Migrating to the Web Services Integration Toolkit from BridgeWorks

Phil Hudson
Software Engineer
OpenVMS Systems Software Division

Overview

BridgeWorks (BWX) has grown from a fledgling utility into a robust product that supports a large assortment of environments and offers a number of high-end features. During the past six years, BWX has gained a user base that has enjoyed the connectivity opportunities offered to existing applications residing on the OpenVMS platform.

With the rise in popularity of Web Services as a connection paradigm within the industry, BridgeWorks is being superseded by a product called the Web Services Integration Toolkit for OpenVMS (WSIT). WSIT shares many of the underlining technologies and uses many of the core components that were originally written for BridgeWorks. This gives WSIT a solid foundation on which to build, and provides the customer a certain level of knowledge reuse when working with both products.

BridgeWorks users may consider migrating their applications to use the Web Services Integration Toolkit for the following reasons:

- New features and enhancements will be added to WSIT.
- BridgeWorks will be maintained on OpenVMS Alpha but will not be ported to OpenVMS Integrity servers.
- Bridgeworks users will be familiar with WSIT because it encapsulates much of the BWX infrastructure.
- WSIT does not attempt to provide 1-1 support for each BWX feature, but it does include support for many of the most popular features.
- In most cases, migrating from BridgeWorks to WSIT is not difficult.

The model and focus that guided the development of WSIT was different than that of BWX. Because of this, there are differences between the two products that must be taken into consideration before attempting to migrate an application from BWX to WSIT.

This document describes those differences, along with the options and tools available to those who want to migrate from BWX to WSIT. Depending on the type of application that you are looking to migrate and which features within BWX you are currently using, there are various choices available to you. Making the right choices will make the migration process smoother.

Common Development Steps

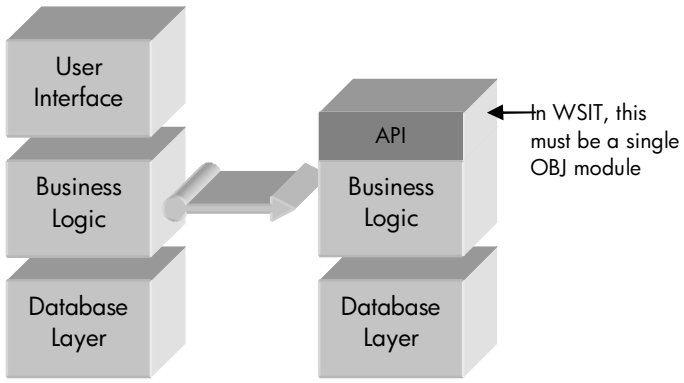
Whether you are working with BWX or WSIT, the general process of wrapping an application and turning it into a reusable service is the same. It is a four step process, where the steps are as follows:

1. Prepare the application to be wrapped.
2. Describe the application interface to the wrapping tool.
3. Run the code generator tool in order to create the newly packaged application.
4. Write a client for the newly created interface.

Although BWX and WSIT have a similar process, a closer look at the individual steps within this process shows some differences. It is with these differences where the majority of the migration effort will need to be focused.

Step 1: Application Preparation

In both BWX and WSIT, the task of preparing the application so that it can be wrapped means to make the application callable and to conform to the application model dictated by the wrapping tool. For instance, both BWX and WSIT are API (business logic) level wrapping tools, which means that both require the removal of terminal- or GUI-based I/O. Therefore, I/O should not be a concern when migrating from BWX to WSIT because the user I/O will have been previously removed.



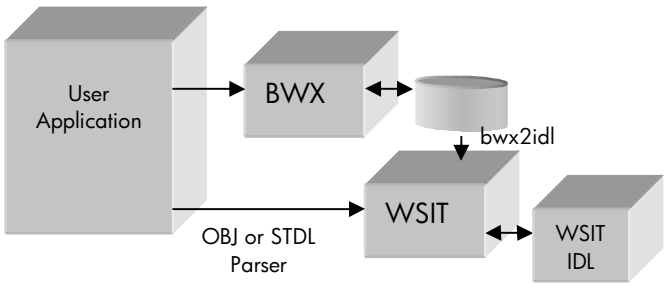
At this point BWX and WSIT differ in their model for creating reusable services. With WSIT, the original application is exposed through one or more interface objects that are written by the user. These interfaces are written in the original application language, such as C or COBOL. The user compiles each interface into an object module (obj) and WSIT generates a JavaBean wrapper with the same interface. Each user written interface has a 1-1 relationship with a generated JavaBean. If the WSIT user wants to expose multiple interfaces, there are two options as follows.

