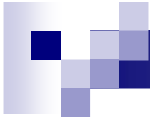
Web Services in J2EE 1.4

© Garth Gilmour 2008

Web Services In J2EE 1.4

- A J2EE 1.4 Web Service can be two things:
  - A Java class living in the Web Container
  - A Stateless Session Bean in the EJB Container
- Services are available at ‘ports’
  - As defined by the WSDL standard
- Every service consists of:
  - A ‘Service Endpoint Interface’
    - The Java interface which maps to the WSDL
    - The mapping is achieved using JAX-RPC
  - A ‘Service Implementation Bean’
    - The class or EJB which implements the interface

© Garth Gilmour 2008



Web Container

JDBC

JAX-RPC

Web Services Client

DB

JAX-RPC

EJB Container

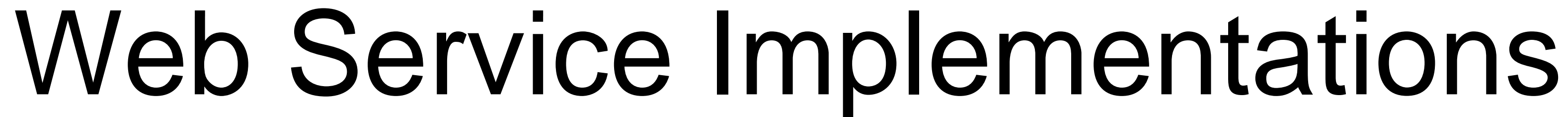
JDBC

© Garth Gilmour 2008

The Java API for XML-Based RPC

- JAX-RPC provides the RPC model of Web Services
  - It allows you call a Java method remotely by:
    - Marshalling method calls into SOAP messages
    - Mapping the WSDL/XML Schema data types into Java
    - Hence any type of client can make the call
  - The standard has several powerful features
    - Calls can be synchronous, asynchronous or one way
    - Handlers can pre-process requests and post-process responses
- Developing with JAX-RPC resembles standard RMI
  - In the .NET framework there is a single remoting standard which can operate transparently using binary or XML messaging

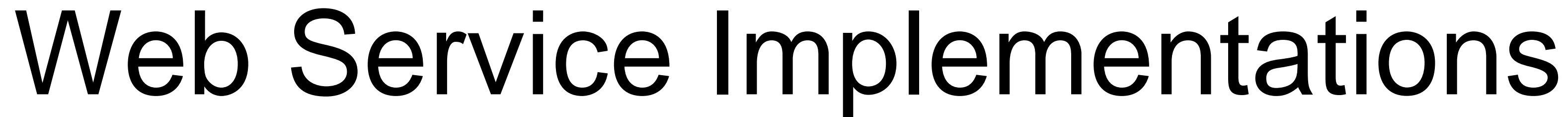
© Garth Gilmour 2008



Web Service Implementations

- If the service is in the EJB Container then:
  - It must be a Stateless Session bean
  - It may or may not implement the Service Endpoint Interface
    - But it must have methods that correspond to those in the interface
  - It is defined in 'ejb-jar.xml'
- Services in the Web Container are very similar
  - But they are called 'JAX-RPC Service Endpoints'
  - Because they follows rules defined in the 'JAX-RPC' spec
    - Which describe services living in the Servlet environment
    - This is <u>very</u> confusing as both implementations use 'JAX-RPC'
  - The class is defined in 'web.xml' using the 'servlet-class' element
    - Although it itself is not a Servlet

© Garth Gilmour 2008



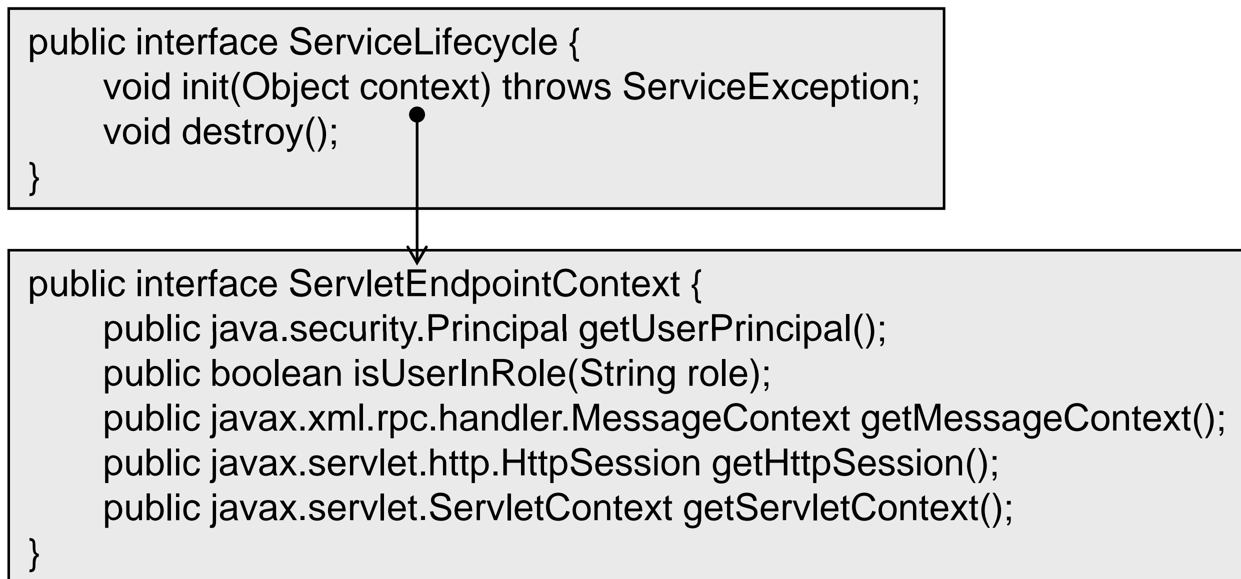
Web Service Implementations

- Rules for implementations in the Web Container are:
  - They will be accessed from multiple threads by default
    - If this cannot be supported it must implement ‘SingleThreadModel’
  - Implementing the ‘Service Endpoint Interface’ is optional
    - But it must have methods corresponding to those in the interface
    - If it does not implement the interface the methods must not be final
  - The implementation must be a stateless
    - As any instance may be used to service a request
  - They may implement the ‘ServiceLifeCycle’ interface
    - This contains ‘init’ and ‘destroy’ methods
    - These are used in the same way as Servlets

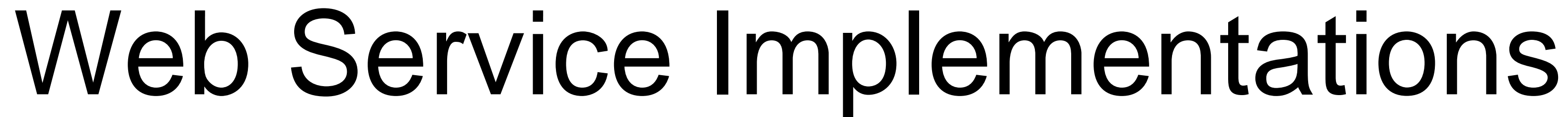
© Garth Gilmour 2008

Web Service Implementations

- When a ‘JAX-RPC’ class is deployed in a Web App the parameter to ‘init’ is a ‘ServletEndpointContext’ object
  - This can be used to access the Session and Context objects



© Garth Gilmour 2008



Web Service Implementations

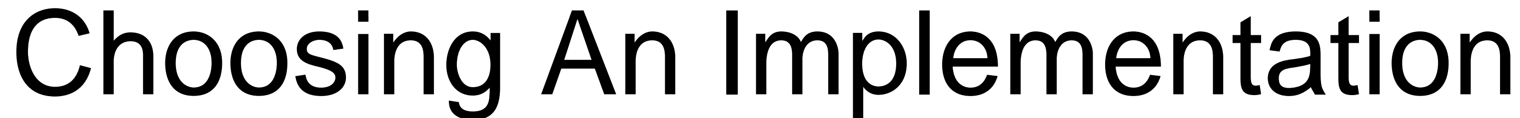
- The container must support ‘JAX-RPC’ handlers
  - Which can be used for pre and post processing
- A new deployment descriptor is added
  - This has the name ‘webservices.xml’
  - As usual its format is controlled by an XML Schema
  - Where the file is placed depends on the implementation
    - For EJB’s it is placed in the ‘META-INF’ folder or the JAR file
    - For Web Applications it is placed in the ‘WEB-INF’ folder
- WSDL docs may be packaged with the implementation
  - In ‘META-INF/wsdl’ for EJB’s and ‘WEB-INF/wsdl’ for Web Apps

© Garth Gilmour 2008

The ‘webservices.xml’ Config File

```
<webservices>
    <description>sample webservices.xml</description>
    <webservice-description>
        <webservice-description-name>ShopService</webservice-description-name>
        <wsdl-file>META-INF/wsdl/shop.wsdl</wsdl-file>
        <jaxrpc-mapping-file>META-INF/shopMappings.xml</jaxrpc-mapping-file>
        <port-component>
            <description>port description</description>
            <port-component-name>ShopPort</port-component-name>
            <wsdl-port xmlns:s=“http://shop”>s:ShopPort</wsdl-port>
            <service-endpoint-interface>sample.Shop</service-endpoint-interface>
            <service-impl-bean>
                <ejb-link>ShopBean</ejb-link>
            </service-impl-bean>
        </port-component>
    </webservice-description>
</webservices>
```

© Garth Gilmour 2008



Choosing An Implementation

- Endpoints will be located beside the business logic
  - If your existing application is entirely Web based you wouldn't extend it into the EJB Container just for Web Services
- EJB endpoints have several advantages
  - Multi threaded access is not a consideration
    - The container manages a pool of instances for you
  - Declarative transaction and access control is available
- Web endpoints have one big advantage
  - They can access the 'HttpSession'
    - Which can be used to store client state
  - They can still manage transactions
    - By using the JTA directly

© Garth Gilmour 2008

Building Web Services Part 5

Web Services in JEE 5

© Garth Gilmour 2008

Web Services in JEE 5

- JEE 5 makes J2EE 1.4 Web Services usable
  - J2EE 1.4 laid a framework for implementing and porting Web Services across different containers
  - But the actual work of writing a Web Service was too complex without vendor-provided tools
    - Such as those in WSAD/RAD and WebLogic Workshop
- The theme of JEE 5 is simplified development
  - EJB's can be created by annotating normal classes
  - The Java Persistence API can be used to load and save normal objects (again via the use of annotations)
  - Java Server Faces (JSF) becomes the official web framework
    - All containers are required to provide an implementation

© Garth Gilmour 2008

Standards for JEE Web Services

- The infrastructure for Web Services in JEE 5 remains the same as it was in J2EE 1.4
  - But existing standards have been revised and new standards written to make implementing them easier
- The new and revised standards are:
  - JSR 109: Web Services for Java EE
  - JSR 181: Web Services Metadata for the Java Platform
  - JSR 224: The Java API for XML Web Services
    - Better known by the abbreviation JAX-WS
  - JSR 222: The Java Architecture for XML Binding
    - Better known by the abbreviation JAXB

© Garth Gilmour 2008

Annotations Declared in JSR 181

Annotation	Description
@WebService	Marks a class as being a Web Service implementation
@WebMethod	Marks a method as implementing a Web Service operation
@Oneway	Indicates that a web method has no output
@WebParam	Customizes the mapping of parameters into WSDL
@WebResult	Customizes the mapping of return values into WSDL
@SOAPBinding	Customizes the type of SOAP messaging used
@HandlerChain	Associates a Web Service with an externally defined handler chain
@SoapMessageHandlers @SoapMessageHandler	Defines a set of handlers to pre/post process a request

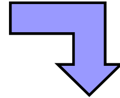
© Garth Gilmour 2008

JSR 181 Web Service Annotation

- '@WebService' is used to mark a class as a service
  - In JEE5 this can be a normal class (POJO) or a Session EJB
  - As defined in J2EE 1.4 each type lives in a separate container
- Web methods can be specified via an interface
  - The 'endpointInterface' attribute may hold the full type name of an interface which declares the methods of the Web Service
  - If this is used then the class must not be decorated with any other annotations, except those used to define handlers
- Otherwise web methods are discovered in two ways
  - By default all public methods become web methods
  - If one or more methods is marked with '@WebMethod' then only these methods are exposed as web methods

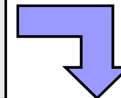
© Garth Gilmour 2008

```
@WebService
public class MathsWS {
}
```



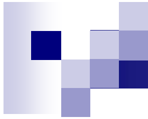
```
<s0:portType name="MathsWS">
	<!-- Operations defined here… -->
</s0:portType>
<s0:service name="MathsWSService">
	<s0:port binding="s1:MathsWSServiceSoapBinding" name="MathsWSSoapPort">
		<s2:address location="http://localhost:7001/TestOne/MathsWS" />
	</s0:port>
</s0:service>
```

```
@WebService(name="Calculator",serviceName="IntCalc")
public class MathsWS {
}
```



```
<s0:portType name="Calculator">
	<!-- Operations defined here… -->
</s0:portType>
<s0:service name="IntCalc">
	<s0:port binding="s1:IntCalcSoapBinding" name="CalculatorSoapPort">
		<s2:address location="http://localhost:7001/TestOne/MathsWS" />
	</s0:port>
</s0:service>
```

© Garth Gilmour 2008



```
@WebService
public class MathsWS {

    @WebMethod
    public int add(int no1, int no2) {
        return no1 + no2;
    }

    @WebMethod
    public int multiply(int no1, int no2) {
        return no1 * no2;
    }
    @WebMethod
    public int subtract(int no1, int no2) {
        return no1 - no2;
    }
    @WebMethod
    public int divide(int no1, int no2) {
        return no1 / no2;
    }
}
```

```
<env:Envelope xmlns:env="…">
<Header xmlns="…" />
   <env:Body>
      <add xmlns="http://demos">
         <no1>12</no1>
         <no2>24</no2>
      </add>
   </env:Body>
</env:Envelope>
```

```
<env:Envelope xmlns:env="…">
   <env:Header />
   <env:Body>
      <m:addResponse
               xmlns:m="http://demos">
         <m:return>36</m:return>
      </m:addResponse>
   </env:Body>
</env:Envelope>
```

© Garth Gilmour 2008

JSR 181 Web Service Annotation

- The other attributes of ‘@WebService’ are used to customize how the WSDL is generated
  - ‘name’ is used to set the value of the WSDL ‘portType’
    - This defaults to the class name without a package prefix
  - ‘serviceName’ is used to set the value of the WSDL ‘service’
    - This defaults to the class name plus ‘Service’
  - ‘targetNamespace’ defines the URL used by generated types
- An existing WSDL file can be specified via ‘wsdlLocation’
  - The type definitions, encoding styles etc must be consistent with the symbol names and annotation information in the class

© Garth Gilmour 2008

```java
@WebService(name="Calculator",
          serviceName="IntCalc",
          targetNamespace="calculator/V1")
public class MathsWS {

    @WebMethod
    public int add(int no1, int no2) {
          return no1 + no2;
    }

    @WebMethod
    public int multiply(int no1, int no2) {
          return no1 * no2;
    }
    @WebMethod
    public int subtract(int no1, int no2) {
          return no1 - no2;
    }
    @WebMethod
    public int divide(int no1, int no2) {
          return no1 / no2;
    }
}
```

```xml
<env:Envelope xmlns:env="…">
<Header xmlns="…" />
   <env:Body>
      <add xmlns="http://calculator/V1 ">
         <no1>12</no1>
         <no2>24</no2>
      </add>
   </env:Body>
</env:Envelope>
```

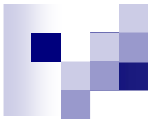
```xml
<env:Envelope xmlns:env="…">
   <env:Header />
   <env:Body>
      <m:addResponse
            xmlns:m="http://calculator/V1 ">
         <m:return>36</m:return>
      </m:addResponse>
   </env:Body>
</env:Envelope>
```

© Garth Gilmour 2008

JSR 181 Web Method Annotation

- '@WebMethod' indicates a method is a web method
  - It is exposed in the WSDL as an operation within a port type
  - As previously noted all public methods become web methods unless one or more is marked with this annotation
- Two attributes can be specified
  - 'operationName' is the name used within the generated WSDL
  - 'action' is the value used by the 'SOAPAction' HTTP header
  - By default the operation name is the same as the method name and the 'SOAPAction' header is left blank
- The method may be further annotated with '@Oneway'
  - This means the web method has no output message
  - Hence the return type of the method must be void

© Garth Gilmour 2008



```
@WebService
public class MathsWS {
    @WebMethod(operationName="addInts")
    public int add(int no1, int no2) {
        return no1 + no2;
    }
    @WebMethod(operationName="multiplyInts")
    public int multiply(int no1, int no2) {
        return no1 * no2;
    }
    @WebMethod(operationName="subtractInts")
    public int subtract(int no1, int no2) {
        return no1 - no2;
    }
    @WebMethod(operationName="divideInts")
    public int divide(int no1, int no2) {
        return no1 / no2;
    }
}
```

```
<env:Envelope xmlns:env="…">
  <Header xmlns="…" />
  <env:Body>
    <addInts xmlns="http://demos">
      <no1>12</no1>
      <no2>24</no2>
    </addInts>
  </env:Body>
</env:Envelope>
```

```
<env:Envelope xmlns:env="…">
  <env:Header />
  <env:Body>
    <m:addIntsResponse
            xmlns:m="http://demos">
      <m:return>36</m:return>
    </m:addIntsResponse>
  </env:Body>
</env:Envelope>
```

© Garth Gilmour 2008

Further Specifying Web Methods

- '@WebParam' can be used to control how parameters are passed from the message into the web method
  - The parameter name and namespace URL can be changed
  - A parameter can be marked as IN, OUT or INOUT
    - OUT / INOUT mean the parameter is a wrapper object
    - The enclosed type can be changed and passed back to the client
  - The 'header' attribute indicates the parameter value is to be taken from the specified message header
    - By default all parameters come from the message body
- '@WebResult' controls the return value
  - Once again the name and namespace URL can be changed

© Garth Gilmour 2008

```
@WebService(name="Calculator",
                    serviceName="IntCalc",
                    targetNamespace="http://calculator/V1")
public class MathsWS {

      @WebMethod
      @WebResult(name="ResultOfAddition")
      public int add(@WebParam(name="firstNumber") int no1,
                            @WebParam(name="secondNumber") int no2) {
            return no1 + no2;
      }
      //Other methods as before…
}
```

```
<env:Envelope xmlns:env="…">
   <Header xmlns="…" />
   <env:Body>
      <add xmlns="http://calculator/V1">
         <firstNumber>12</firstNumber>
         <secondNumber>24</secondNumber>
      </add>
   </env:Body>
</env:Envelope>
```

```
<env:Envelope xmlns:env="…">
   <env:Header />
   <env:Body>
      <m:addResponse xmlns:m="http://calculator/V1">
         <m:ResultOfAddition>36</m:ResultOfAddition>
      </m:addResponse>
   </env:Body>
</env:Envelope>
```

© Garth Gilmour 2008

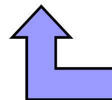
Complex Types in Web Methods

- When using the Document Wrapped encoding JAX-WS requires that two JavaBeans be generated
  - The Request Bean holds the parameter information
    - By default it is named after the method
  - The Response Bean hold the return value
    - By default it is the method name plus “Response”
  - Both beans are placed in a sub package called ‘jaxws’
    - For the method ‘add’ in ‘demo.Maths’ the beans ‘demos.jaxws.Add’ and ‘demos.jaxws.AddResponse’ would be created
- The beans XML mappings are defined via JAXB
  - As are the mappings for all the information that is inserted

© Garth Gilmour 2008

```java
@XmlRootElement(name = "quantityInStock", namespace = "http://five.ws.demos/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "quantityInStock", namespace = "http://five.ws.demos/")
public class QuantityInStock {
    @XmlElement(name = "arg0", namespace = "")
    private Item arg0;
    public Item getArg0() {
        return this.arg0;
    }
    public void setArg0(Item arg0) {
        this.arg0 = arg0;
    }
}
```

```java
public class Item {
    public Item() {
        super();
    }
    public String getProductCode() {
        return productCode;
    }
    public void setProductCode(String productCode) {
        this.productCode = productCode;
    }
    public String getSupplierCode() {
        return supplierCode;
    }
    public void setSupplierCode(String supplierCode) {
        this.supplierCode = supplierCode;
    }
    private String supplierCode;
    private String productCode;
}
```



```java
@WebService
public class ComplexShop {
    @WebMethod
    public StockCheck quantityInStock(Item item){
        return new StockCheck(20, 10);
    }
}
```

© Garth Gilmour 2008

The SOAP Encoding and Handlers

- ‘@SOAPBinding’ specifies the encoding style set in the WSDL file and used by all operations in the port type
  - ‘style’ can be DOCUMENT or RPC
  - ‘use’ can be ENCODED or LITERAL
  - ‘parameterStyle’ can be BARE or WRAPPED
- Annotations can also be used to specify handlers
  - Logical handlers operate on message contents only
    - ‘@HandlerChain’ points to a file containing handler definitions
  - Protocol handlers operate on protocol specific headers
    - ‘@SOAPMessageHandlers’ and ‘@SOAPMessageHandler’ point to handler classes for pre and post processing SOAP headers

© Garth Gilmour 2008

```
@WebService(serviceName="MathsDocWrapped")
@SOAPBinding(style=DOCUMENT,
             use=LITERAL,
             parameterStyle=WRAPPED)
public class Maths {
    @WebMethod
    public double add(int no1, int no2) {
        return no1 + no2;
    }
    //other methods omitted
}
```

```
@WebService(serviceName="MathsDocBare")
@SOAPBinding(style=DOCUMENT,
             use=LITERAL,
             parameterStyle=BARE)
public class Maths {
    @WebMethod
    public double add(int num) {
        return num + 5;
    }
    //other methods omitted
}
```

```
@WebService(serviceName="MathsRPC")
@SOAPBinding(style=RPC,
             use=LITERAL,
             parameterStyle=WRAPPED)
public class Maths {
    @WebMethod
    public double add(int no1, int no2) {
        return no1 + no2;
    }
    //other methods omitted
}
```

