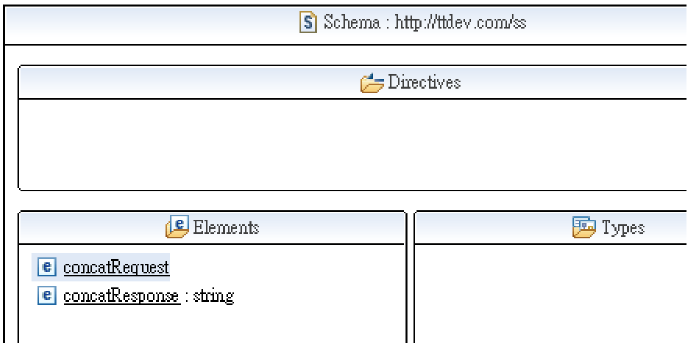
Then it will be like:



That's it. To review the whole schema, click on the icon at the upper left corner:



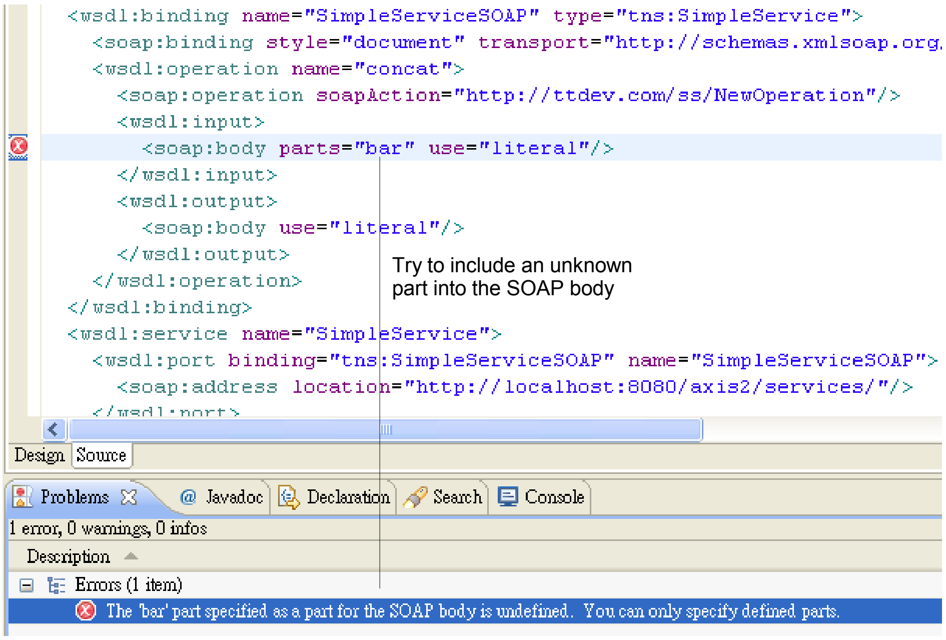
Then you'll see:



This looks fine. Now, save the file.

Validating the WSDL file

The next step is to validate the WSDL file to make sure it conforms to the various web services standards. To do that, right click the SimpleService.wsdl file in Eclipse and choose "Validate". If there were anything wrong, they would be reported in the Problems window. For example, here I had introduced an error into the file:



```
<wsdl:binding name="SimpleServiceSOAP" type="tns:SimpleService">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org,
  <wsdl:operation name="concat">
    <soap:operation soapAction="http://ttdev.com/ss/NewOperation"/>
    <wsdl:input>
      <soap:body parts="bar" use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="SimpleService">
  <wsdl:port binding="tns:SimpleServiceSOAP" name="SimpleServiceSOAP">
    <soap:address location="http://localhost:8080/axis2/services/"/>
```

1 error, 0 warnings, 0 infos

Errors (1 item)

The 'bar' part specified as a part for the SOAP body is undefined. You can only specify defined parts.

Generating a service stub

Next, in order to implement the web service, you will generate a "service stub" (see the diagram below). When a request message comes in, the service stub will convert the <concatRequest> XML element into a ConcatRequest Java object. Then it will pass it to the concat() method in a service skeleton to be supplied by you. Your concat() method will create and return a ConcatResponse Java object. The service stub will convert it into a <concatResponse> XML element and return it to the client: