OpenVMS Technical Journal V7

Reusing OpenVMS Application Programs from Java

David J. Sullivan, Expert Member Technical Staff

Overview

This article introduces a new product developed by HP OpenVMS engineering. The Web Services Integration Toolkit (WSIT) for OpenVMS was designed to ease the burden of calling non-Java applications from Java applications. Non-Java applications are typically older, stable, and provide a significant business value. An example of a non-Java application might be a C or COBOL application written in the 1980s. Java applications are typically newer and leverage the latest technology. An example of a Java application might be a web service application called by Microsoft .NET. With WSIT, these two applications can be easily integrated while keeping the original application running in its current form.

The Web Services Integration Toolkit can be used to integrate application libraries written in programming languages such as C, BASIC, COBOL, and FORTRAN. It also has support for integrating ACMS applications.

The latest kit can be obtained from the website http://hp.com/products/openvms/wsit. The kit will be bundled with the next OpenVMS Alpha e-Business CD and the Foundation Operating Environment (FOE) on HP OpenVMS Integrity servers.

The Problem Addressed by WSIT

The popularity of the Java programming language has grown within the OpenVMS installed base because of its platform neutrality and ease of use. However, businesses using OpenVMS cannot rewrite all of their applications in Java -- they need to reuse the logic in their older non-Java applications. Unfortunately, Java was not designed to allow developers to call programs written in other languages. The exception here is C. Java does have some low level support for calling C programs, but it is very difficult and cumbersome to use. Simply put, writing the code to call legacy application from Java is difficult, time consuming, and error prone.

WSIT was designed to address this issue. It handles all of the difficulties of writing Java to some-other-

© Copyright 2006 Hewlett-Packard Development Company, L.P

language integration code. The toolkit is composed of small and simple tools. These tools can be extended and customized by the developer. Each tool is specific and obvious in it use. The developer using WSIT is able to wrap older application libraries and exposes them as a Java classes.

The Opportunity for Web Services and OpenVMS

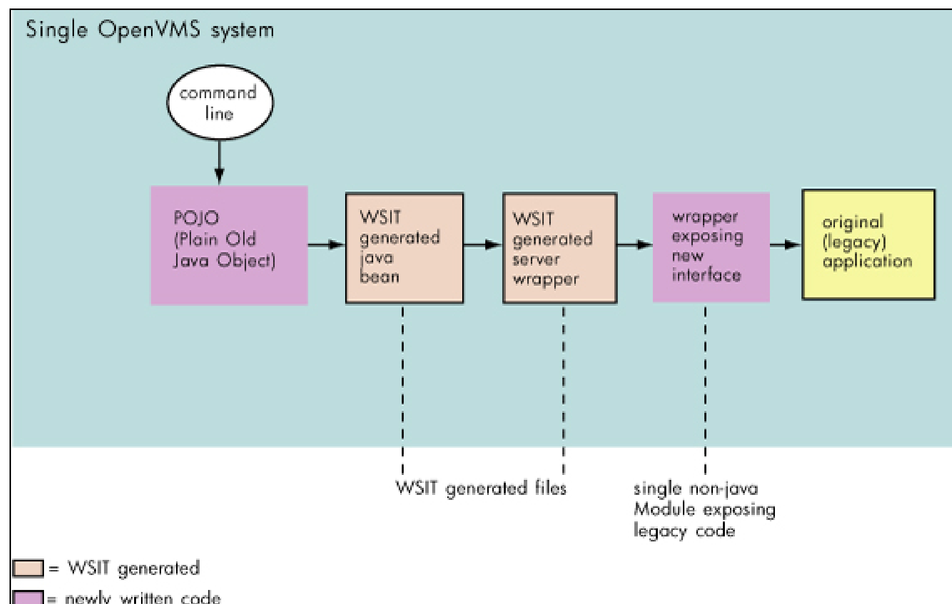
The toolkit generates Java classes that can be called from any Java technology. However, a popular use of these Java classes is likely to be from web services. Web services are the fastest growing integration technology. They are designed for language-neutral and operating-system-neutral application integration. OpenVMS applications are perfect candidates to benefit from web services.

Use Scenarios

As mentioned earlier, the Java classes generated by WSIT can be called from any Java technology on OpenVMS. Some of the most popular technologies for calling the WSIT Java classes are:

- POJO (Plain Old Java Object), perhaps accessed by a command line interface
- JSP (Java Server Page), accessed by a web browser
- Web service accessed by a web service client on any platform (for example Microsoft .NET)

In the following section we will take a high level look at each of these scenarios.



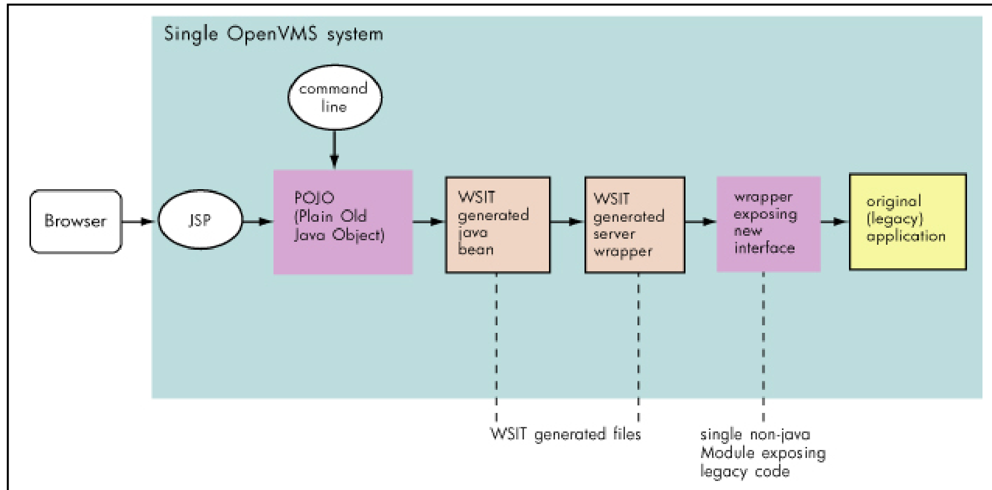
POJO (Plain Old Java Object) Called from the Command Line

This is the most basic scenario. It is a great way to get started using WSIT without having to worry about writing clients in other environments. The figure below illustrates a simple Java class calling another Java class which was generated by WSIT. The generated classes forward the clients calls to the non-Java application and return the result.

Note that all the code executes on a single OpenVMS system.

The POJO with command line processing can be written using any version of the [Java Software Development Kit (SDK) for OpenVMS](#).

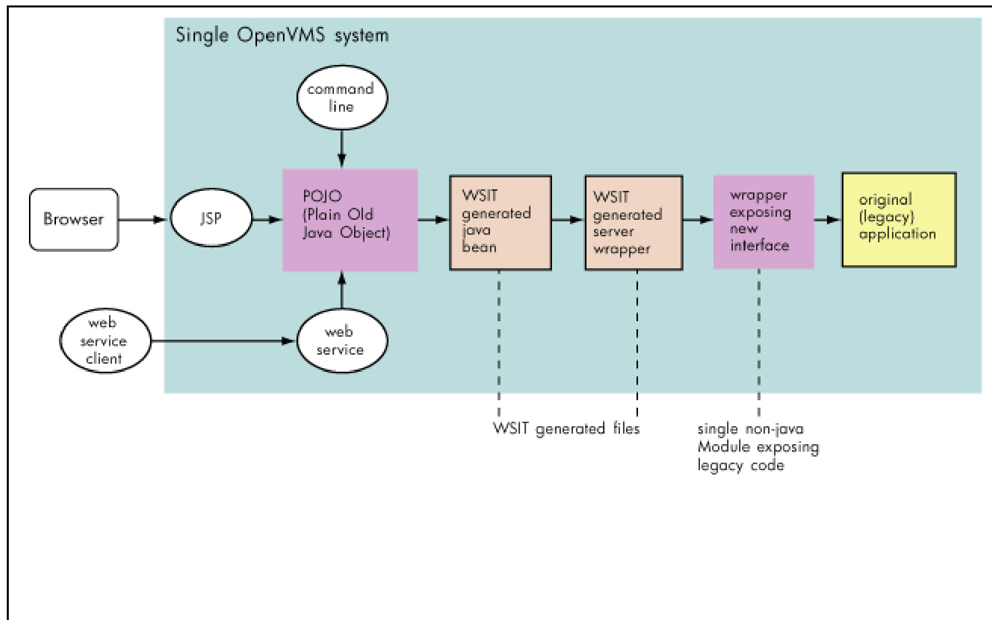
© Copyright 2006 Hewlett-Packard Development Company, L.P



Java Server Pages (JSPs) Called from Browser

This scenario illustrates how to add a web interface to an older non-Java OpenVMS application. A JSP on OpenVMS provides a web base interface to a browser. When the user clicks on the browser web page, it triggers the appropriate calls to the JSP. The JSP then forwards the calls to the Java class from the POJO scenario. Note that the POJO may still have the command line interface in addition to the web interface.

JSP are deployed using web servers. Web servers on OpenVMS include: Apache Tomcat, BEA WebLogic Server.



Web Services and Web Services clients (Microsoft .NET, J2EE, Java)

This scenario illustrates how to reuse the OpenVMS application library by exposing it as a web service that can be called from any operating system and language that supports web service clients. The web service client connects to the service on OpenVMS and calls a method. The service forwards the request to the POJO which returns the result of the methods call. This data is then returned to the web service client.

Web Service Engines on OpenVMS include: Apache Axis, BEA WebLogic Server.

Web Service Clients can be written in many different ways including: Microsoft .NET applications, Java JAX-RPC clients, J2EE application servers such as BEA WebLogic Server, most handheld software, and many more.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Using WSIT

Next we will take a look at the WSIT tools and see how they are used to wrap a non-Java application.

The Goal

For the purpose of illustration we will use a very simple application written in the C programming language. For future reference, more complex sample applications in various languages, as well as an ACMS application, are included in the WSIT kit.

The math application has two routines, *sum* and *product*. The interface is exposed in the file DISK$:[VTJ]math.c

```
$ set default DISK$:[VTJ]
$ ty math.c

unsigned int sum ( int number1, int number2) {
  return number1 + number2;
}

unsigned int product ( int number1, int number2) {
  return number1 * number2;
}
$
```

The purpose of the WSIT tools is to generate a Java class that presents a Java version of the math.C routines. The Java interface should look similar to the figure below.

```
public interface Imath {

    public int sum (int number1, int number2)
                            throws WsiException;

    public int product (int number1, int number2)
                            throws WsiException;
}
```

WSIT provides the following tools: The use of these tools will become clear as we look at the typical development steps. For now just know that these tools exist.

OBJ2IDL.EXE (I64 only) takes an OpenVMS object file (.obj) as input and generates an XML description of the applications interface. Referred to in this article as an XML IDL file.

© Copyright 2006 Hewlett-Packard Development Company, L.P

STDL2IDL.JAR the ACMS equivalent to obj2idl.exe. It takes an ACMS STDL description as input and generates an XML description of the applications interface. Referred to in this article as an XML IDL file.

VALIDATE.JAR takes an XML IDL file and validates the XML against the OpenVMS IDL schema.

IDL2CODE.JAR takes an XML IDL file and generates a Java class with the same interface. The Java class knows how to call the original non-Java application.

Typical Development Steps

Step 1: Prepare the application. [Not required but highly encouraged]

Step 2: Describe the interface with an XML Interface Definition Language (IDL) file.