1) Write multiple interfaces and generate a matching JavaBean for each interface.
2) Expose a single interface and generate a single JavaBean. Then write, in Java, the individual interfaces that will be exposed to their callers.

(See *Migrating 3GL Applications* for more information.)

Step 2: Describing the Application Interface

After the application has been modified so that it is callable, the interface into this application must be described to the wrapper tool. BWX uses either an ANA or STDL parser to automatically import the exposed application interface information. It then stores this information within a relational database for later modification and code generation. WSIT uses either an object file debug record parser (Integrity servers only) or STDL parser to import this interface description. It then stores the description within a WSIT specific Interface Definition Language (IDL) file, in XML format, for later modification and generation. Because BWX users have already defined their interfaces, a tool is provided that can generate a WSIT IDL XML file directly from the BWX database. (See either *Migrating 3GL Applications or Migrating ACMS Applications* for more information.)



Step 3: Wrapper Generation

The BWX and WSIT code generators provide similar functionality in that they read the description of a previously defined API and generate the code to wrap that API. However, the individual tools are very different in look and feel. (See the WSIT Generator (idl2code) documentation in the WSIT Developer's Guide for more information.) Assuming that you have the API description in WSIT IDL format, then there is no defined migration task in this step except to learn the WSIT Generator.

Step 4: Create Client

After the generated code is built, both BWX and WSIT leave it up to you to create a client that will call into the new Java based interface. Although BWX and WSIT generate very similar interfaces, if you are migrating an existing BWX based client to use WSIT, there are some differences that you must handle.

- The Bwx* holder classes (ex: BwxLong, BwxDecimal, …), used for in/out parameters have been replaced with Web Services-friendly Holder classes (LongHolder, IntHolder, …). Holder classes provide similar functionality and offer the added advantage that they are used by current Web service environments. The migration is straight-forward; instead of each class having a setValue() and getValue() method as in BWX, each WS Holder class has a simple 'value' data member to store and retrieve the held value.
- In WSIT, the BwxException exception class has been replaced by WsiException. Also note that all WSIT throwable exceptions are derived from WsiException, which in turn is derived from RemoteException. This makes all exceptions suitable for throwing within a Web service environment. (See the wsirtl Javadocs in wsi$root:[docs.runtime] for more information.)
- In WSIT, structures that are passed as in/out parameters must be packaged within a StructureHolder class. This makes structures consistent with the other datatypes and with the way it works within Web service environments. Structures do not require a holder class within BWX.
- In WSIT, octawords are mapped to the BigInteger class so that it can accommodate the entire octaword value. In BWX, octawords were only mapped to Java longs, limiting their values.
- WSIT attempts to follow the naming conventions used by the Web service environments. Because of this, some of the names within the WSIT generated interface may be cased differently.

In general, the migration of a client from BWX to WSIT is a mechanical process and should not require any fundamental code changes.

Migrating ACMS Applications

ACMS applications are the easier of the two supported application types when migrating from BWX to WSIT. The only "*Application Preparation*" concern for WSIT, and BWX, is the forms management (user interface) integration support built into ACMS. As mentioned previously, all user interface code must be removed from the application being wrapped, and this holds true for ACMS applications. However, because this is a migration from BWX to WSIT, it can be assumed that this code has been previously removed.

ACMS STDL files, which are generated by the ACMS application compiler, provide a complete description of the ACMS applications to be wrapped. Because of this, there are two equally reasonable approaches that you can take in "*Describing the Application Interface*" to WSIT when migrating from BWX. The first and most desirable approach is to ignore the BWX description of this application, and use the WSIT stdl2idl utility to import the STDL file directly. This would be identical to wrapping a fresh ACMS application. The two benefits of this method are as follows:

1. Because STDL files are complete descriptions, the created WSIT IDL file requires no additions or changes.
2. This method avoids needing access to the Windows machine where BWX is installed.

A less desirable method, if directly importing the STDL file is not an option, is to import this information from the BWX database. You will need access to the Windows machine that contains the BWX database that houses the description of this application. You can then run the supplied bwx2idl utility on that Windows platform to extract the definition directly from the BWX database, creating a WSIT IDL file. This file must be copied back to the OpenVMS platform for further use. (See *The BWX2IDL Utility* for more information.)

The benefit of this method is that BWX maintains a very complete description of the ACMS application being wrapped, including any modifications and fixes that were made to the description after its original creation. Because of this, the generated WSIT IDL file requires no changes.

