WebSphere Application Server V4.01 for zOS and OS/390

Enabling the SOAP IVP

This document can be found on the web at:
www.ibm.com/support/techdocs
Search for document number **WP100241** under the category of "White Papers"

IBM Washington Systems Center

Donald C. Bagwell
301-240-3016
dbagwell@us.ibm.com

Many thanks to Duncan Elliott for his assistance in helping me understand how to run the SoapIVP program.

Table of Contents

(This page intentionally left blank)

An Overview of the SoapIVP Provided with WAS 4.01

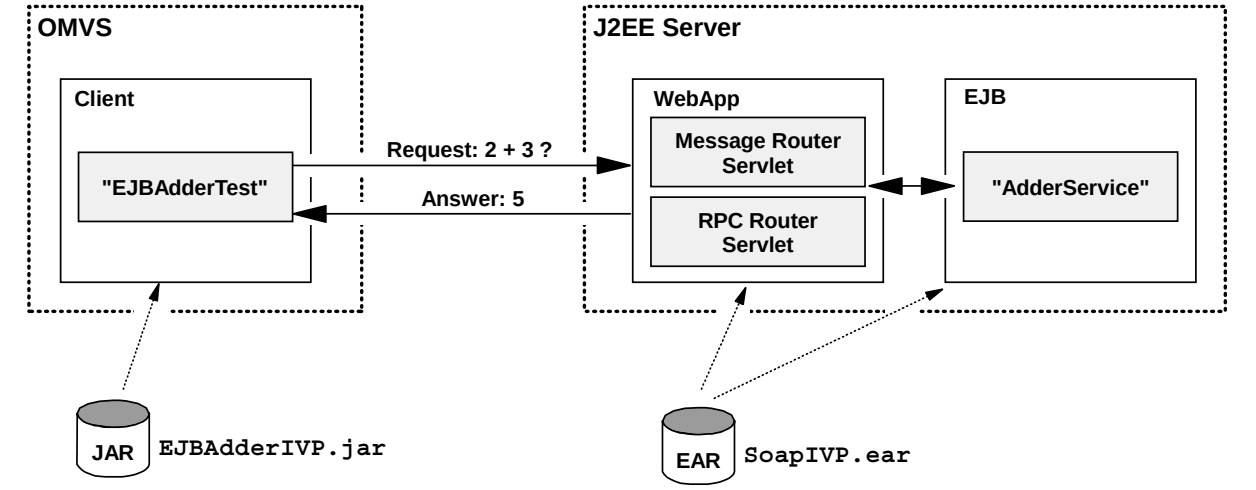
The SoapIVP application supplied with WebSphere V4.01 for zOS and OS/390 is a very simple application that takes as input hard-coded numbers (the integers 2 and 3) and adds them together to provide an answer of 5. A stunning leap forward in web services? Hardly. But it does fill its purpose of verifying that the SOAP support provided in WAS 4.01 does in fact work.

The verification program is provided in the form of an EAR file called `SoapIVP.ear`. It's location in the HFS is `/<install root>/samples/SoapIVP.ear`, where `/<install root>` is the location where WebSphere is installed on your system.

Note: Depending on the maintenance level of your system, the `SoapIVP.ear` file may not be a "resolved" file, which means you would have to run it through AAT before deploying it. APAR PQ53989 identifies this issue and the PTF for that APAR will provide a resolved EAR file. Trying to deploy the unresolved EAR file will result in an error.

Background: the overall picture

The SoapIVP application looks like this:



Overall picture of SoapIVP application

The `SoapIVP.ear` file contains both a web application and an EJB. The web application consists of two servlets, both of which are standard SOAP servlets required of any web service: the Message Router servlet and the RPC (remote procedure call) Router servlet. This web application receives the request from the client and routes it to the EJB that represents the "service." The EJB is known as "AdderService" and simply takes whatever two numbers it receives and adds them together.

Background: the SoapIVP "AdderService"

This is a *stateless* session bean that simply adds together the two integers you pass it. At this time the only kind of bean that's supported for SOAP services is a stateless session bean.

Background: the SoapIVP web application

This web application was generated by the tool used to create SOAP web services. (That tool is called SoapEAREnabler, and is supplied in the `/bin` directory of WebSphere 4.01. This document won't go into using that tool because it's not needed to run the `SoapIVP.ear` application. Someone has already run SoapEarEnabler and the result was the `SoapIVP.ear` file with all the SOAP parts inside.)

