Web Services

Let us write More Robust Web Services through the command line
Part IV

Working With a Production Web Service

- `.jws` files are quick ways to get our Java classes as Web Services, but they're not always the best choice.

- For one thing, **we need the source code** – if we want to expose a pre-existing class on our system without source, or do not want to expose the service code, problem!

- Also, the amount of configuration we can do as to how the service gets accessed is pretty limited - we can't specify custom type mappings, or control which gets invoked when people are using our service.

Deploying via descriptors

- To really use the flexibility available to us in Axis, we should get familiar with the Axis **Web Service Deployment Descriptor (WSDD)** format.
- A deployment descriptor contains a bunch of things we want to "deploy" into Axis - i.e. make available to the Axis engine.
- The most common thing to deploy is a Web Service, so let's start by taking a look at a deployment descriptor for a basic service.

FtoC & CtoF with Deployment Descriptor

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"

    xmlns:java="http://xml.apache.org/axis/wsdd/providers/ja
    va">

     <service name="ConvService" provider="java:RPC">
       <parameter name="className" value="Converter"/>
       <parameter name="allowedMethods" value="*"/>
     </service>
</deployment>
```

defines the "java" namespace

tells the engine that any public method on that class may be called via SOAP

Deploy with Deployment Descriptor

- To deploy the service

```
% java org.apache.axis.client.AdminClient
  deploy.wsdd
  <Admin>Done processing</Admin>
```

- To get the list of all deployed component

```
% java org.apache.axis.client.AdminClient
  list
<big XML document returned here>
```

Un-Deploy services

- To un-deploy the service (file name undeploy.wsdd)

```
<undeployment xmlns="http://xml.apache.org/axis/wsdd/">
   <service name="ConvService"/>
</undeployment>
```

```
% java org.apache.axis.client.AdminClient
  undeploy.wsdd
<Admin>Done processing</Admin>
```

Java2WSDL: Building WSDL from Java

- Step 1: Provide a Java interface or class
- Step 2: Create **`WSDL`** using **`Java2WSDL`**
- Step 3: Create Bindings using **`WSDL2Java`**
- Step 4: Deploy service
- Step 5: Write client to access it

Java2WSDL: Building WSDL from Java

- *Java2WSDL*: Generate the WSDL file for the given Converter interface.
- *WSDL2Java*: Generate the server side wrapper code, and stubs for easy client access.
- *Converter**SoapBindingImpl*: Fill in wrapper to call the existing Converter code.
- *Deploy*: Deploy the service to Apache Axis.
- *Client*: Write a client that uses the generated stubs, to easily access the Web service.

Generate the Converter interface

```
public interface Converter {
        public double FtoC(double x);
        public double CtoF(double y);
}
```

Compile it normally

Generate the WSDL file for the given Converter interface

```
%java org.apache.axis.wsdl.Java2WSDL -o Con.wsdl
-l "http://localhost:8080/axis/services/Converter"
-n "urn:WebSer" -p"samples.WebSer" "urn: WebSer"
samples.WebSer.Converter
```

Where:

- -o indicates the name of the *output WSDL* file
- -l indicates the *location of the service*
- -n is the target *namespace* of the WSDL file
- -p indicates a mapping from the *package to a namespace.*
- the class specified contains the interface of the web service.

WSDL2Java: Generate the Server-side Wrapper Code and Stubs For Easy Client Access

```
% java org.apache.axis.wsdl.WSDL2Java -o . -d
  Session -s -S true -Nurn:WebSer samples.WebSer
  Con.wsdl
```

- Base output directory (.)
- Scope of deployment (Application, Request, or Session)
- Turn on server-side generation (we wouldn't do this if we were accessing an external Web service, as we would then just need the client stub)
- Package to place code (samples.WebSer)
- Name of WSDL file (Con.wsdl)

WSDL2Java: Generate the Server-side Wrapper Code and Stubs For Easy Client Access

This will generate the following files:

- ***ConverterSoapBindingImpl.java:*** **Java file containing the default server implementation of the Converter web service. You will need to modify this file to add your implementation to call the existing Converter service.**
- ***Converter.java:*** **New interface file that contains the appropriate java.rmi.Remote usages.**
- ***ConverterService.java:*** **Java file containing the client side service interface.**
- ***ConverterServiceLocator.java:*** **Java file containing the client side service implementation class.**
- ***ConverterSoapBindingSkeleton.java:*** **Server side skeleton.**
- ***ConverterSoapBindingStub.java:*** **Client side stub.**
- ***deploy.wsdd:*** **Deployment descriptor**
- ***undeploy.wsdd:*** **Undeployment descriptor**

WSDL2Java: Generate the Server-side Wrapper Code and Stubs For Easy Client Access

- Now you have all of the necessary files to build your client/server side code and deploy the web service!
- Compile all the .java files
- **Deploy the Web Service using the WSDD Deployment Descriptor:**

```
% java org.apache.axis.client.AdminClient
  deploy.wsdd
<admin>Done processing</admin>
```

- Now our Converter Web service is alive and running in the server!

Generate the WSDL file for the given Converter interface

```
public interface Converter {
  public double FtoC(double x)
  {
    return (x-32.0) * 0.56;
  }

  public double CtoF(double y)
  {
    return (1.8 * y + 32.0);
  }
}
```

Don’t you have any other simpler way to do all this?

Our Steps for easy deployment

- **Step 1: Write a service java code into your directory (`c:\axis-1_3\sapmles\WebSer\`) and compile it**
- **Step 2: Copy class file it into `C:\Tomcat4.1\webapps\axis\WEB-INF\classes\`**
- **Step 3: Deploy service using deployConv.wsdd file**
- **Step 4: Create client code automatically using following command at your directory**

```
java org.apache.axis.wsdl.WSDL2Java
http://localhost:8080/axis/services/ConvService?wsdl
```

- **Step 5: Write client at local directory and compile it to access services**

Step 1

- Step 1: Write a service java code into your directory (**`c:\axis-1_3\sapmles\WebSer\`**) and compile it

Step 1

```
public class Conv{
   public double FtoC(double x)
   {
      return (5.0/9.0)*(x-32.0);
   }

   public double CtoF(double y)
   {
      return ((9.0/5.0)*y)+32.0;
   }
}
```

Step 2 & 3

- Step 2: Copy generated class file it into **`C:\Tomcat4.1\webapps\axis\WEB-INF\classes\`**
- Step 3: Deploy service using **`deployConv.wsdd`** file

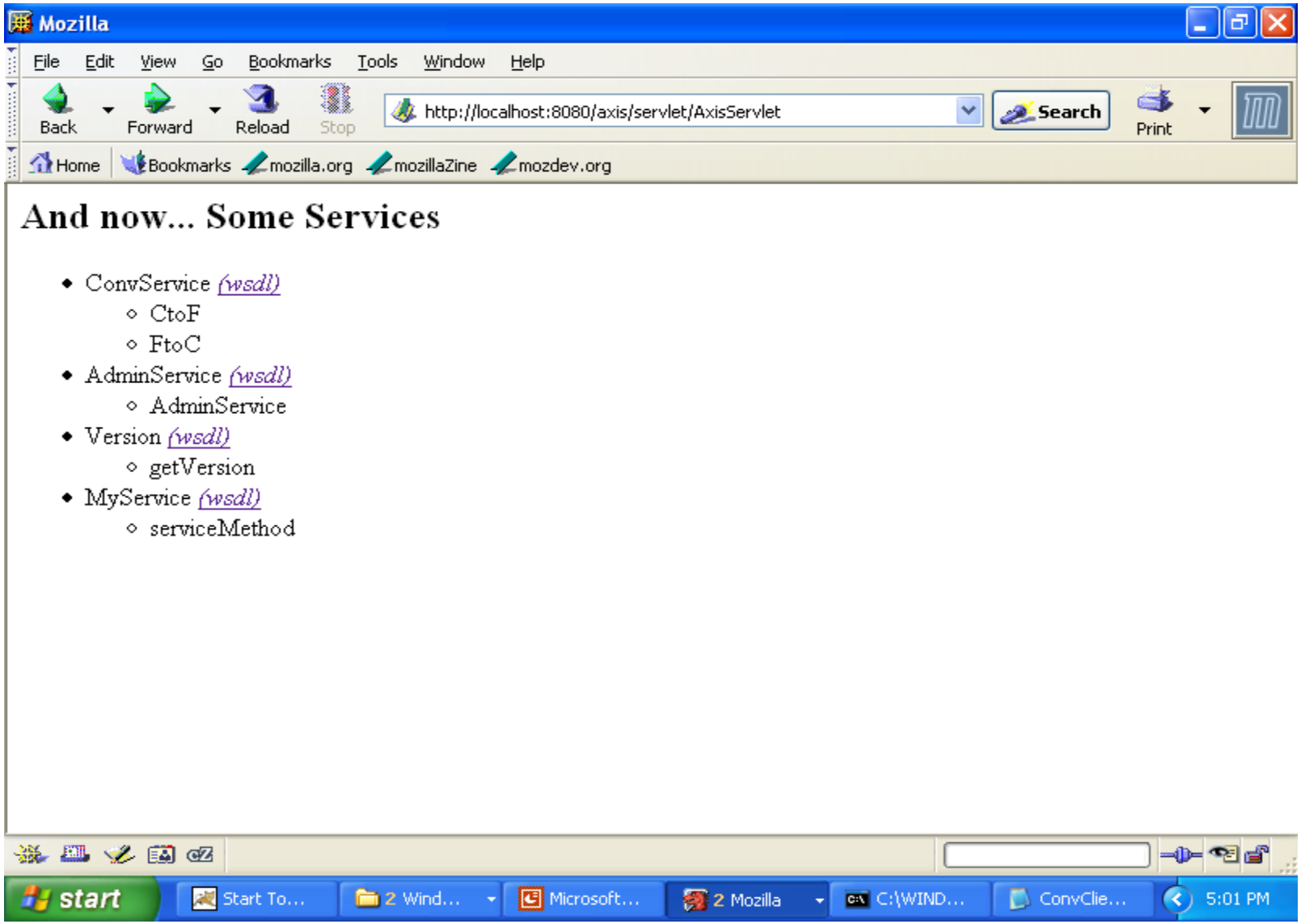
Step 3
(deployConv.wsdd and undeployConv.wsdd)

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/provider
   s/java">
 <service name="ConvService" provider="java:RPC">
  <parameter name="className" value="Conv"/>
  <parameter name="allowedMethods" value="*"/>
 </service>
</deployment>
```

Step 3 (if needed) undeployConv.wsdd

```
<undeployment
  xmlns="http://xml.apache.org/axis/wsdd/">
 <service name="ConvService"/>
</undeployment>
```