© Garth Gilmour 2008

Implementing JEE5 Web Services

- JEE5 servers support JAX-WS based Web Services
  - These will be deployed via IDE’s or ANT tasks
    - WebLogic Workshop and RAD are example IDE’s
- The JAX-WS 2.1 RI works on top of the server
  - The service runs within a Web Application that generates the WSDL and maps SOAP messages to method calls
- The steps for developing a service are:
  - Write a class decorated with JSR 181 annotations
  - Use the supplied tools to generate helper classes
  - Package the classes with the RI libraries in a Web App
  - Configure the RI specific Servlet and listener in ‘web.xml’
  - Configure the details of the endpoint in a ‘sun-jaxws.xml’ file

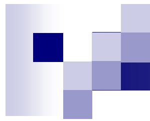
© Garth Gilmour 2008

ANT Tasks Within the JAX-WS RI

- The JAX-WS RI contains tools for creating services and clients
  - These can be run from the command line or as Ant tasks
- Everything generated by the tools is portable
  - All the helper classes can be used in any JEE5 container
  - By default the WSDL is generated at runtime

JAX-WS RI Tool	Description
wsimport	Generates Web Service artefacts based on a WSDL file - normally used to generate Web Service clients
wsgen	Generates server side Web Service artefacts based on a class decorated with JSR 181 annotations
apt	Generates Web Service artefacts by processing annotations in Java source code via a JSR 175 annotation processor

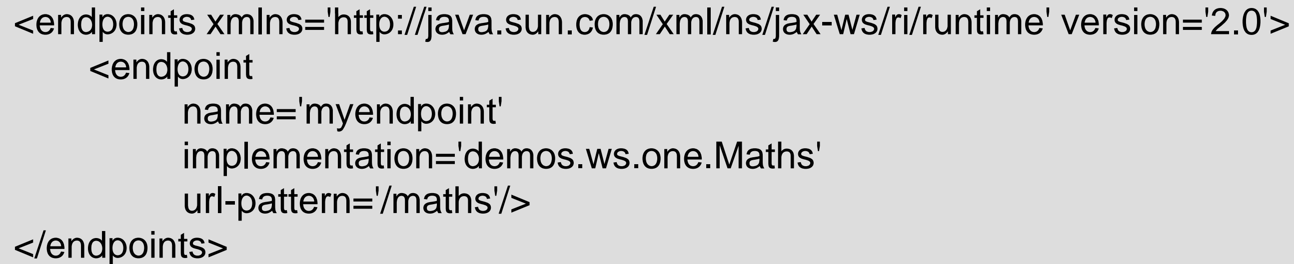
© Garth Gilmour 2008



```
<target name="buildWebService" depends="compilePartOne">
      <taskdef name="wsgen" classname="com.sun.tools.ws.ant.WsGen">
            <classpath refid="myClasspath"/>
      </taskdef>
      <wsgen sei="${ws.impl}" destdir="${build.bin}"
                  sourcedestdir="${build.generated.src}" keep="true"
                  verbose="true" protocol="soap1.1">
            <classpath refid="myClasspath"/>
      </wsgen>
</target>
```

```
<target name="createProxy">
      <taskdef name="wsimport" classname="com.sun.tools.ws.ant.WsImport">
            <classpath refid="myClasspath"/>
      </taskdef>
      <wsimport verbose="true"
                  extension="true"
                  wsdl="${proxy.wsdl.url}"
                  sourcedestdir="${build.generated.src}"
                  destdir="${build.bin}">
      </wsimport>
</target>
```

© Garth Gilmour 2008



```
<endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime' version='2.0'>
    <endpoint
        name='myendpoint'
        implementation='demos.ws.one.Maths'
        url-pattern='/maths'/>
</endpoints>
```

```
<web-app … >
  <listener>
    <listener-class>com.sun.xml.ws.transport.http.servlet.WSServletContextListener</listener-class>
  </listener>
  <servlet>
    <servlet-name>JaxWsRiServlet</servlet-name>
    <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>JaxWsRiServlet</servlet-name>
    <url-pattern>/maths</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

© Garth Gilmour 2008