

## **Using Java classes with ASP**

ASP is a great language, and you can do an awful lot of really great things with it. However, there are certain things you cannot do with ASP, such as use complex data structures or handle some heavy math operations, or even do anything remotely object-oriented.

As an ASP programmer for several years now, I've turned to Visual Basic to handle some of the more intensive things by creating ActiveX COM objects. COM objects are compiled and therefore process faster than ASP.

However, COM objects have a serious drawback: They must be registered to the server in order to be used, and they must be unregistered so that they can be modified.

There is a solution that's been used for a few years now that does look promising. It involves using ASP to use Java classes and Java data structures.

This article will walk you through a few examples in which we use ASP to bring java classes together.

### **Prerequisites**

In order to use this article, you need to be comfortable using ASP. You need to know your way around a Windows file system, and you need to have administrative privileges on a Windows web server so that you can restart IIS applications.

I will not be covering setting up or managing IIS applications in this article.

You will also need to be familiar with writing and compiling Java classes. I will be covering the Java sections of this article by using the Eclipse SDK, available at <http://www.eclipse.org/>

### **What Java is supported?**

Unfortunately, the only versions of Java that will work is anything that can work with the Microsoft JVM. This equates to something around Java 1.15

## Using Java Data Structures in ASP

Any of the java data structures can be used within ASP and will be treated like any other object.

You can use the `GetObject` function in ASP to return an object. The `GetObject` function can be used for many different things, such as connecting to the WINNT provider, working with WMI, or in this case, for creating instances of Java classes.

The syntax for the `GetObject` command is as follows:

```
Set MyObj = GetObject(command as string)
```

To use a Java object, your string must start with **Java:** followed by the package name and class.

A really simple example would be using the `Vector` class ( `java.util.Vector` ).

```
Dim objJavaObject
set objJavaObject = GetObject("java:java.util.Vector")
```

Once we have the object instance, we can use its methods just like we would with any other object. Here's the whole script.

```
<%
Option Explicit
Dim objJavaObject
set objJavaObject = GetObject("java:java.util.Vector")

objJavaObject.addElement "John"
objJavaObject.addElement "Tom"

` // Retrieve values
` // Remember, real programming languages kuse zero-based arrays!

response.write objJavaObject.elementAt(0) & "<br/>" & vbCrLf
response.write objJavaObject.elementAt(1) & "<br/>" & vbCrLf

`// Destroy the object when done!
set objJavaObject = nothing
%>
```

Save this as **UseVector.asp** and test it out. It should work fine. If it doesn't, you'll need to look over your typing and make sure you didn't make any mistakes. Also, check to make sure you have the JVM installed on your server.

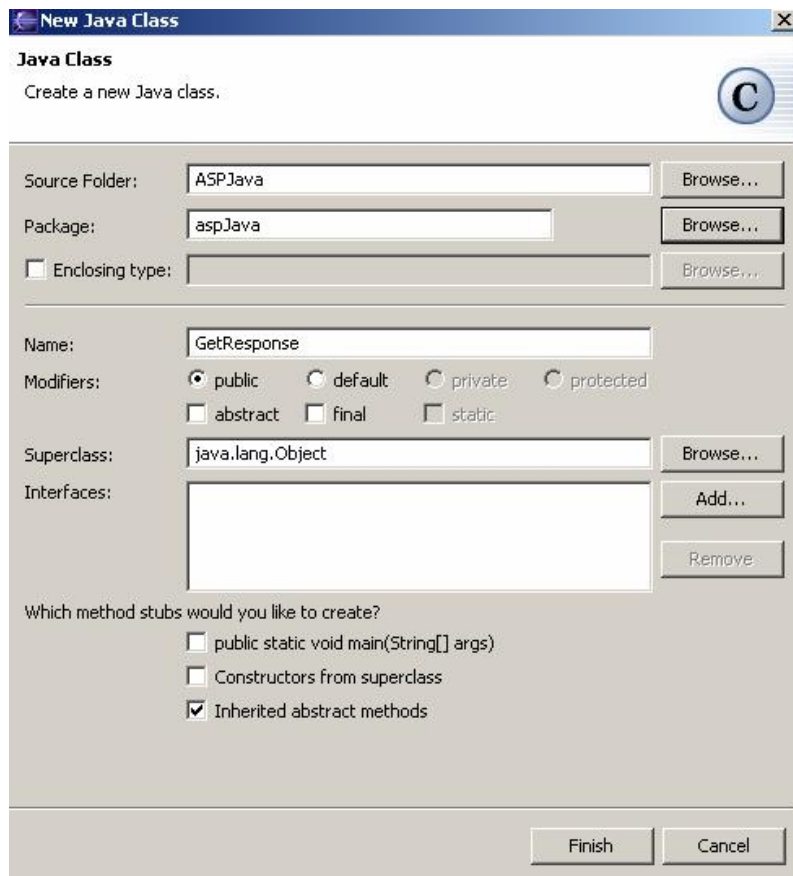
If you can't get this to work, you can't get the next example to work either.

