# Creating Web Services with MATLAB® Builder™ JA

## Background

MATLAB® Builder™ JA is an extension to MATLAB® Compiler™ and is useful for wrapping MATLAB® functions into one or more Java™ classes that make up a Java™ component, or package. These classes can then be used in a Java™ application.

The example included here is a simple temperature conversion module that takes user-supplied temperature values specified in either degrees Fahrenheit or degrees Celsius and converts them to the other scale. The example demonstrates how to use the MATLAB® Builder™ JA API to handle temperatures in various different formats (scalars, matrices, cell arrays, and structs).

## Prerequisites

The components were built using MATLAB® Version 7.3 (R2006b) and MATLAB® Builder™ JA Version 1.0. MATLAB® Builder™ JA 1.0 requires MATLAB® Compiler™ version 4.5.

The web service application was written using Apache Axis 1.4 package, which is a SOAP engine developed by Apache. Axis also requires an XML parser (refer to relevant links under the Setting up the web server and SOAP engine section). Axis recommends the Xerces parser, which was also used for this project.
The Java™ classes were built with JDK™ 1.5.0_6 and run against JRE™ 1.5.0_6. The web service was hosted on an Apache Tomcat 5.5 web server.

## Included Files

This example includes the MATLAB® Builder™ JA component as well as the Web application files. These files are described here:

### MATLAB® file and MATLAB® Builder™ JA project

All of the MATLAB® code used to build this project is included in the **`Source`** directory. It is organized into a MATLAB® Builder™ JA project with appropriate settings. The project file is named `temperatureComp.prj`.

### Web service

The **`Axis_Source`** directory includes the Axis web service, `TemperatureModule.java`.

### Web client

The **`Axis_Source`** directory includes the Axis web client, `TemperatureModuleClient.java`.

## Setting up the web server and SOAP engine

In order to run this example, Apache Tomcat 5.5 needs to be installed. The web server can be downloaded at the following site:

http://tomcat.apache.org/

Note: Throughout the rest this document, **$TOMCAT_HOME** will be used to refer to the root directory where Tomcat is installed and **$MATLABROOT** will be used to refer to the root installation directory of MATLAB® 7.3 (R2006b).

You will also need Apache Axis (Java™ version) 1.4 which is required to write the web service and client code:

http://ws.apache.org/axis/

Follow the instructions on Apache's website for setting up Axis to work with Apache Tomcat. The installation steps are outlined here:

http://ws.apache.org/axis/java/install.html

If you follow the instructions correctly, you should have the following libraries on **AXISCLASSPATH**: `axis.jar`, `commons-discovery-0.2.jar`, `commons-logging-1.0.4.jar`, `jaxrpc.jar`, `saaj.jar`, `log4j-1.2.8.jar`, `xml-apis.jar`, `xercesImpl.jar`, and `wsdl4j-1.5.1.jar`.

Once you have configured Tomcat to work with Axis, you should have a directory called **axis** in **$TOMCAT_HOME\webapps**.

Note that some additional libraries need to be downloaded to fully integrate Axis to work with Tomcat:

i. Axis requires an XML parser. The Xerces parser is recommended by Apache. The `xml-apis.jar` and `xercesImpl.jar files` must be added to **$TOMCAT_HOME \webapps\axis\WEB-INF\lib** and **AXISCLASSPATH**.

ii. JavaMail API: `mailapi.jar` must be added to **$TOMCAT_HOME \webapps\axis\WEB-INF\lib** and **AXISCLASSPATH**.

iii. JavaBeans Activation Framework: `activation.jar` must be added to **$TOMCAT_HOME \webapps\axis\WEB-INF\lib** and **AXISCLASSPATH**.

To allow Tomcat to recognize MATLAB® Builder™ JA components, you must copy the following file to the **$TOMCAT_HOME\common\lib** folder:

```
$MATLABROOT\toolbox\javabuilder\jar\javabuilder.jar
```

# Building the MATLAB® Builder™ JA component

The Java™ component used in this example can be built from the source MATLAB® code, `convertTemperature.m`, in the **Source** directory.

To build the component via the command line, type the following command:

```
mcc -W 'java:temperatureComp,temperatureClass' convertTemperature.m
```