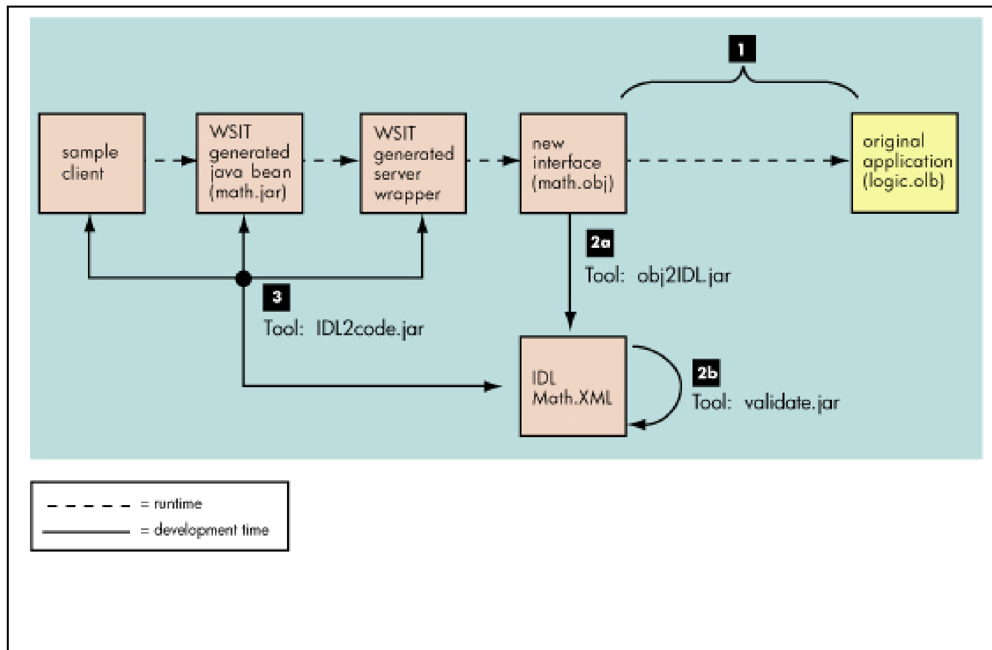
- Tools: **OBJ2IDL.EXE (for C,BASIC,COBOL,FORTRAN), STDL2IDL.JAR (for ACMS), VALIDATE.JAR**

Step 3: Generate Java wrapper based on XML IDL file from step 2.

- Tools: **IDL2CODE.JAR**

Step 4: Test the generated code from a client.

- Tools: **IDL2CODE.JAR** (WSIT version 1.1 or higher)

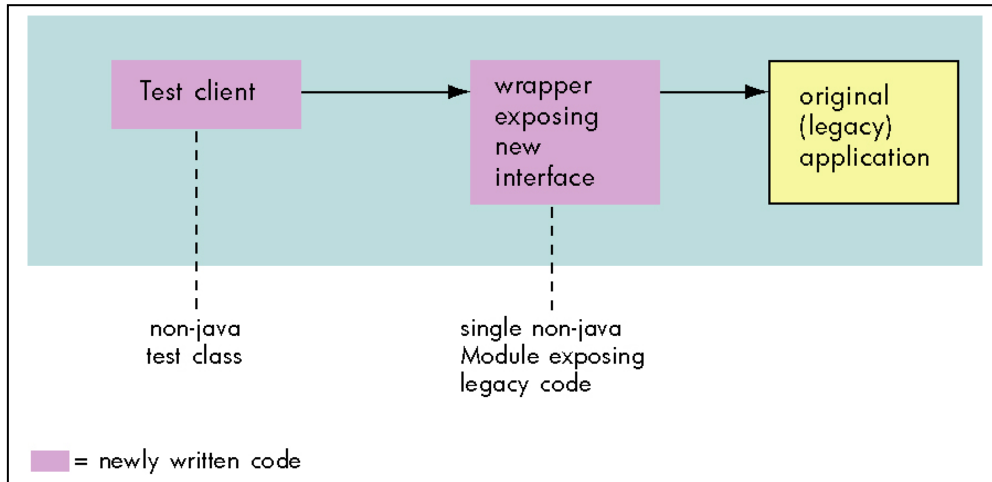


Step 1: Prepare the application [not required but highly encouraged]

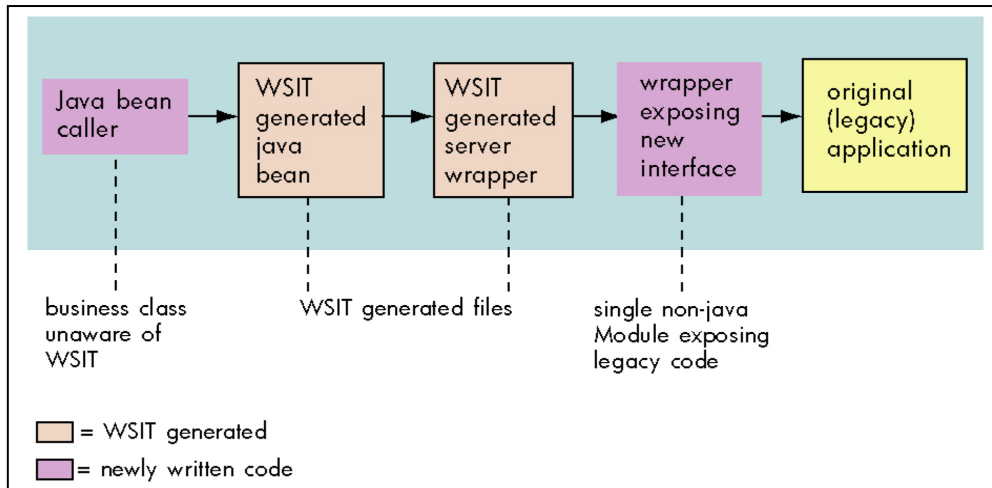
Before using any integration technology, you should evaluate the original application. The application is likely to have been written long ago and will benefit from having a wrapper expose a new and clean interface. The new interface will expose the legacy implementation. Separating the interface from the implementation provides encapsulation and the ability to easily extend and reuse the implementation.

© Copyright 2006 Hewlett-Packard Development Company, L.P

To avoid complexity, these new interfaces should be tested with a simple client before you use the Web Services Integration Toolkit. When you know that the interface classes are working properly, you can use WSIT to extend the use of the new interface to the Java environment.



After you have prepared the application, WSIT can extend the features of the new interface to Java, as shown in the following diagram.



Step 2: Describe the interface

The WSIT development tools generate and consume a simple XML file that describes an application's interface.

You create an XML IDL file using the tool named **OBJ2IDL.EXE** (for 3GL languages) or **STDL2IDL.JAR** (for ACMS).

Note: OBJ2IDL.EXE runs on OpenVMS I64 only. If you are using WSIT on OpenVMS Alpha, you can create the XML IDL file manually in any editor using the many samples as a guide. If you have both OpenVMS I64 and Alpha systems, run OBJ2IDL.EXE on I64 and copy the resulting XML IDL file to your Alpha system.