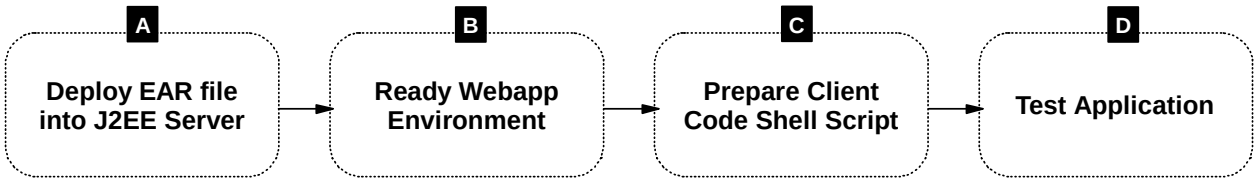
This web application has within it two servlets: one is called the `messagerouter` and the other is called `rpcrouter`. Together they provide the way in which someone across the web may ask for and receive the services of an EJB -- the `AdderService` EJB in this case.

Background: the SoapIVP client code that calls the service

The client code is a single Java class file called `ejbaddertest.class`, and it is executed out of the OMVS environment. This client provides the two integer values to be added (they are hard coded as "2" and "3") and then constructs the SOAP request to be sent. A shell script called `EJBAdderIVP.sh` is provided that sets the required `CLASSPATH` values and then invokes the `ejbaddertest` program. You invoke the shell script and pass it the host IP name of the system on which the web service is installed.

It's interesting to note that a web browser may not be used as a client directly accessing this web service. That's because a web browser will issue an HTTP "get", and the web service is expecting a "post" operation.

High-Level Overview of Process



Schematic overview of verification process

A. The EAR file is supplied in the following location in the HFS:

`/<install root>/samples/SoapIVP.ear`

Download that to your workstation and use the SMEUI tool to deploy the application into a J2EE server, setting the bean's JNDI path to `/soapivp/` and JNDI name to `AdderService`. The web application's JNDI information is set to the default.

See "Deploying the Application into the J2EE Server" on page 3 for more details.

B. Insure that the J2EE server's `webcontainer.conf` is configured, and that the context root for the application (`/SoapIVP`) can bind to a virtual host. If you're using the HTTP Server as the listening agent, add a `Service /soapivp/*` statement to the `httpd.conf` file.

See "Readying the Web Application Environment" on page 7 for more details.

C. Extract the client code shell script and provide your OMVS environment with `JAVA_HOME` and `WAS_HOME` variables.

See "Readying the SOAP Client Code Shell Script" on page 9 for more details.

D. Run the shell script and pass in as a parameter the virtual host (and port, if other than 80) to which the application is bound.

See "Driving the Application and Verifying SOAP Services" on page 12 for more details.

Deploying the Application into the J2EE Server

The `SoapIVP.ear` file supplied with the product is ready to deploy using the Systems Management End User Interface (SMEUI) tool. It contains one EJB and one web application.

Activity: FTP EAR file to workstation

Do following:

- ☐ FTP to your workstation the `SoapIVP.ear` file. It is located at the following HFS location:

  `/<install_root>/samples/SoapIVP.ear`

  Make certain to FTP in *binary format*.

  Note: Depending on the maintenance level of your system, the `SoapIVP.ear` file may be in a "resolved" EAR file, which means you will have to run it through the AAT tool prior to deploying the application. If you try to deploy an unresolved EAR file, you'll get an error. APAR PQ53989 addresses this issue.

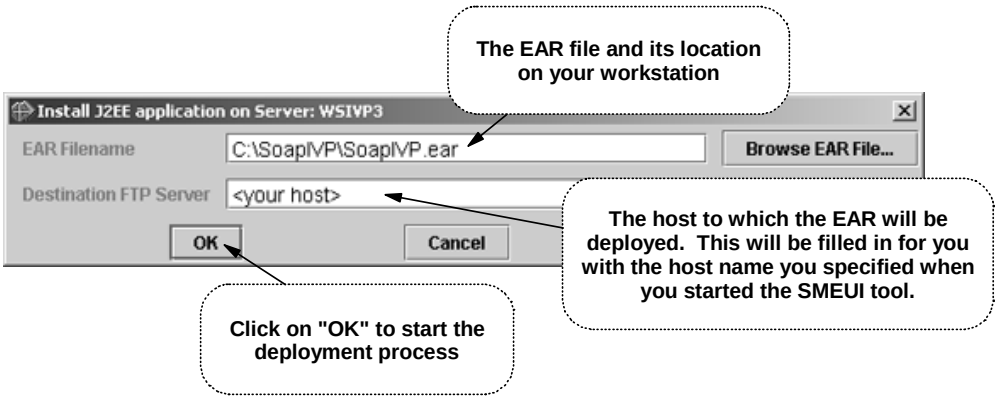
Activity: Start SMEUI, create conversation and select EAR file

- ☐ Start the SMEUI and create a new conversation.