Step 3 Screenshot after the deployment



Step 4

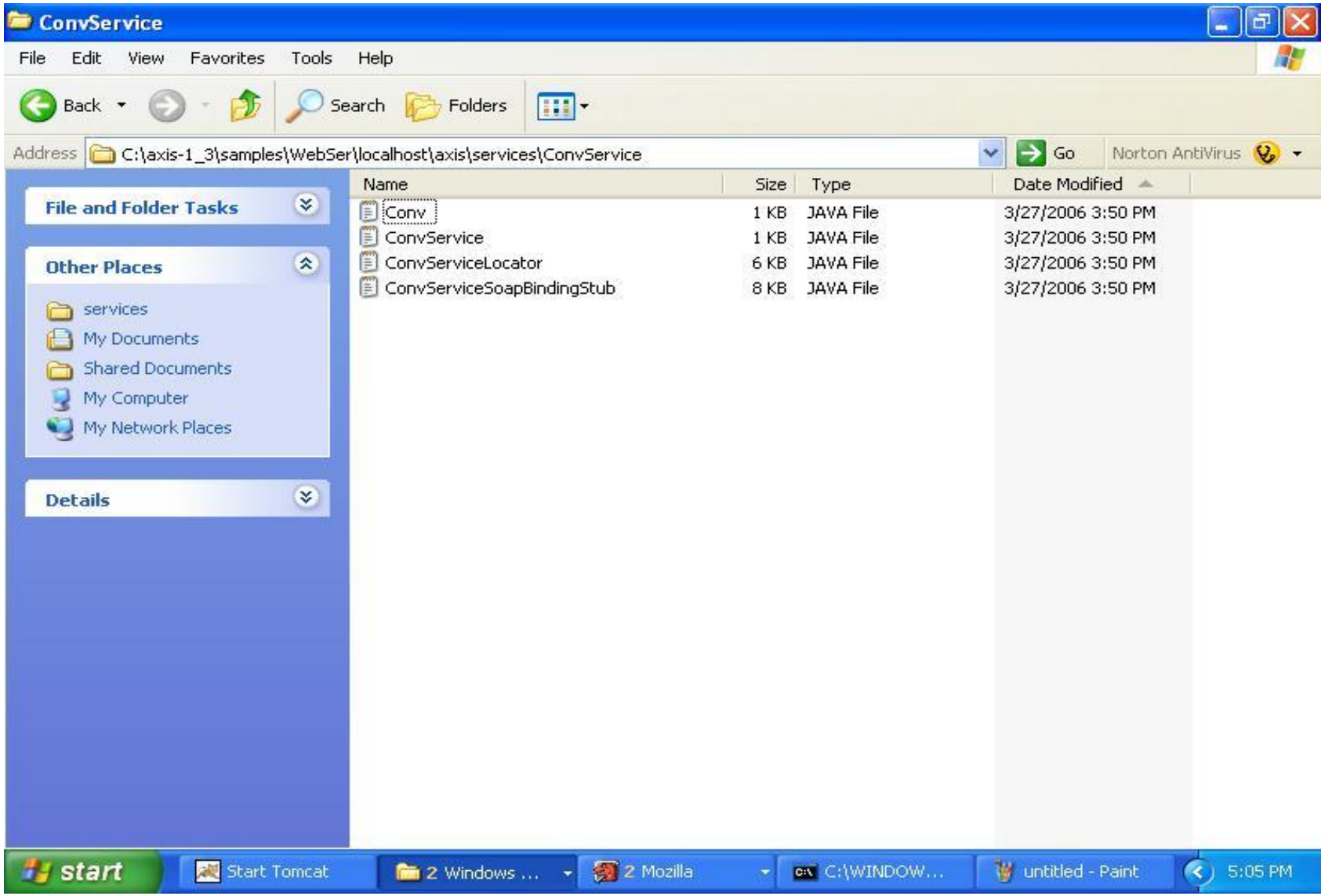
- Step 4: Create client stub code automatically using following command at your directory

```
java org.apache.axis.wsdl.WSDL2Java
http://localhost:8080/axis/services/
ConvService?wsdl
```

Step 4

- This will generate a bunch of classes with the package hierarchy according to the URL of the WSDL. It generates the interface file.
- **Caution!** The method names in the interface may change, so you need to change it in the client file.

Step 4 Screenshot after the deployment



Step 5

- Step 5: Write client at local directory to access it

Step 5

```
import localhost.axis.services.ConvService.*;
import org.apache.axis.AxisFault;

public class ConvClient
 {
     public static void main(String[] args)
     {
         try
         {
          ConvService service = new ConvServiceLocator();
          localhost.axis.services.ConvService.Conv port = service.getConvService();
          double result =  port.ctoF(-40.00);
          System.out.println("Convertion result is " + result);
         }
         catch (AxisFault af)
         {
             System.err.println("An Axis Fault occurred: " + af);
         }
         catch (Exception e)
         {
             System.err.println("Exception caught: " + e);
         }
     }
 }
```

Alternative Step 5

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;

public class FibClient {
public static void main(String [] args) {
try
{
    String endpoint = "http://localhost:8080/axis/services/FibService";
    Service service = new Service();
    Call call = (Call) service.createCall();
    call.setTargetEndpointAddress(new java.net.URL(endpoint));
    call.setOperationName(new QName("http:///", "fibNo"));
        Integer i = new Integer(15);
    Integer ret = (Integer) call.invoke(new Object[]{i});
    System.out.println("Fin Number is " + ret);
       } catch (Exception e) {
           System.err.println(e.toString());
          }
      }
}
```

Step 5

- Compile and run the java client- you get the answer.

Step 5 Screenshot after the Compilation