Pitfalls and Issues

Due to the structure of an ACMS application, there are very few differences between BWX and WSIT that are specific to ACMS. One difference is that WSIT can only wrap a single ACMS application within a WSIT application. If the BWX Connection that you are migrating contains more than one ACMS application, you will need to split each of the embedded ACMS applications into its own WSIT application. You can then write an encapsulating JavaBean that provides a single interface into the WSIT-generated JavaBeans that front-end these applications. Another, less desirable, option is to rework the ACMS applications so that all of the exposed tasks are encapsulated within a single ACMS application. (For more differences, refer to *General Differences* below.)

Migrating 3GL Applications

Migrating 3GL applications from BWX to WSIT can be more of a challenge than ACMS applications, depending on the application's program structure and features being used.

The primary "*Application Preparation*" work will require you to determine if you want to define the new application interfaces in Java or in the original programming language. If the new interfaces are to be written in Java, then a single object file must be created to expose the features of the original application. If the new interfaces are to be written in the non-Java language, then a single object module must be created for each interface. See the options in the "*Pitfalls and Issues*" section below.

As with ACMS applications, there are two approaches to "*Describing the Application Interface*" to WSIT when migrating from BWX. The first approach is to act as though the BWX description did not exist, and to parse the information directly from the debug records within the object file using the

supplied obj2idl utility. This has the benefit that everything can be done directly on the OpenVMS for Integrity server. However, this approach has two key drawbacks:

1. The obj2idl utility only works with object files produced on the IA64 platform. Alpha users would need to write the IDL XML by hand or generate it on an Integrity server
2. The contents of the object file that obj2idl reads may not contain all of the required information about the interface. In this case, you would need to edit the XML file to add the missing information. (Note that the utility knows when the information is missing or questionable and flags these entries within the IDL file.)

Because of these limitations, it may be preferable to import this information from BWX when migrating 3GL based applications. In this second approach, you will need access to the Windows machine that contains the BWX database that houses the description of this application. You can then run the supplied bwx2idl utility on that Windows platform to extract the definition directly from the BWX database, creating a WSIT IDL file. This file must be copied back to the OpenVMS platform for further use. (See *The BWX2IDL Utility* for more information.)

The benefit of this method is that BWX maintains a very complete description of the 3GL application being wrapped, including any modifications and fixes that were made to the description after its original creation. Because of this, the generated WSIT IDL file requires no changes.

Pitfalls and Issues

Depending on the program structure of the application being wrapped, there are a number of potential issues. The primary one regarding the WSIT application interface model has already been discussed.

Other pitfalls and issues include:

1. The interface object file written by the developer must be linked into the original application along with the WSIT generated server wrapper files. This may be achieved by including the WSIT generated files into any existing build procedure that you may already have in place. Alternatively, the WSIT generated server build procedure can be modified to link with the other object files, object libraries, and sharable images required for the application. (Note that it may be easier to include the WSIT generated files into any existing build procedure that you may already have in place.)
2. WSIT does not support special link command line options, such as those required by an Oracle based application. If this is needed, then it can be changed afterward within the generated build and option files. (This is similar to the issue above.)
3. Within a single connection, BWX supports creating multiple interfaces into the same user application, making it look like different applications to different users. Similar functionality can be accomplished within WSIT in one of two ways.
    - Writing multiple interfaces and generating a matching JavaBean for each interface.
    - Exposing a single interface and generating a single JavaBean. Then writing, in Java, the interfaces to be exposed to their callers.

(For more differences, refer to *General Differences* below.)

Unsupported BWX Applications

There are two application types that are supported by BWX but are not addressed by WSIT. They are the DCL procedure and the ASCII file. Although these are not directly supported by WSIT, both application types can be migrated with a small amount of manual coding.

Working with Wrapped Files

For example, the BWX support for wrapping files was nothing more than an open, read, and close of the file within the server. It is easy enough to write your own file read routine that replicates BWX's routine. You can then wrap it along with the other routines being exposed by your application. Further, if you need to hide the filename of the file being accessed, you can create individualized 'get file' methods within the generated JavaBean interface. (This is similar to how BWX implements this functionality.)

Working with DCL procedures

The BWX support for DCL procedures consists of a core routine that creates a subprocess and runs the specified DCL procedure within it. During this time, it captures all of the subprocess output within a mailbox and returns it as a string. Although DCL wrapping is more complex than file wrapping, the same process will work. A single routine can be written within the application that knows how to start, run, and capture the output of DCL procedures. This routine can then be wrapped along with the other exposed application routines. Finally, as in file wrapping, if you want to hide the name of the DCL procedure being run, individualized 'run procedure' methods can be added to the generated JavaBean. (This is similar to how BWX implements this functionality.)