  Note: Make sure you're using the level of SMEUI that is shipped with WAS 4.01. Earlier versions of the tool will not work with WAS 4.01. It should be at least Version **4.01.006**. If the version you have is back level from that, install the newer code by downloading the install executable from the HFS:

  `/<install_root>/bin/bboninst.exe`

- ☐ Expand the tree and locate the J2EE server into which you wish to deploy the SoapIVP application.
- ☐ Right-click on the J2EE server name, then select *Install J2EE Application*. The next panel looks like this:



Specifying the SoapIVP.ear for deployment

- ☐ After clicking "OK" on the previous panel, you will get the following message:

Message indicating application shouldn't be deployed into multiple-region servers

Don't worry: just click on "OK" to get by this.

Explain: This pops up because one of the XML deployment descriptors in the EAR file defines the activation policy for this EJB as "once." (Look inside the `soapivpejb.jar` file that's inside the `SoapIVP.ear` file. The file `websphere390xdd.xml` has a tag of:

```
<activation>once</activation>
```

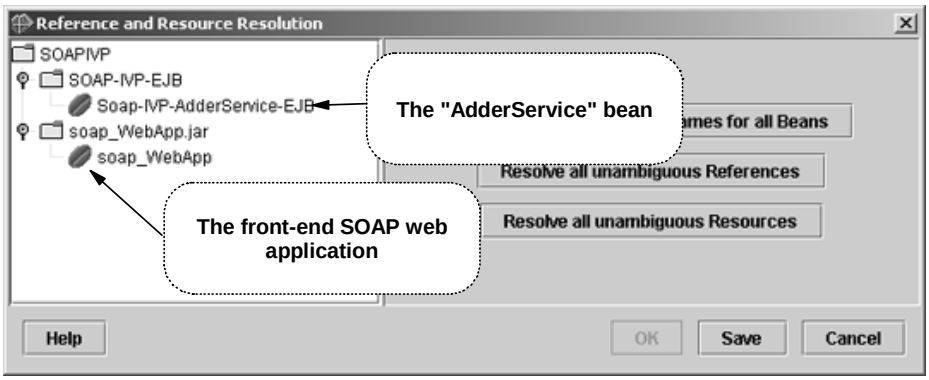
That means you shouldn't have copies of this running in different regions of the same server. Different servers is okay, but recall that WAS 4 allows multiple *server regions* of a given server to be started by WLM based on the workload its sees. You may very well have defined your WLM application environment with "support a single server region only" (which would prevent multiple regions from starting), but the SMEUI tool doesn't know that. So it issues this message.

This message also applies to multiple instances of the same server (also known as "replicated servers). So for applications defined as `<activation>once</activation>`, you should not deploy into servers defined with mulitple instances.

For the purposes of this SoapIVP application, don't worry about this message. You won't be driving enough workload against this to make a difference. Just be aware of why this message pops up.