To generate an XML IDL File for the math.c application interface, use the command line below. The obj2idl executable accepts a switch ***–f*** which specifies the name of the object file from which to extract the interface definition. The object file must be compiled with /debug /noopt.

© Copyright 2006 Hewlett-Packard Development Company, L.P

```
!Establish WSIT logicals
$ @wsi$root:[tools]wsi-setenv - wsi$dev


!Establish a foreign command for the obj2idl tool:
$ obj2idl = "$WSI$ROOT:[tools]obj2idl.exe"


!Compile the wrapper that exposes the new interface:
$ set default DISK$:[VTJ]
$ cc/debug/noopt math.c


!Use obj2idl to generate an xml file with the interface
definition:
$ obj2idl -f math.obj
```

The XML file generated by the obj2idl tool is shown in the appendix. Even those unfamiliar with XML should be able to understand what this file is doing.

Whenever you use the obj2idl tool you must verify that the XML IDL file correctly describes the interface being exposed. If it does not, manually update the XML IDL file until the interface definition is correct.

If you have modified the XML file you can ensure that it is still well-formed and valid by using the tool **VALIDATE.JAR**. For those unfamiliar with XML, an XML file is considered well formed if it is syntactically correct. The file is considered valid if it is semantically correct. The XML rules for validity are defined in the file openvms-integration.xsd.

The validate tool is an executable jar file that accepts the following arguments:

-x DISK$:[VTJ]math.xml is a switch that specifies the name of the XML file to be validated.

-***s wsi$root:[tools]openvms-integration.xsd*** is a switch that specifies the XML schema file defining the semantic rules for validating the XML file specified with the switch -x.

```
$ java -jar wsi$root:[tools]validate.jar -x
DISK$:[VTJ]math.xml -s wsi$root:[tools]openvms-
integration.xsd
XML file validated sucessfully
$
```

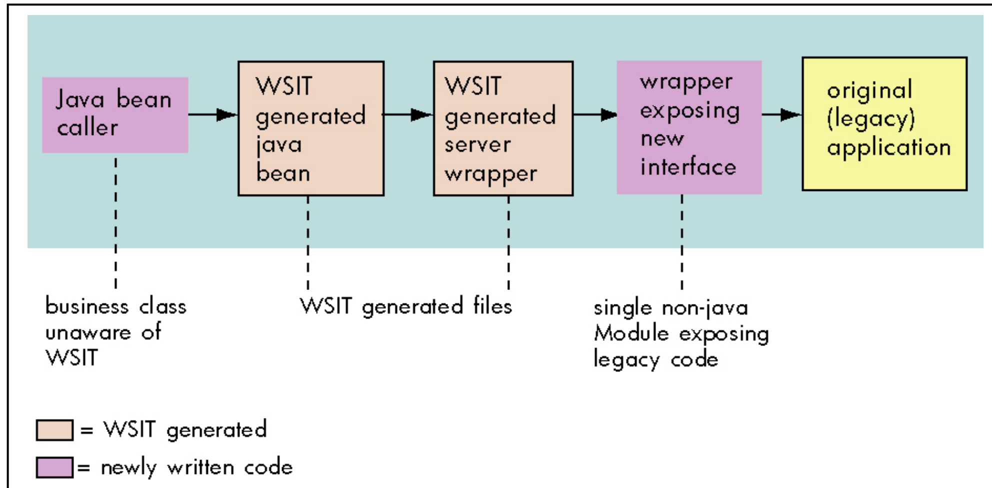
© Copyright 2006 Hewlett-Packard Development Company, L.P

Step 3: Generate code

The **IDL2CODE.JAR** tool generates the necessary code to wrap the routines described in the XML IDL file. There are two components generated:

1. One WSIT server wrapper: this code knows how to call the routines in math.obj.

2. One WSIT JavaBean: This code is the Java version of the routines in the math application. It also knows how to call the server wrapper.



The figure below illustrates how to use the **idl2code** tool. The main routine in the tool is named **com.hp.wsi.Generator**. The arguments are as follows:

-i math.xml is a required switch that specifies the name of the xml file that describes the interface being wrapped.

-a math is a required switch that specifies the name to be used for the generated files.

-c SJ is an optional switch that specifies that one or more sample clients should be generate. The argument S will generate a command line based interface client. The argument J will generate a JSP web based interface client. We will look at the clients in more detail in later. Note that this switch was added in WSIT version V1.1. If you are using an earlier version, the switch will be ignored.

-o [.generated] is an optional switch that specifies a root directory where the generated files should be placed.

Note: the command below illustrates a new switch added in WSIT Version 1.1. If you are using an earlier version of WSIT the switch ***–c SJ*** *will be ignored in the command line.*

© Copyright 2006 Hewlett-Packard Development Company, L.P

```
$ set default DISK$:[VTJ]
$ java "com.hp.wsi.Generator" -i math.xml  -a math -c SJ -o
[.generated]
File: ./generated/mathServer/build-math-server.com generated.
File: ./generated/mathServer/methIds.h generated.
File: ./generated/mathServer/structkeys.h generated.
File: ./generated/mathServer/math.wsi generated.
File: ./generated/mathServer/math.opt generated.
File: ./generated/mathServer/math-server.h generated.
File: ./generated/mathServer/math-server.c generated.
File: ./generated/math/build-math-jb.com generated.
File: ./generated/math/Imath.java generated.
File: ./generated/math/mathImpl.java generated.
File: ./generated/mathSamples/POJO/mathMain.java generated.
File: ./generated/mathSamples/POJO/build-math-PoJoClient.com
generated.
File: ./generated/mathSamples/JSP/index.html generated.
File: ./generated/mathSamples/JSP/mathMethodList.html generated.
File: ./generated/mathSamples/JSP/mathPopulate.jsp generated.
File: ./generated/mathSamples/JSP/mathDoCall.jsp generated.
File: ./generated/mathSamples/JSP/build-math-JspClient.com
generated.*** Application math generated! ***
$
```

<u>Build the generated server wrapper</u>

The Server build procedure creates an executable named math.exe. This file was linked with math.obj. Then math.exe is automatically copied to the WSIT deployment directory wsi$root:[deploy]

```
$ set default DISK$:[VTJ.generated.mathServer]
$ @BUILD-MATH-SERVER
Begin server build procedure.
  ..configuring switches and compiler options
  ..compiling native server code
  ..linking shareable image
  ..installing server image
End server build procedure.
$
```