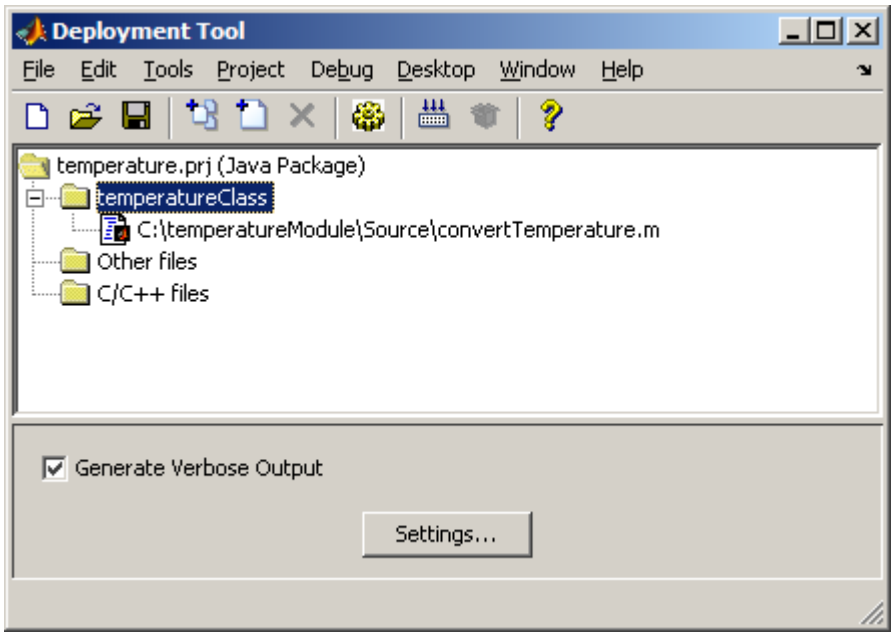
You can also build the component using the MATLAB® Builder™ JA project, which is included in the **`Source`** directory. This project file can be opened using `DEPLOYTOOL`. For a full list of instructions on what you need to setup before using `DEPLOYTOOL`, type the following command at the MATLAB® command prompt:

```
web([docroot,'/toolbox/javabuilder/ug/bqp8hfx-1.html'])
```

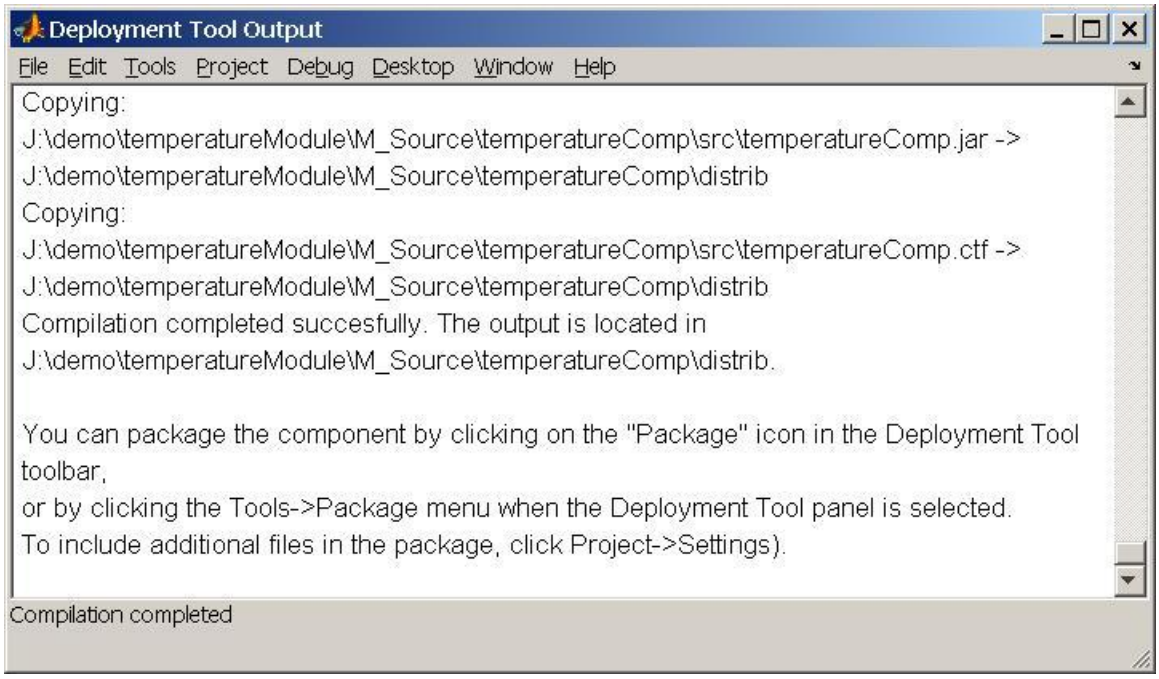
also accessible on the web here:

http://www.mathworks.com/access/helpdesk/help/toolbox/javabuilder/ug/bqp8hfx-1.html

Type `DEPLOYTOOL` and go to "File->New Deployment Project" to create a new project. Choose the appropriate component for MATLAB® Builder™ JA, naming the project file `temperatureComp.prj`. There should be a folder called `temperatureCompClass` that gets created. Rename this to `temperatureClass` by right-clicking on the folder name and choosing "Rename Class". Then, add the source file by right-clicking and choosing "Add File". Browse for the `convertTemperature.m` found in the **`Source`** directory.




To build the component, choose "Build" from the "Tools" menu. Verify that the component has been built successfully. The `DEPLOYTOOL` output window should say that the compilation is complete via some text to the effect of "Compilation completed."




You should see a new folder called **`temperatureComp`** that contains another folder called **`distrib`**, which in turn contains the component: `temperatureComp.jar` and `temperatureComp.ctf`.

Now you must make Axis aware of the built component. To do this, place `temperatureComp.jar` and `temperatureComp.ctf` in **`$TOMCAT_HOME\webapps\axis\WEB-INF\lib`**

## Building the Axis Web Service

Now that we have built the MATLAB® Builder™ JA component, we must create an Axis web service that will call this component. The **`Axis_Source`** directory contains the service, `TemperatureModule.java`. This class exposes a single method called `convertTemp` that takes in the temperature input and conversion string and returns the converted temperature(s). The input and output temperature(s) are specified as `java.lang.Object` to keep the input flexible.

The web service uses Java™ reflection to determine the type of the input. The input can be a scalar double, a two-dimensional array of doubles, a `TemperatureCellArray` type or a `TemperatureStruct` type. The latter two classes are used to indicate to the Web Service that the corresponding input to the component should be an `MWCellArray` or `MWStructArray`, respectively. Refer to the comments within the service class for further explanation.

One line of code in `TemperatureModule.java` worth mentioning is line 36, where the MATLAB® Builder™ JA component is instantiated. You must instantiate the MATLAB® Builder™ JA object specifying the full path to the CTF file's directory so that Tomcat can locate it.

To build the service, the following classes should be on your classpath: `javabuilder.jar`, `temperatureComp.jar`, and the libraries on **`AXISCLASSPATH`**. Note that you will need to use JDK™ 1.4 or 1.5, as listed in the following supported compiler technical note in order to build the Java classes:

http://www.mathworks.com/support/tech-notes/1600/1601.html

To make AXIS aware of the web service and utility classes, perform the following steps:

1. Build the `TemperatureCellArray.java`, `TemperatureStruct.java`, and `TemperatureModule.java` using the command line or your favorite Java™ IDE. This will generate class files in the following package structure:

   `com.mathworks.toolbox.javabuilder.examples`

2. Place the folder structure **`com\mathworks\toolbox\javabuilder\examples`** which contains the web service and utility classes into:

   **`$TOMCAT_HOME\webapps\axis\WEB-INF\classes`**

# Deploying the Axis Web Service

Now we must register the new web service with Axis. To do this, we need a web deployment descriptor which contains information about the class. More information about deployment descriptors can be found in the Axis User's Guide. The deployment descriptor for the temperature service, `deploy_temperature.wsdd`, can be found in the **`Deploy`** folder. The contents are as follows:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="temperatureModule" provider="java:RPC">
    <parameter name="className"
value="com.mathworks.toolbox.javabuilder.examples.TemperatureModule"/>
    <parameter name="allowedMethods" value="*"/>
    <beanMapping qname="myNS:TemperatureCellArray" xmlns:myNS="urn:TemperatureModule"
languageSpecificType="java:com.mathworks.toolbox.javabuilder.examples.TemperatureCellArra
y"/>
    <beanMapping qname="myNS:TemperatureStruct" xmlns:myNS="urn:TemperatureModule"
languageSpecificType="java:com.mathworks.toolbox.javabuilder.examples.TemperatureStruct"/
>
  </service>
</deployment>
```

As can be seen from the `name` attribute of the `service` tag, the service will be called `temperatureModule`. Important points to note are the `beanMapping` tags in the descriptor. These are used to tell Axis that we're passing customized beans, i.e., `TemperatureCellArray` and `TemperatureStruct` to the service. Axis includes the ability to serialize/deserialize, without writing any code, arbitrary Java™ classes which follow the standard JavaBean pattern of get/set accessors. The `TemperatureCellArray` and `TemepratureStruct` classes were designed to follow the JavaBeans™ pattern. For more information on using arbitrary classes, refer to "Encoding Your Beans – the BeanSerializer" in the Axis user guide.

To deploy the service, perform the following steps:

1. Start Tomcat.
2. Deploy the service using the `deploy_temperature.bat` file in the **`Deploy`** folder by double-clicking on it. This batch file assumes that you have set up an environment variable called **`AXISCLASSPATH`** as discussed in the Axis installation guide.

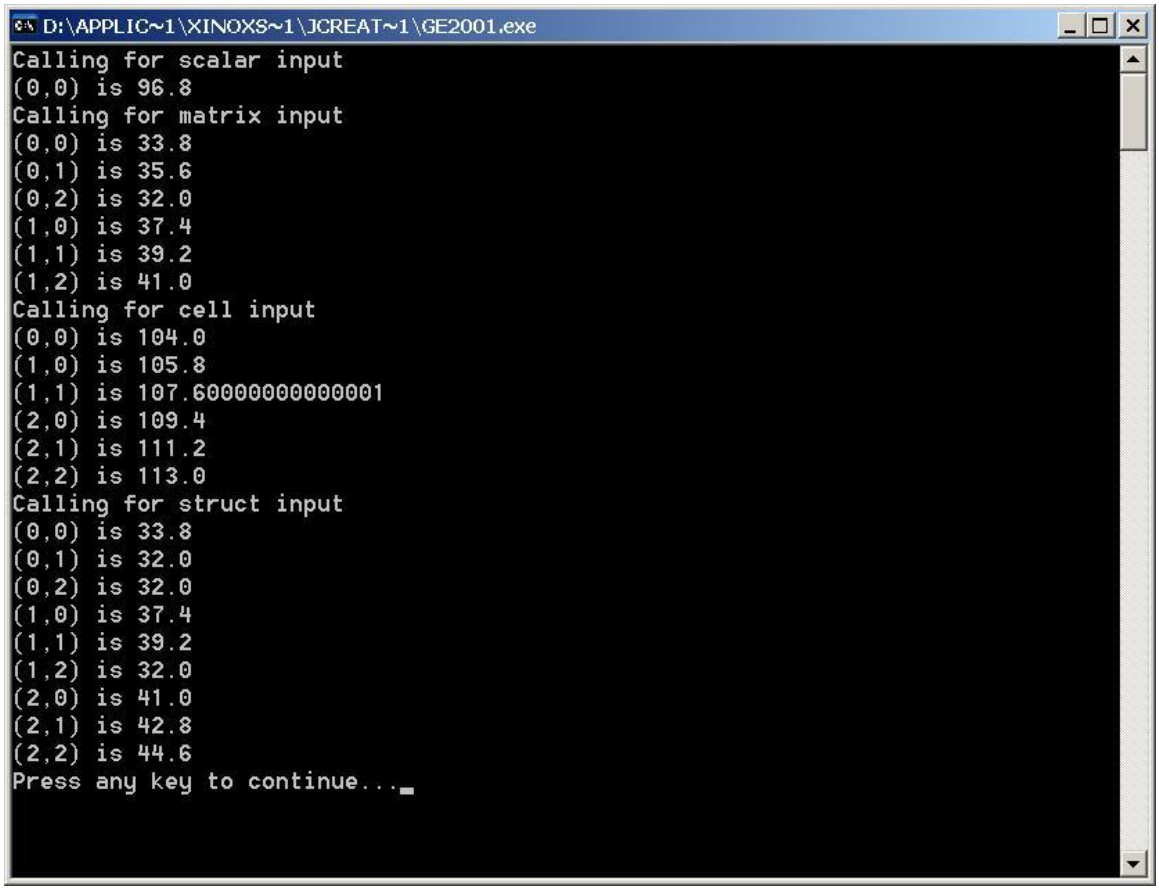
3. Verify that you see the service listed on the Axis page that displays the list of deployed services.


## Running the Axis client

Now you can run the client to connect to the web service and run some temperature conversions. The client code is `TemperatureModuleClient.java`, located in **`Axis_Source`**. To run the code, you will need to have all the libraries specified in **`AXISCLASSPATH`** on your Java™ classpath.

Pay special attention to the fourth case, where a jagged array of temperatures are passed in a `TemperatureStruct` object. The output that is returned and displayed is not jagged. The "holes" in the array were replaced with zeros on the MATLAB® side and converted to 32 degrees Fahrenheit. The reason why this differs from the `TemperatureCellArray` case is because of the way the input was created inside the web service. Refer to line 160 in `TemperatureModule.java`. An `MWStructArray` is instantiated and the input is specified directly as a `double[][]` rather than creating separate `MWNumericArrays` for each row of the matrix. See how this differs from line 87, where each cell of the `MWCellArray` is assigned a separate `MWNumericArray`.

Following are the results of running the client code:


D:\APPLIC~1\XINOXS~1\JCREAT~1\GE2001.exe
Calling for scalar input
(0,0) is 96.8
Calling for matrix input
(0,0) is 33.8
(0,1) is 35.6
(0,2) is 32.0
(1,0) is 37.4
(1,1) is 39.2
(1,2) is 41.0
Calling for cell input
(0,0) is 104.0
(1,0) is 105.8
(1,1) is 107.60000000000001
(2,0) is 109.4
(2,1) is 111.2
(2,2) is 113.0
Calling for struct input
(0,0) is 33.8
(0,1) is 32.0
(0,2) is 32.0
(1,0) is 37.4
(1,1) is 39.2
(1,2) is 32.0
(2,0) is 41.0
(2,1) is 42.8
(2,2) is 44.6
Press any key to continue...