## Creating and Using A Custom Java Class

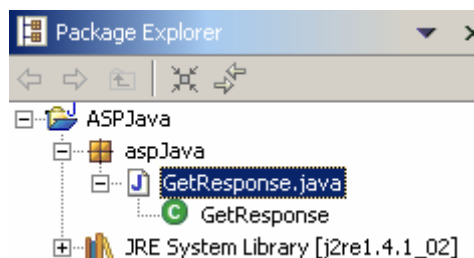
Assuming you already have Eclipse installed and you understand the Java language, we can proceed with this second example.

With the Eclipse SDK open, create a new project called “**ASPJava**” and create a new package in that project with the name “**aspJava**”

Create a new class called **GetResponse** and make sure it is placed in the **aspJava** package.



Click **Finish** to create the class.



You should see your package and class in the Package Explorer within Eclipse.

All class source files use the **.java** file extension. Classes will be compiled into **.class** files when they are compiled. This will be important to remember later.

Your default class file will contain the definition for the class, but you will need to create a constructor. unless you cheated before. If you're an ASP developer, I strongly suggest you get used to creating constructors on your own, rather than letting Eclipse create them for you.

A Constructor is basically the same as the **CLASS\_INIT** method in an ASP or VB class... it's called when the instance of the class is invoked. The constructor is declared as public and has the same name as the class.

```
public GetResponse() {  
  
}
```

Place the above code after the class definition.

Just like any other language, methods can be declared as **private** or **public**. **Public** methods can be used by other classes, and since we want to expose this object to ASP, we'll need some public methods.

So, we'll create a method that will return today's date.

```
public String getDate(){  
    String date = new java.util.Date().toString();  
    return date;  
}
```

This method returns a string containing today's date. The **java.util.Date()** object has a **.toString** method that converts the value of Date() to a string for us.

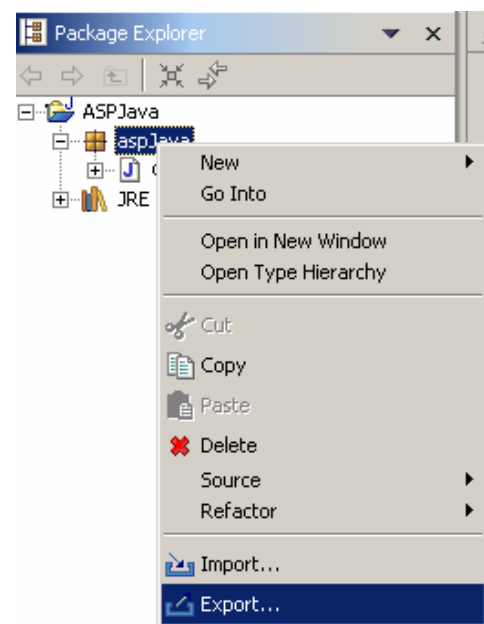
Save this class, because we're going to compile it for use on our server.