© Copyright 2006 Hewlett-Packard Development Company, L.P

Build the generated JavaBean

The JavaBean build procedure creates a JAR file that contains the WSI Java classes used to call the server wrapper generated earlier.

```
$ set default DISK$:[VTJ.generated.math]
$ @BUILD-MATH-JB
Begin java bean build procedure.
The New JAVA$CLASSPATH is:
   "JAVA$CLASSPATH" = "WSI$ROOT:[LIB]WSIRTL.JAR" (LNM$JOB_86D5AE00)
        = "[]"
        = "WSI$ROOT:[LIB]VELOCITY-DEP-1_4.JAR"
        = "WSI$ROOT:[TOOLS]IDL2CODE.JAR"
  ..Compiling structure classes
  ..Compiling math Interface classes
  ..Creating math.JAR file from classes
End of JavaBean build procedure.
$
```

Congratulations! You have used WSIT to generate a wrapper for the math applications routines. The generated Java wrapper is packaged in DISK$:[VTJ.generated.math]math.jar

Step 4: Test the generated code from a client

In the previous step the switch **–c SJ** was used to tell the WSIT generator to generate a sample client with a command line interface (**S**) and to also generate a sample client with a JSP interface (**J**). These samples are provided for your convenience. They are intended to ease testing and development when using WSIT.

Using the generated POJO client sample

Normally this class will be written to integrate the WSIT generated JavaBean with the Java technology of your choice.

The sample must first be built as illustrated in the figure below.

```
$ @wsi$root:[tools]wsi-setenv - DISK$:[VTJ.generated.math]math.jar
$ set default DISK$:[VTJ.generated.mathSamples.POJO]
$ @build-math-PoJoClient
Begin client build procedure.
  ..Compiling mathMain client class
End of client build procedure.


To run this client, type the following at the command line:
     $ java "math.mathMain"
$
```

© Copyright 2006 Hewlett-Packard Development Company, L.P

The sample client is able to make calls to the methods of the generated JavaBean. There is a limitation that only methods with primitive arguments can be called. To see which methods the sample client can call use the switch **–m** as in the figure below.

```
$ set default DISK$:[VTJ.generated.mathSamples.POJO]
$ java math.mathMain -m
 The list of available methods within math are:


    sum(int P1, int P2)
    product(int P1, int P2)


(Methods that take structures or arrays as parameters are not callable
from this command line interface.  These methods are denoted by the *
next to them.)
$
```

To call the *sum* and *product* methods with arguments of 5 and 2 use the commands below.

```
$ set default DISK$:[VTJ.generated.mathSamples.POJO]
$ java math.mathMain -m sum -p1 5 -p2 2
Calling mathImpl.sum:
 P1 = 5
 P2 = 2
 Return value = 7
 ++ The client was successful ++

$ java math.mathMain -m product -p1 5 -p2 2
Calling mathImpl.product:
 P1 = 5
 P2 = 2
 Return value = 10
 ++ The client was successful ++
$
```

Using the generated JSP client sample

As with the POJO sample, the JSP sample client must first be built as illustrated in the figure below.

© Copyright 2006 Hewlett-Packard Development Company, L.P

```
$ @wsi$root:[tools]wsi-setenv - DISK$:[VTJ.generated.math]math.jar
$ set default DISK$:[VTJ.generated.mathSamples.JSP]
$ @build-math-JSPClient
Begin JSP client build procedure.
Unpacking static files into current location.
Copying math.jar into local [.WEB-INF.lib] directory
Creating mathJsp.war file
End of JSP client build procedure.

To deploy this client:
        Copy mathJsp.War into the deployment
        directory for your JSP server.
$
```

To deploy the JSP, copy the mathJsp.War file to a web server servlet deployment directory. For example, if you have Tomcat on OpenVMS, the command may look like this:

```
$ copy mathJSP.war sys$common:[apache.jakarta.tomcat.webapps]
```

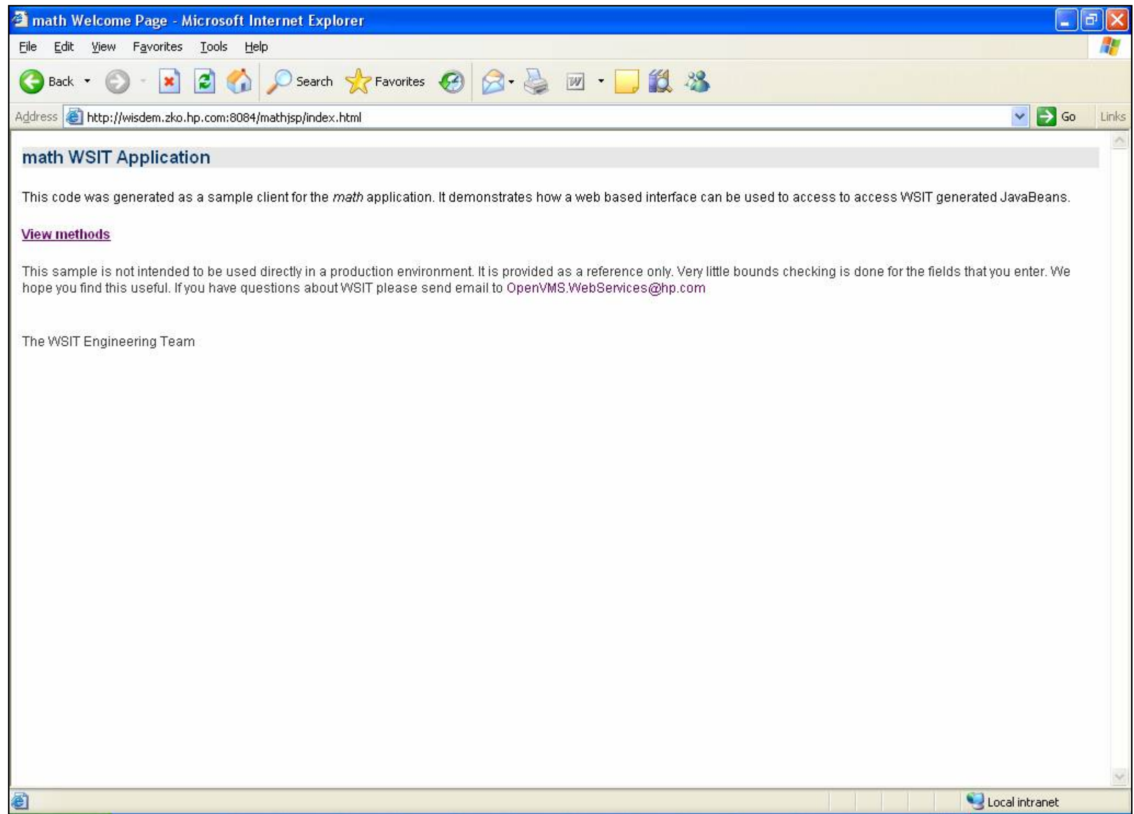
Once the war file has been copied, you can view the JSP pages by using a URL similar to the one shown below. (Replace yourwebserver.hp.com with the actual name of your web server.) By default, Tomcat listens on port 8080. If the system manger changed the port number, replace 8080 with the new number.

```
http://yourwebserver.hp.com:8080/mathjsp/index.html
```