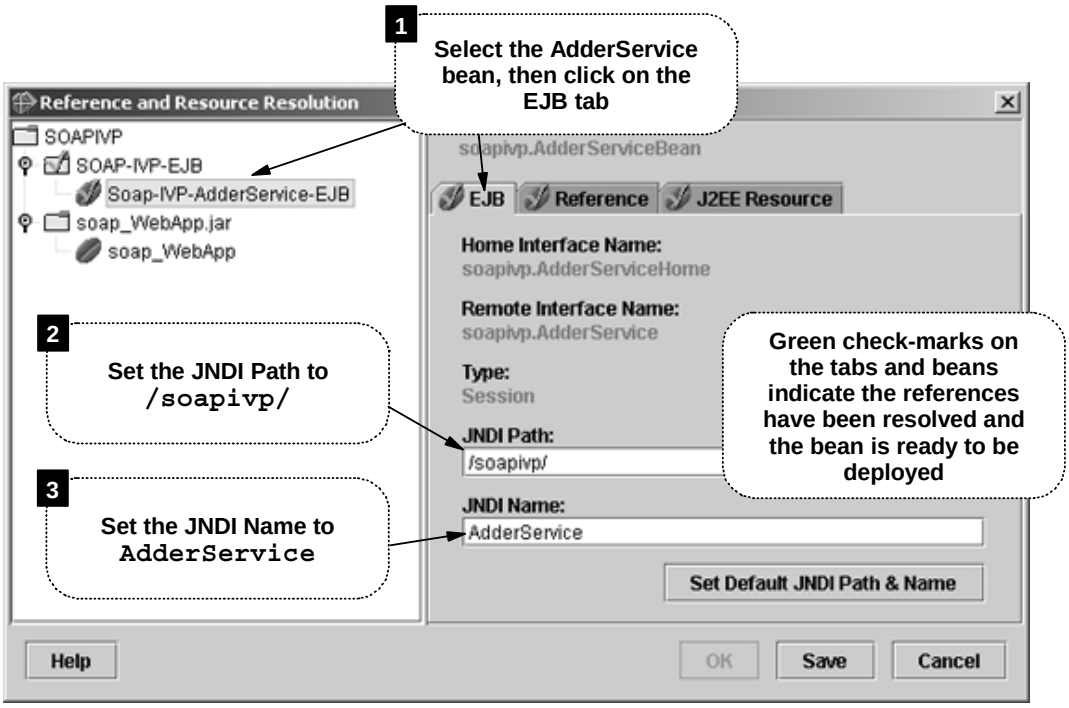
Activity: set JNDI properties for bean and webapp

☐ The next panel that pops up will show the contents of the EAR file:



Two beans -- EJB and webapp -- in the SoapIVP.ear file

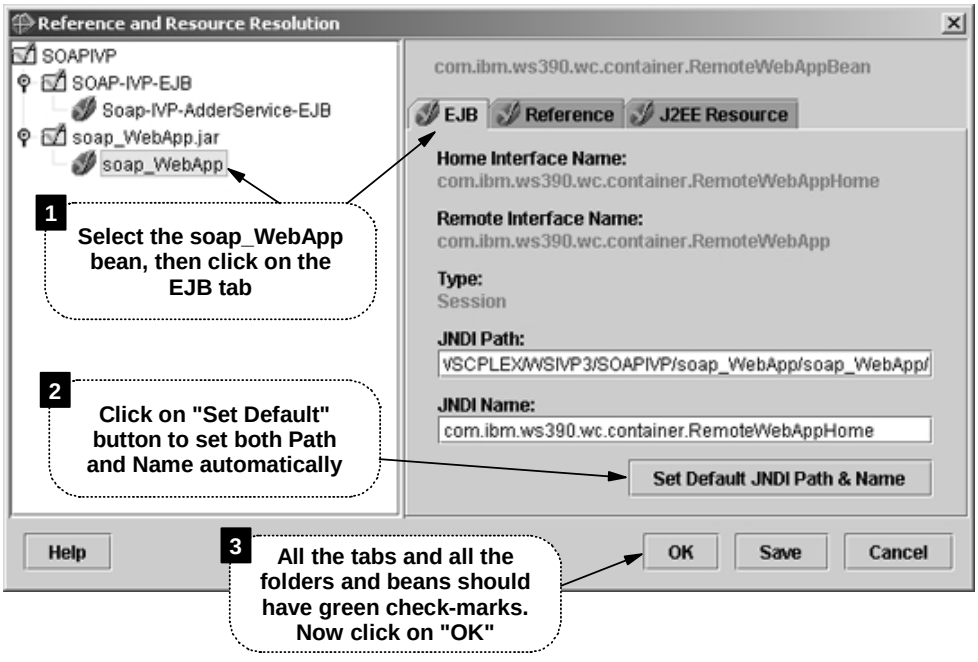
☐ Set the AdderService bean's JNDI path and JNDI name as illustrated here:



JNDI Path and Name for the AdderService bean

Note: It's important to set these values the way it is shown here. The client code you'll use to verify this has these values hard-coded. If you set these values to something different, it won't work.