General Differences

Your migration from BWX to WSIT is highly dependant on the type of application that is being wrapped, as well as which BWX features that you are currently using. Because of this, each general type of application is discussed in detail within its own section above. However, there are some general statements about migrating from BWX to WSIT that apply to all application types.

JavaBean Interface Only

WSIT always generates a JavaBean interface into the wrapped application. Although BWX allowed the option to generate a COM based interface, this support was limited, and not highly used by customers. BWX also offered the ability to generate a J2EE compliant session bean (EJB) interface. If this is still required, it is easy enough to write a session bean that encapsulates the generated JavaBean interface. However, in reality, the only feature that the EJB interface offered over the generated JavaBean interface was support for transactions, which is not a highly used feature.

Out-of-Process Across Machines

WSIT and BWX differ in support for out-of-process (distributed) wrapping. Although BWX allowed the Middle Component (interface) and the Server Component (user's application) to run in two different processes on the same machine or across two different machines, WSIT only supports out-of-process communication across two different processes on the same machine. Note that due to its better applicability to this environment, WSIT uses ICC exclusively for this communication.

If this multi-machine configuration is still required after migrating to WSIT, there are a number of solutions available, including:

1. Serving up the generated interface as a Web service using the SOAP Toolkit V2 for OpenVMS.
2. Serving up the generated interface as a Java servlet within a Tomcat environment.
3. Making the generated interface remotely available within a J2EE environment such as BEA's WebLogic Server.
4. Making the generated interface remotely callable using RMI.

The bottom line is that the provided JavaBean interface offers great flexibility and tool availability in how you can integrate in this new component.

No Support for Transactions

Because XA based transactions was not a highly used feature within BWX, it was decided not to support this feature within WSIT at this time. If you require Transaction support, this can be achieved by writing a little code outside of the generated JavaBean. The general steps are as follows:

1. Write a module within your application that exposes the XA Gateway API. (See the XA Gateway documentation for more information.)
2. Expose and wrap these routines along with the other routines within your application.
3. After using WSIT to generate the JavaBean interface, write an EJB or other class* that encapsulates this JavaBean. (*This class must implement the XAResource interface, making it directly callable by the Transaction Manager.)
4. Call the appropriate XA Gateway routines from within the XAResource class. (You should find this to be a 1-to-1 mapping of XAResource methods to XA Gateway routine calls.)

The BWX2IDL Utility

If you decide that the correct approach is to import the information directly from the BWX Database, then bwx2idl is the tool to use. This tool is written in Java and packaged within the jar file named bwx2idl.jar. The utility must be run on the Windows machine where the BWX database is installed. It uses the JDBC to ODBC Bridge to read the application description directly, and therefore requires that the ODBC DSN, BWXData_2_03, be correctly set up. (If this DSN does not exist, then BWX was not correctly installed on this machine.)

When successfully run, the bwx2idl utility generates three files, as follows.

- <application_name>-idl.xml

  This file contains the WSIT IDL description of the imported application definition.

- <application_name>.wsi

  This file contains the runtime configuration settings found for this application within the BWX Database.

- <application-name>.log

  This file contains a log of the import process. It identifies the information that imported cleanly, as well as flags the information that will require further migration work.

Command Options

Assuming that the classpath has been previously defined, the command line has the following format:

```
C\> java com.hp.wsi.bwx2idl [Option 1] … [Option n]
       ( Option = [switch [value]] )
```

If the bwx2idl utility is run without any options, then the list of Connections available within the BWX database will be displayed.

The following table shows the options available to the bwx2idl command line.

Option	Description
-c <Connection Name>	Name of the Connection to extract from BridgeWorks. If this option isn't specified, a list of available connections will be displayed by the utility.
-o <WSIT IDL Directory>	Output directory in which to generate the XML IDL files.
-j <Jar file specification>	Full file specification for the bwx2idl utility jar file. The utility requires a reference to its own jar file. In most cases the utility can determine this automatically. Because of this, it only needs to be specified if the tool prints a message requesting it.
-h	Instructs the utility to display a list of these options

The bwx2idl utility can be downloaded from the WSIT download page.

Summary

Remember, even if a feature isn't directly supported by WSIT, you can still successfully migrate from BWX. There are solutions available for each possible hurdle you may encounter during the migration process from BWX to WSIT.

For more information

For more information, please refer to the following resources.

- The WSIT documentation, which is installed on OpenVMS as part of the WSIT kit. After installation, it can be found in WSI$ROOT:[docs].
- The BridgeWorks online help, which is installed on your PC as part of BridgeWorks.
- The following websites:

  http://www.hp.com/go/bridgeworks

  http://www.hp.com/products/openvms/wsit