## Compiling to a Windows Server

In order to use Java classes from an ASP page, the classes must be placed in the **Java\Trustlib** folder located in the Windows folder. On NT4.0 and Windows 2000, this would be **c:\winnt\Java\Trustlib** and for WindowsXP and Windows Server 2003, this would be **c:\windows\Java\Trustlib**.

You can use Eclipse to compile your classes right to the **Trustlib** folder complete with the folders for the package.

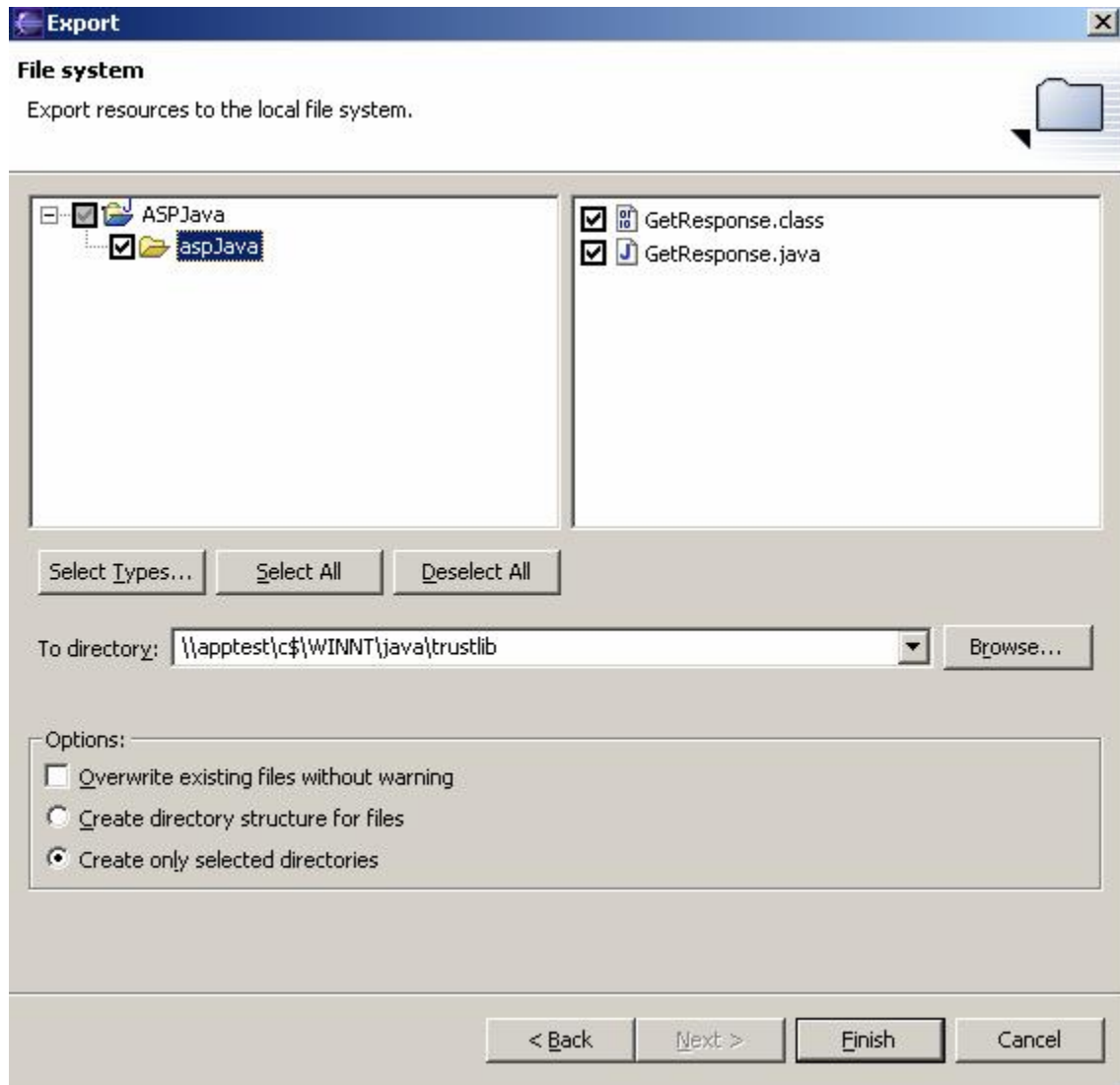
Right-click on the package in your Package Explorer and choose **Export** from the pop-up menu.