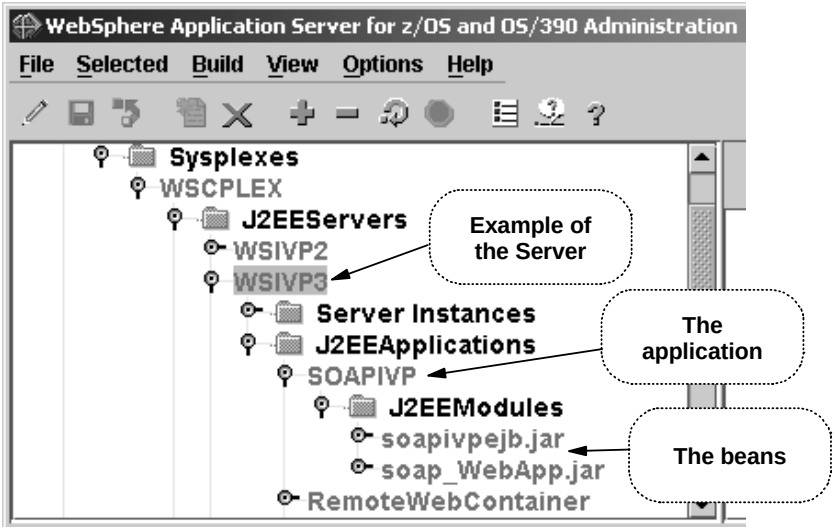
☐ Now set the web application's JNDI information:



Default JNDI Path and Name set for the web application

Activity: validate, commit and activate the conversation

The EAR file should now be deployed into the server. You can verify that by expanding the J2EE server and looking under "J2EE Applications":



SMEUI tool's view of the environment after EAR has been deployed into server

☐ Now you have to do the standard *Validate*, *Commit* and *Activate* of the conversation.

Readying the Web Application Environment

For this SoapIVP web service to work successfully after deploying the code, three things must be in place:

1. An HTTP listener must be ready to accept the request from the client
2. The J2EE web container for the server in which SoapIVP is deployed must be configured
3. The SoapIVP application must be successfully bound to a virtual host in the web container

This document will touch on each conceptually, but the actual checklist of configuration activities can be found in the following document:

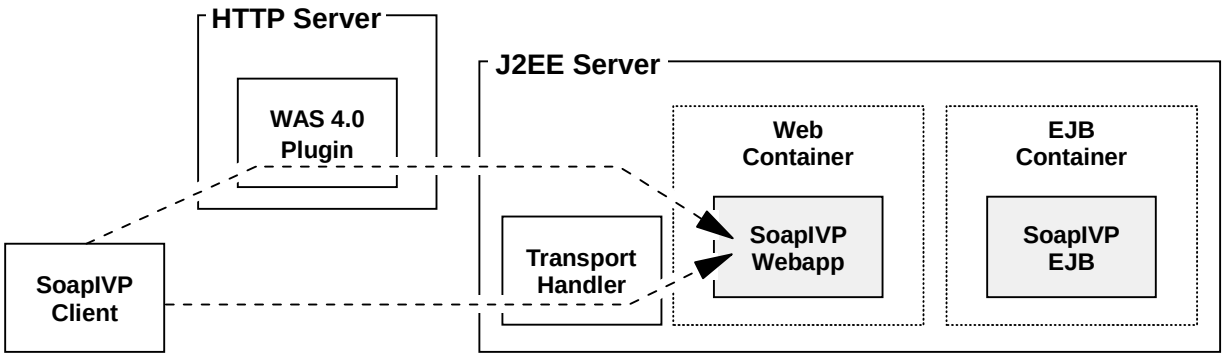
Other Documentation: An extensive write-up on configuring web applications in the WAS 4.0 and 4.01 environment can be found on the web at:

`www.ibm.com/support/techdocs`

Search for document number WP100238 under the category of "White Papers"

Background: The HTTP listener

Some form of HTTP listener needs to be part of the configuration. For WAS 4.01, that HTTP listener may be in the form of the IBM HTTP Server configured with the WAS 4.01 "plugin" code, or it may be the new WAS 4.01 "Transport Handler" function, which is an HTTP listener integrated into the J2EE server itself:



The SOAP client will gain access to the SOAP Service through the HTTP Server and the Plugin code, or through the new Transport Handler.

Choice of either HTTP Server+plugin or new Transport Handler

Both HTTP listeners will work equally well in testing the SoapIVP application. If you're unfamiliar with either HTTP listener, I would recommend you consult the WP100238 document referenced on page 7. That document provides background information as well as step-by-step configuration information.

Activity: if you're using the HTTP Server and Plugin

Note: This short section does not apply to the Transport Handler. It only applies when you're using the IBM HTTP Server with the WAS 4.01 plugin as your HTTP "catcher."