The following screen captures illustrate the JSP sample client calling the C Math application.

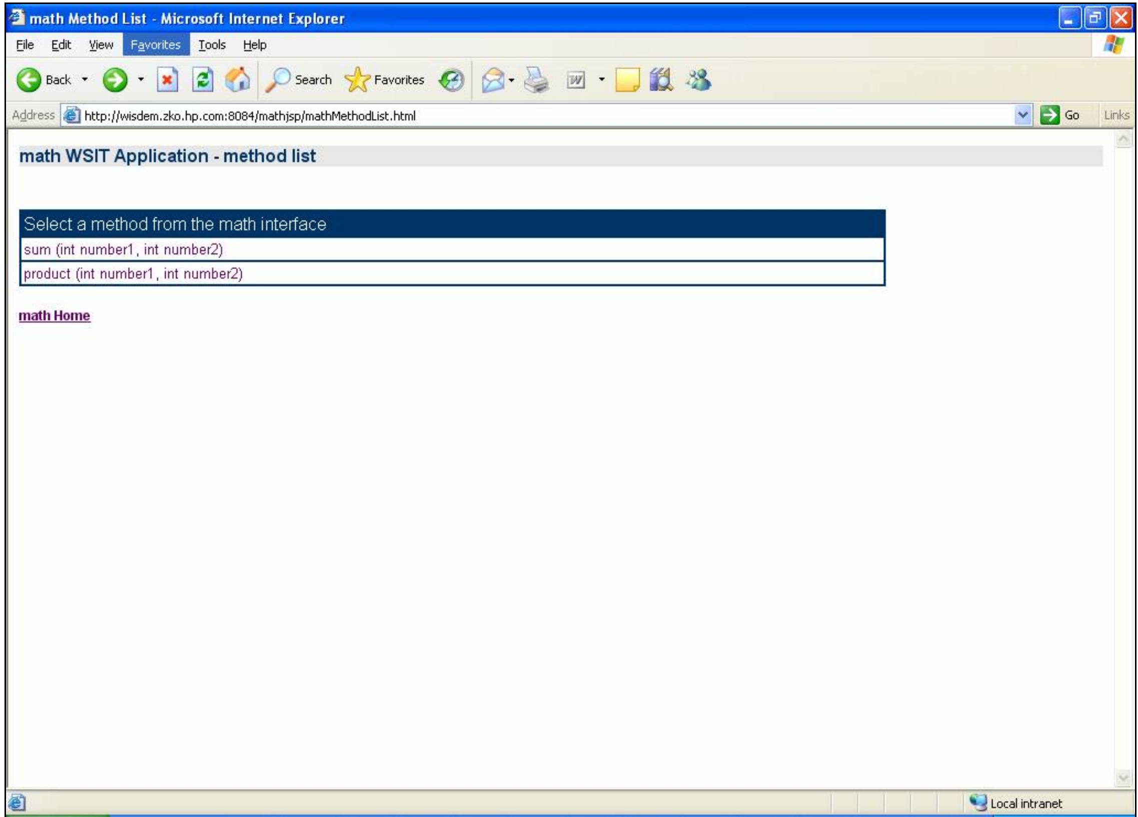
© Copyright 2006 Hewlett-Packard Development Company, L.P

