

Using Web services with WebSphere Process Server

[Anh-Khoa Phan](#)

Software Engineer
Rochester, MN

[Eric Herness](#)

WebSphere Business Integration Chief Architect
Rochester, MN

December, 2005

© Copyright International Business Machines Corporation 2005. All rights reserved.

This article explains the key role of Web Service Definition Language and XML in IBM WebSphere Process Server and shows you how to use WebSphere Process Server to leverage Web service technologies.

Introduction.....	2
Hello World example.....	2
XML technology usage in WebSphere Process Server	3
WSDL	3
XSD.....	6
Web services supported export development patterns.....	7
Top-down endpoint development	7
Existing endpoint development.....	8
Web services supported import development patterns	9
Importing Web Services SCA exports.....	9
Importing other Web services.....	11
WebSphere Process Server and WebSphere Integration Developer limitations and restrictions.....	12
Limitation: Manipulation of types containing “anys” will be with Java	13
Limitation: WebSphere Process Server components can only reference components described by WSDL interfaces	13
Limitation: Web service dependencies to artifacts in library projects are not refreshed after an update.....	13
Limitation: Changing the name of a Web services export requires binding regeneration	14
Restriction: Default WSDL generation pattern is doc-literal Wrapped.....	15
Limitation on types with no "targetNamespace" usage	16
Limitation: WSDL binding section not honored in v6	17
Conclusion	18
Resources	19
About the authors.....	19

Introduction

Web services is the de-facto standard for service integration. IBM® WebSphere® Process Server, built to solve integration problems, leverages Web services based technologies, including the extensive use of Web Service Definition Language (WSDL) to define component interfaces and XML Schema Definition Language (XSD) to define the structure of data flowing in and out of WebSphere Process Server.

A brief description of the sections included in this article follows:

- **XML technology in WebSphere Process Server** provides an overview of the role of XML in WebSphere Process Server as it relates to interfaces and interoperability.
- **Web services supported export development patterns** describes the patterns by which Service Components authored in WebSphere Integration Developer and run on WebSphere Process Server can be accessed via a standard Web Services binding.
- **Web services supported import development patterns** describes how you can leverage off-the-shelf Web services in an overall integration solution.
- **Known limitations of WebSphere Process Server and WebSphere Integration Developer 6.0.0.0** are described, with known workarounds presented as well.

Once you've absorbed the information in this article, you will understand the key role of WSDL and XML in WebSphere Process Server integration scenarios. You will also understand how to leverage existing services as well as add to the set of "services" that can be accessed by standards-based Web services clients.

Hello World example

To illustrate the concepts that we will introduce in this article, we start off by describing a simple Hello World example.

In this example we have a single Service Component Architecture (SCA) module containing a single component that is implemented in Java™, named HWComponent. This component has one export that has a Web services binding, named HWComponentExport. Here is the Assembly Diagram view (a WebSphere Integration Developer editor) for this module:

Figure 1. Assembly diagram (a WebSphere Integration Developer editor) for HWModule



Additionally, the HWComponent exposes an interface that is typed by WSDL and passes data described by XSD. Here's the Interface editor view (a WebSphere Integration Developer editor) of this WSDL, named HelloWorld:

Figure 2. Interface editor (a WebSphere Integration Developer editor) showing HelloWorld interface

	Name	Type
▼ sayHello		
Input(s)	reqMsg	HWMessage
Output(s)	isSuccessful	boolean

Here's the Business Object (BO) editor for the XSD, named HWMessage:

Figure 3. BO editor showing HWMessage BO

HWMessage	
message	string
sourceEmail	string

Further details of this example will be discussed where appropriate within each section.

XML technology usage in WebSphere Process Server

WSDL and XSD are major players in Web services – building upon the foundations of modeling provided by XML. WebSphere Process Server uses WSDL and XSD internally. This section provides the details you need to use these technologies.

Future sections cover details on "exporting a Web services interface" to a WebSphere Process Server based service and "importing" and using traditional Web services.

WSDL

WSDL is used extensively across WebSphere Process Server as the preferred document type to define interfaces that components expose. The Interface wizard creates interfaces that are defined using WSDL conforming to the style and use of document/literal wrapped, but can also consume other WSDL styles and use (for example, document/literal, rpc/literal, and rpc/encoded).

Here is a schema source view of the HelloWorld interface (created by the Interface wizard):