☐ If you are using the IBM HTTP Server and WAS 4.0 Plugin method of accessing the SoapIVP web application, you must add the following `Service` statement to your `httpd.conf` file:

```
Service  /soapivp/*   /usr/lpp/WebSphere/WebServerPlugIn...
```

This will allow the request to be mapped over to the plugin code and then routed to the webapp environment in the J2EE server.

- ☐ Make certain the `Userid` directive specifies a value that *does not* require user-supplied authentication. Typically this would be a *surrogate* ID such as `PUBLIC`:

```
Userid  PUBLIC
```

Why? The client code that is used (`ejbaddertest.class`) is going to issue an HTTP POST to the HTTP webserver. If the webserver is coded with `Userid %%CLIENT%%`, the webserver will return a request for user login. The client code is fairly simple code and wasn't written to handle this. It will fail with the following message returned to your OMVS or Telnet screen:

```
Exception in thread "main"  SOAPException:
 faultCode=SOAP-ENV:Client; msg=Failed to encode mime multipart:
 java.io.UnsupportedEncodingException: IBM-1047;
 targetException=java.io.IOException:
```

To avoid this, you need to allow the POST to be processed without that "please login" request coming back from the webserver.

Note: If you don't have a surrogate ID defined and you just want a way to test this in a quick-and-dirty way, you may code `Userid %%SERVER%%`. This will run the request under the userid of the webserver process, and no "please login" panel will be issued.

Coding `Userid %%SERVER%%` is a huge security exposure, and you would *never* code that for actual production use. But if your test system is properly isolated and you *promise* to remove this coding after you verify SoapIVP, then you may use it.

Background: The J2EE web container configuration

The web application that is part of the `SoapIVP.ear` file is deployed into what's known as the "web container" of the J2EE server. For that web application to be properly recognized, the web container needs to be configured with a `webcontainer.conf` file. It is based on definitions in the `webcontainer.conf` file that applications -- including SoapIVP -- are bound to virtual hosts. Without being bound to a virtual host, you'll never be able to access and test the SoapIVP application.

For more information on configuring the web container, see the document WP100238, which can be obtained at the website listed on page 7.

Background: binding SoapIVP to a virtual host

The `webcontainer.conf` provides a way for an application to be bound to a "virtual host." This is important because in order to run the application, the host specified on the URL coming in from the browser must match the "virtual host" to which the application is bound.

The document WP100238, which can be obtained off the web at the location specified on page 7, provides an extensive write-up on what a virtual host is, and how you bind applications to virtual hosts.

The SoapIVP web application has a "context root" value of `/soapivp`. Therefore, your `webcontainer.conf` file must have a `contextroots=` definition that allows the context root of `/soapivp` to bind to a virtual host

Why is this mentioned in this document? Because the virtual host to which the application is bound is what you'll need to specify as a parameter when you invoke the OMVS shell script to test the application. Provide a parameter that's different from the virtual host, and the test will fail. (See "Driving the Application and Verifying SOAP Services" on page 12 for information on invoking the shell script).

Readying the SOAP Client Code Shell Script

With the `SoapIVP.ear` file deployed into the server and the application successfully bound to a virtual host, you're ready to prepare the client code. It's tempting to think it's as easy as pointing a browser at the HTTP listener, but that won't work. The SOAP webapp doesn't accept HTTP "Get" requests, which is what browsers send. So the client code is a Java application that is run from the OMVS environment. That will send a properly formatted HTTP request to the SoapIVP webapp, which will then drive the service.

Activity: copy SoapIVPClients.jar file from installation directory

The Java class file that represents the client code is kept in a JAR file in the `/samples` directory of the WebSphere V4 installation root. That file also contains the shell script that's used to invoke the client code.

- ☐ Copy the following file:

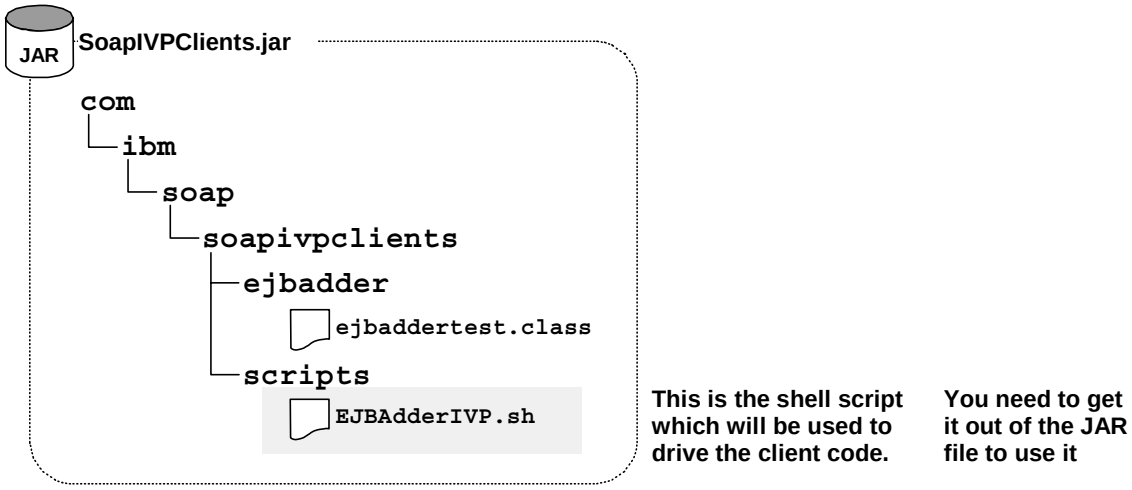
  From: `/<WAS installation root>/samples/SoapIVPClients.jar`

  To: `/<user directory>/SoapIVPClients.jar`

  The `<user directory>` is one you may safely write into without worrying about disturbing installation files. For example, `/u/bagwell/SoapIVP` was the one I used when testing this.

Activity: extract EJBAdderIVP.sh shell script from JAR file

Your interest here is the `EJBAdderIVP.sh` shell script, which is buried deep within the JAR file. The contents of the JAR file look like this:



Contents of the SoapIVPClients.jar file supplied with WAS 4.01

When you "explode" the entire JAR file, the directories contained within the JAR will be created in the HFS as well. But even if you want to extract one file (which is what you want to do), the directories that come before the file will be created in the HFS. That's not a problem, but it is a bit of a nuisance.

Do the following:

- ☐ Go into the OMVS shell of OS/390 (or access the system via Telnet)
- ☐ Change directories to the user directory in which you copied the `SoapIVPClients.jar` file.

- ☐ Issue the following command:

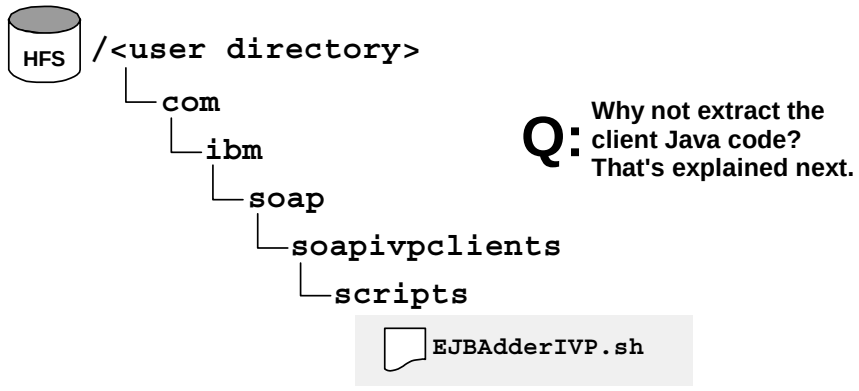
```
jar -xvf SoapIVPClients.jar com/ibm/soap/soapivpclients/scripts/EJBAdderIVP.sh
```

Note: If this fails with the message with an indication that the command "jar" is not found, then you'll probably have to update your `PATH` environment variable to point to the Java `/bin` directory (which is where the `jar` command resides). Find out where Java is installed on your system. Typically it's at a place like `/usr/lpp/java2/J1.3`, or something like that. Then, from your OMVS prompt, issue the following command:

```
export PATH=$PATH:/usr/lpp/java2/J1.3/bin
```

Then try the jar command again.

- ☐ If the command worked, you should have a directory structure that looks like this:



Directory structure after EJBAdderIVP.sh shell script extracted from JAR file

Note: The reason I don't have you extract the Java client code (the actual `ejbaddertest.class` file) is because the shell script is set up to look for that in the `/samples` directory of the installation root for WAS. Here's the `CLASSPATH` update in shell script that points to that:

```
CP=$WAS_HOME/samples/SoapIVPClients.jar
```

You could have extracted the class file as well as the shell script, but it wouldn't be used by the shell script. The shell script would go back to the JAR file that's in the `/samples` directory. To use the extracted class file would have meant updating the shell script to change the `CLASSPATH` entry. That's an unnecessary extra step. Rather than introduce that extra layer of complexity, I'm having you extract just the shell script and then allow it to go get the class file out of the supplied JAR in the `/samples` directory.