Web Page 1: The mathJSP Application Homepage

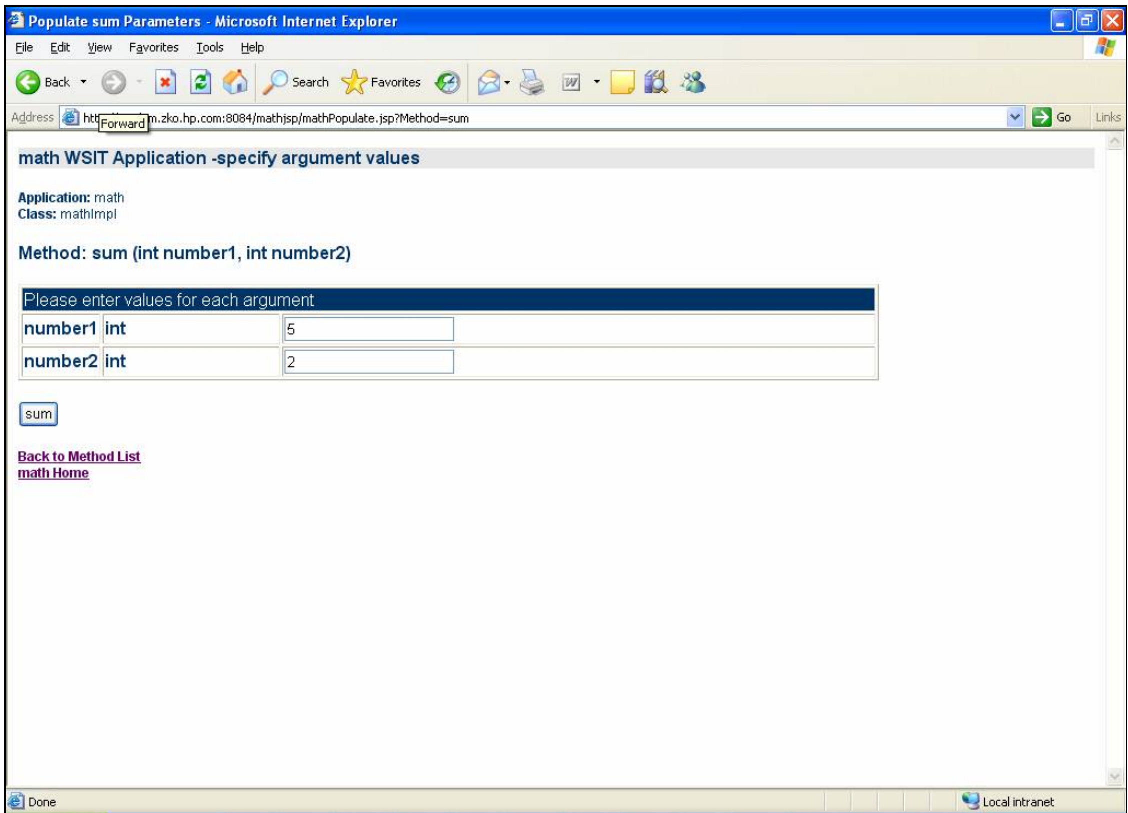


© Copyright 2006 Hewlett-Packard Development Company, L.P

Web Page 2: The MathJSP Application Methods

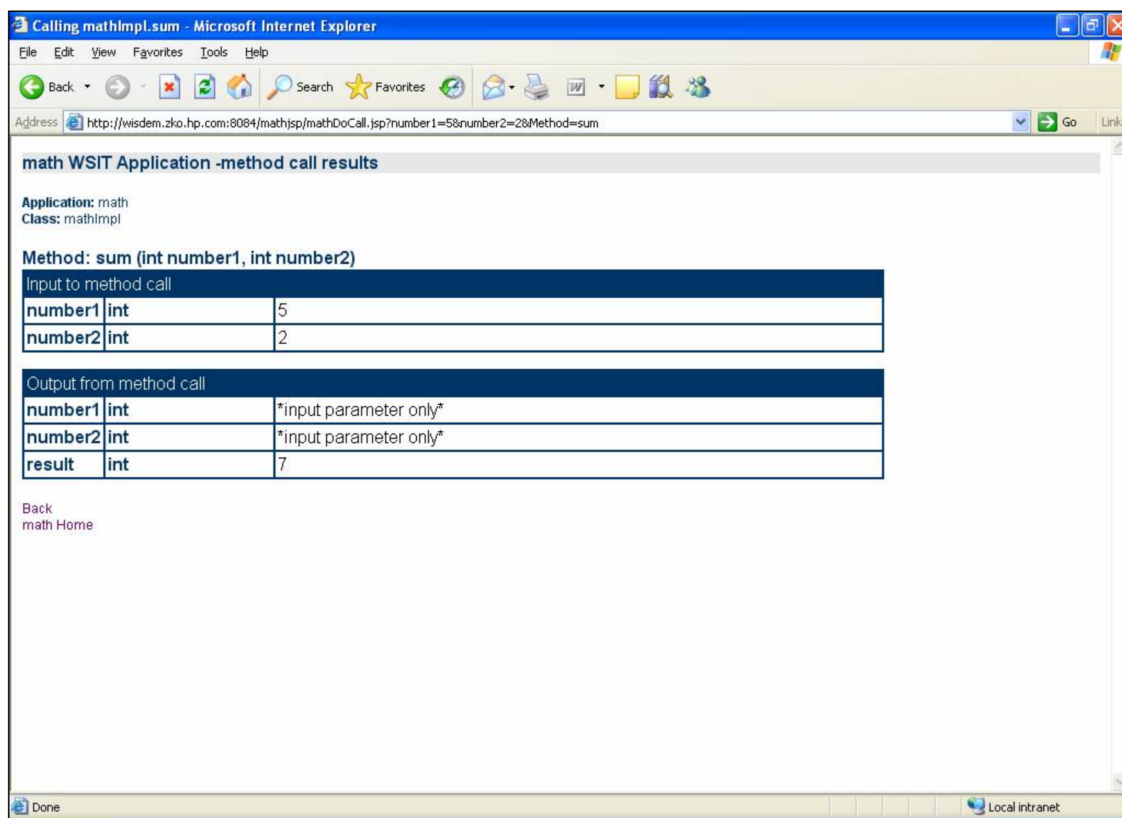


Web Page 3: The MathJSP Application Method *sum*



© Copyright 2006 Hewlett-Packard Development Company, L.P

Web Page 4: The MathJSP Application Method *sum* results



Deploying the Application Inproc / Outproc

There are two ways in which you can deploy your application using WSIT: **in-process** deployment and **out-of-process** deployment.

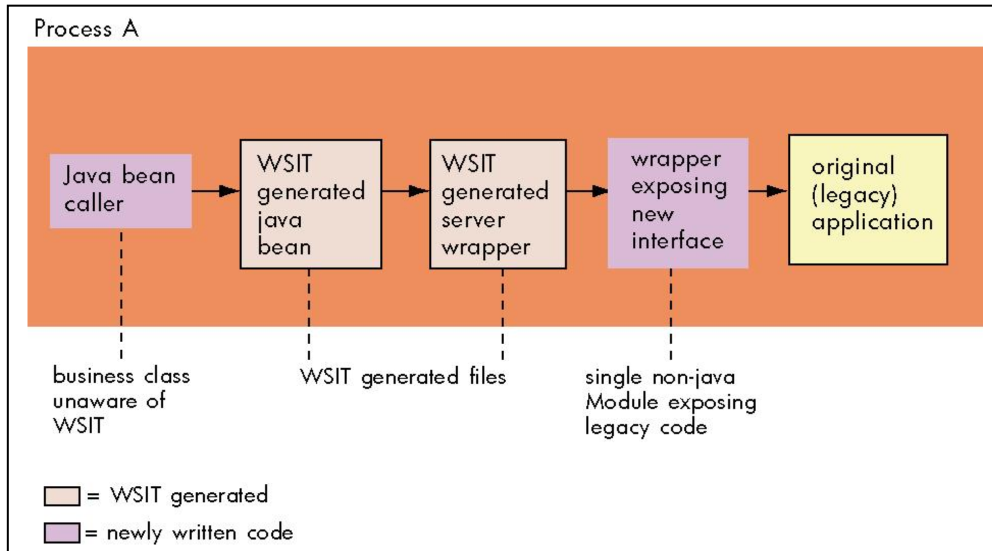
In the section Use Scenarios, we looked at a few of the different types of clients which can be used to access the WSIT wrapped application. These clients may interact with the application in different ways and may dictate the application's deployment settings.