Choose **File System** from the list of export options and click the **Next** button.

Next, you will see the **File System** part of the Export Wizard. The left pane contains your package structure, and the right pane contains the files.

This version on Eclipse seems to have some bugs when exporting files. Therefore, we'll just be selecting the package folder on the left, and selecting both the CLASS and JAVA files on the right.



The output location should be to your server's TRUSTLIB folder. IF you have IIS running on your development machine, save the files to **c:\windows\java\trustlib** for WindowsXP or **c:\winnt\java\trustlib** for Windows NT and 2000.

I'm using an admin share to my **trustlib** folder on my Windows 2000 server.

## Building the ShowDate ASP page

Now that we have the files deployed to our server, we can easily invoke them by using a similar approach.

Our package **aspJava** contains our **GetResponse** class, so we invoke it with the **GetObject** function in ASP as follows:

```
set objJavaObject = GetObject("java:aspJava.GetResponse")
```

Then we can call the methods on **objJavaObject**.

Here's the complete code for the example.

```
<%  
  
    dim objJavaObject  
    set objJavaObject = GetObject("java:aspJava.GetResponse")  
    response.write objJavaObject.getDate()  
    set objJavaObject = nothing  
  
%>
```

Publish this page to an ASP-enabled folder on your web server and run it. You should see a page that looks like this:

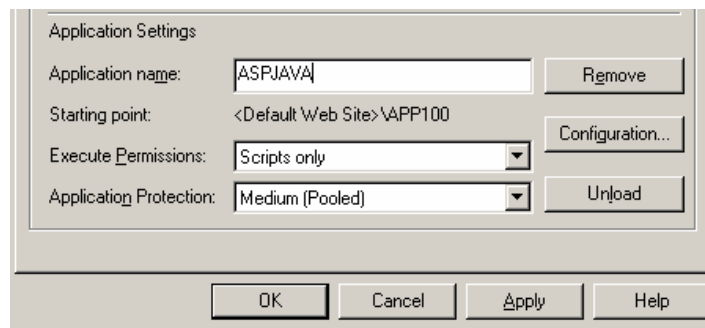
Mon Jun 09 11:38:57 CDT 2003

## Adding Additional Methods

That was pretty simple, but let's add some more methods to our object. This will then bring us to another "gotcha" about using Java with ASPs... IIS caches the objects in memory. In order to make changes to the Java classes, you have to **unload** the IIS application that uses the Java class. In some cases, this may involve restarting the entire IIS process!

To unload your application, use the IIS Management console, locate your application, right-click on the application folder and choose Properties.

On the Home Directory tab, there will be an area for application settings. Click the **Unload** button and all objects will be cleared from the server's cache.



Now that you've cleared your cache, let's go back to Eclipse and add a method to our **GetResponse** class.

```
public int getSum(int x, int y)
{
    return x + y;
}
```

This simple class will take in two integers as parameters and return their sum. Pretty easy, eh?

Save the file and deploy it to the server just like before. Then modify your previous ASP page.

```
<%
    dim objJavaObject
    set objJavaObject = GetObject("java:aspJava.GetResponse")
    response.write objJavaObject.getDate()
    response.write "<br/>"
    response.write objJavaObject.getSum(4,5)
    set objJavaObject = nothing
%>
```

Save this page and test it out. This time, below the date, you should see the sum of the parameters 4 and 5.

```
Mon Jun 09 11:53:34 CDT 2003
9
```

## Conclusion

This just scratches the surface of the things you can do when you leverage Java through an ASP front-end. You should have a good understanding of how this process works and you should be able to find creative ways to use this method to build better web applications.