Activity: set permissions on shell script to permit execution

After extracting the `EJBAdderIVP.sh` shell script from the JAR file, the permissions on the JAR file will be `644`, which doesn't permit *anybody* to execute the script. Therefore, you'll need to change the permissions so the "execute bit" is on:

- ☐ Go into the OMVS (or Telnet) environment
- ☐ If you're not already there, change directories to the directory in which the `EJBAdderIVP.sh` file is stored. See the previous picture for that location.
- ☐ Issue the command `chmod 755 EJBAdderIVP.sh`

Activity: update your OMVS environment profile

For the shell script `EJBAdderIVP.sh` to work, two OMVS environment variables need to be set:

`JAVA_HOME`	Needs to be set to the directory in which the Java JDK is installed on your system. A typical location might be `/usr/lpp/java2/J1.3`
`WAS_HOME`	Needs to be set to the root directory in which WebSphere V4.01 is installed on your system. A typical location might be `/usr/lpp/WebSphere`

Do the following:

- ☐ Go into the OMVS shell (or Telnet) and issue the following two commands:

```
echo $JAVA_HOME
echo $WAS_HOME
```

- ☐ If the response received for each "echo" command indicated the variables are already set and set correctly, then skip to "Driving the Application and Verifying SOAP Services" on page 12. Otherwise, continue.
- ☐ If a `.profile` file exists in your userid's home directory, then edit that file; otherwise, create the file (note the "dot" at the beginning of that name).
- ☐ Code the following two lines into the `.profile` file:

```
export JAVA_HOME=$JAVA_HOME:/usr/lpp/java2/J1.3
export WAS_HOME=$WAS_HOME:/usr/lpp/WebSphere
```

> **Note:** The directory values in *italics* in the example above should be set to the directories in which Java and WebSphere are installed on your system.

- ☐ Exit the OMVS environment and re-enter. This will cause the new .profile to be executed.
- ☐ Issue the two "echo" commands again to verify the variables have been set:

```
echo $JAVA_HOME
echo $WAS_HOME
```

Driving the Application and Verifying SOAP Services

You are now ready to invoke the shell script and drive the service. Do the following:

- ☐ Make sure J2EE server in which the `SoapIVP.ear` file was deployed is started.
- ☐ If the HTTP access you are using is the IBM HTTP Server (as opposed to the WAS 4.01 Transport Handler), then make sure the HTTP Server is started.

  Note: If you are using the Transport Handler, you do not need to worry about starting that service. If configured into the J2EE Server, the Transport Handler starts automatically when the server is started.

- ☐ Go into the OMVS shell and change directories to:

  `/<user directory>/com/ibm/soap/soapivpclients/scripts`

  This is where the `EJBAdderIVP.sh` shell script resides.

- ☐ Issue the command:

  ```
  ./EJBAdderIVP.sh <IP host name of server>[:port]
  ```

  for example:

  ```
  ./EJBAdderIVP.sh wsc1.washington.ibm.com:8080
  ```

  Important: The value you provide for `<IP host name of server>` (and the port designation if you're using something other than 80) *must exactly match* -- character for character -- the "virtual host" to which the SoapIVP application is bound. If the virtual host in the web container is actually `wsc1:8080`, then that is what you pass as a parameter to the shell script. Passing a value that will resolve to the right network adapter is not enough; the received URL must match the virtual host specification exactly.

  You should see the following:

  ```
  ./EJBAdderIVP.sh wsc1.washington.ibm.com:8888
  5
  Done
  ```

That's it ... you've verified the SOAP services of WAS 4.01. If you deployed this `SoapIVP.ear` file into a copy of WAS 4.0 and tried to run it, you would get all sorts of "class not found" error messages. The fact that it ran in the WAS 4.01 environment proves the service (in the form of supplied class files) is present.

There's another aspect of this topic that's not explored in this document. That topic is the enabling of an existing EJB to be a "web service." At the present time you can do that with any stateless session bean. The "Assembling J2EE Applications" document (`SA22-7836-02`) goes into detail on the "SoapEAREnabler" tool, which will provide the necessary piece-parts to "SOAP-ify" a service.

Document Change History

Check the date in the footer of the document for the version of the document.

January 9, 2002	Updated with information regarding "UnsupportedEncodingException" when using the HTTP Server and plugin as your HTTP listening device.
November 14, 2001	Original document.

Index

(This page intentionally left blank)

End of Document