For example, a JSP client will be deployed in a web server. The web server may accept multiple concurrent browser requests and execute each request on a separate thread. Many older OpenVMS applications were written with the assumption that they would never have more than one client using its services at a single time. In this scenario each client would need a private copy of the application to ensure that other clients do not interfere with its work. This can be accomplished by using out-of-process deployment.

In-Process Deployment

In-process deployment occurs when the application and the client are called from the same process, as illustrated in the following diagram.

© Copyright 2006 Hewlett-Packard Development Company, L.P



There are advantages and disadvantages to using in-process deployment.

Pros: Fastest execution time. No overhead added by the WSIT runtime.

Cons: A crash will bring down both client and server applications.

In-process deployment provides the fastest execution time, but it requires that the developer ensure that the client calling the wrapped application does not establish an environment in which the application will fail.

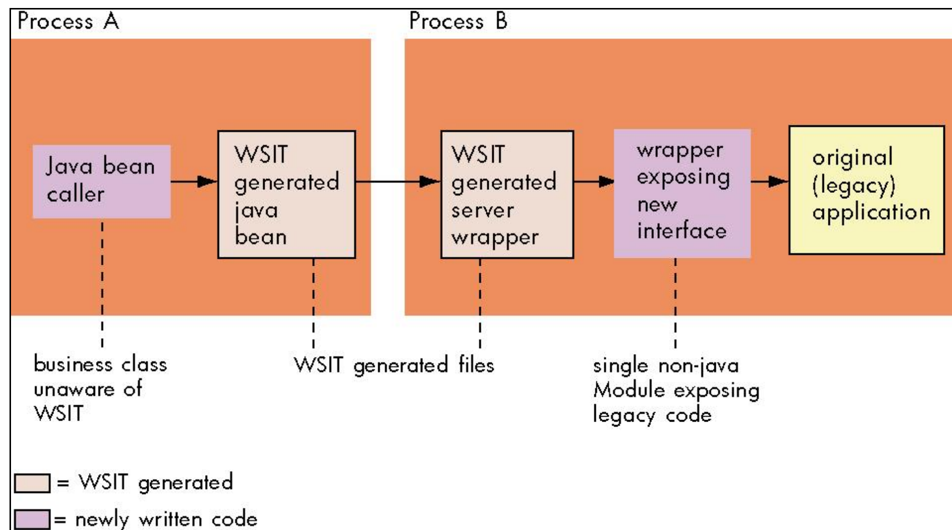
If you do not specify out-of-process deployment settings (described in the following sections), **your application will run in-process by default.**

Out-of-Process Deployment

As mentioned in Use Scenarios, the WSIT wrapper can be called from a variety of different technologies. Sometimes it may be desirable for the original application to run in a separate process than the calling technology. For example, a web server may allow multiple requests to be sent to the wrapped application. Each request may be running on a separate thread. If the original application was not designed to run in a multithreaded environment you can ensure that each request has its own private instance of the original application by running it out-of-process.

Out-of-process deployment occurs when the client and application are run in different processes, as illustrated in the following diagram. The WSIT runtime environment manages the interaction between the two processes. You can customize this environment by modifying a deployment descriptor file.

© Copyright 2006 Hewlett-Packard Development Company, L.P



There are advantages and disadvantages to using out-of-process deployment.

Pros: Typically scales better than in-process deployments. Allows the use of the WSIT runtime deployment properties.

Cons: Adds complexity and overhead to every call.

Most older applications benefit from using an out-of-process deployment to avoid complex issues that result from mixing older and newer environments.

Migrating from Other Products

BridgeWorks users have the ability to migrate their applications to WSIT. A tool is provided that generates a WSIT XML IDL file from a BridgeWorks connection description. This tool is named BWX2IDL and can be downloaded from the WSIT download site. See http://hp.com/products/openvms/wsit for more information.

Summary

The Web Services Integration Toolkit (WSIT) focuses on providing Java wrappers for non-Java applications. The Java wrappers can be used from a wide rage of technologies to provide both local and remote access.

Getting started with WSIT is easy. Simply pick one of the many WSIT samples and use the tools provided. In a few steps you can be calling a program written in C, BASIC, COBOl, FORTRAN or ACMS.

For more information

For more information about integrating OpenVMS applications, contact the author.

© Copyright 2006 Hewlett-Packard Development Company, L.P

Appendix A: Contents of math.xml File Generated from obj2idl Tool

```
<?xml version="1.0" encoding="UTF-8"?>
<OpenVMSInterface
       xmlns="hp/openvms/integration"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="hp/openvms/integration openvms-integration.xsd"
       ModuleName="DISK$:[VTJ]math.OBJ"
       Language="C89">
       <Primitives>
         <Primitive Name = "unsigned int"
                    Size = "4"
                    VMSDataType = "DSC$K_DTYPE_LU"/>
         <Primitive Name = "signed int"
                    Size = "4"
                    VMSDataType = "DSC$K_DTYPE_L"/>
       </Primitives>
       <Routines>
         <Routine  Name = "sum"
                   ReturnType = "unsigned int">
                   <Parameter Name = "number1"
                              Type = "signed int"
                              PassingMechanism = "Value"
                              Usage = "IN"/>
                   <Parameter Name = "number2"
                              Type = "signed int"
                              PassingMechanism = "Value"
                              Usage = "IN"/>
         </Routine>
         <Routine  Name = "product"
                   ReturnType = "unsigned int">
                   <Parameter Name = "number1"
                              Type = "signed int"
                              PassingMechanism = "Value"
                              Usage = "IN"/>
                   <Parameter Name = "number2"
                              Type = "signed int"
                              PassingMechanism = "Value"
                              Usage = "IN"/>
         </Routine>
       </Routines>
</OpenVMSInterface>
```

© Copyright 2006 Hewlett-Packard Development Company, L.P