Listing 1. Source view of HelloWorld interface

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:bons1="http://HWModule/hw/data"
  xmlns:tns="http://HWModule/hw/intf/HelloWorld"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="HelloWorld"
  targetNamespace="http://HWModule/hw/intf/HelloWorld">
  <wsdl:types>
    <xsd:schema targetNamespace="http://HWModule/hw/intf/HelloWorld"
      xmlns:bons1="http://HWModule/hw/data"
      xmlns:tns="http://HWModule/hw/intf/HelloWorld"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import namespace="http://HWModule/hw/data"
        schemaLocation="../data/HWMessage.xsd" />
      <xsd:element name="sayHello">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="reqMsg" nillable="true"
              type="bons1:HWMessage" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="sayHelloResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="isSuccessful" nillable="true"
              type="xsd:boolean" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="sayHelloRequestMsg">
    <wsdl:part element="tns:sayHello" name="sayHelloParameters" />
  </wsdl:message>
  <wsdl:message name="sayHelloResponseMsg">
    <wsdl:part element="tns:sayHelloResponse" name="sayHelloResult" />
  </wsdl:message>
  <wsdl:portType name="HelloWorld">
    <wsdl:operation name="sayHello">
      <wsdl:input message="tns:sayHelloRequestMsg"
        name="sayHelloRequest" />
      <wsdl:output message="tns:sayHelloResponseMsg"
        name="sayHelloResponse" />
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>

```

Doc-literal wrapped style used for interfaces

The default style generated by the Interface wizard is doc-literal wrapped (DLW). This style is aligned with the direction of [WS-I](#), as it tries to group together standards for Web services interoperability.

Even though the Interface wizard generates only DLW interfaces, other styles of interfaces are supported as well including: doc/literal, rpc/encoded, and rpc/literal. This means that interfaces created with non-DLW styles will be honored when editing them within the Interface editor.

If you are not familiar with the differences between these style and the pros and cons of the different styles, here is a good article describing the differences and similarities between the styles – [Which style of WSDL should I use?](#)

XSD

XSD is used extensively across WebSphere Process Server as the preferred document type to define the data that components consume and return. This schema document is generated using the BO wizard in WebSphere Integration Developer. The default creation style of the BO wizard is to create one type per XSD file. This, however, does not restrict the viewing of an XSD file with multiple types in the BO editor.

Here is a schema source view of the HWMessage BO (created by the BO wizard):

Listing 2. Source view of HWMessage BO

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://HWModule/hw/data">
  <xsd:complexType name="HWMessage">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="message" type="xsd:string" />
      <xsd:element minOccurs="0" name="sourceEmail"
        type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Service Data Object ↔ BO

Data described by XSD is represented in memory by [Service Data Objects](#) (SDO). A brief description of SDO is:

As defined on the [Eclipse Modeling Framework](#) page, “SDO is a framework that simplifies and unifies data application development in a service-oriented architecture (SOA). It supports and integrates XML and incorporates J2EE patterns and best practices.”

SDO provides a standard API for manipulating data and the presents type mapping rules from XML to Java, and vice versa.

WebSphere Process Server goes beyond what’s described in present [SDO V1.0](#) and presents additional functionality that is coming in [SDO V2.0](#). This functionality is

presented in the BO Service operating on Business Objects (BO), which in essence are SDOs.

Web services supported export development patterns

The following patterns are supported in WebSphere Process Server for exposing components as Web services.

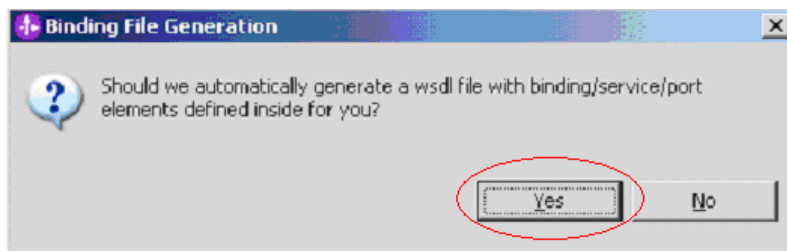
Top-down endpoint development

In this context, top-down endpoint development refers to the creation of a Web services export to expose a component to the Web with the following assumptions:

- The binding and service information for the endpoint is kept separately (in another generated WSDL) from the interface information (for example, portType). This new WSDL will be generated at the root of your Module project and will be named using the following naming convention: [exportName] + '_' + [portTypeWSDLName] + [Http || Jms] + '_Service', where Http or JMS refers to the protocol you chose your service to be exposed (that is, SOAP/HTTP or SOAP/JMS, respectively).
- The address of the endpoint follows the following naming convention: [moduleName] + 'Web/sca/' + [exportName].
- The style/use pattern for the endpoint is document/literal.

You create the binding WSDL after this dialog prompt (prompted during the creation of the Web services export):

Figure 4. Binding File Generator dialog window



Here's the snippet for the binding WSDL of our Hello World example:

Listing 3. Source view for HWComponentExport_HelloWorldHttp_Service binding WSDL file

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="HWComponentExport_HelloWorldHttp_Service"
  targetNamespace="http://HWModule/hw/intf/HelloWorld/Binding"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:Port_0="http://HWModule/hw/intf/HelloWorld"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:this="http://HWModule/hw/intf/HelloWorld/Binding"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:import namespace="http://HWModule/hw/intf/HelloWorld"
    location="hw/intf/HelloWorld.wsdl" />
  <wsdl:binding name="HWComponentExport_HelloWorldHttpBinding"
    type="Port_0:HelloWorld">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="sayHello">
      <soap:operation />
      <wsdl:input name="sayHelloRequest">
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output name="sayHelloResponse">
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="HWComponentExport_HelloWorldHttpService">
    <wsdl:port name="HWComponentExport_HelloWorldHttpPort"
      binding="this:HWComponentExport_HelloWorldHttpBinding">
      <soap:address
        location="http://localhost:9080/HWModuleWeb/sca/HWComponentExport" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Existing endpoint development

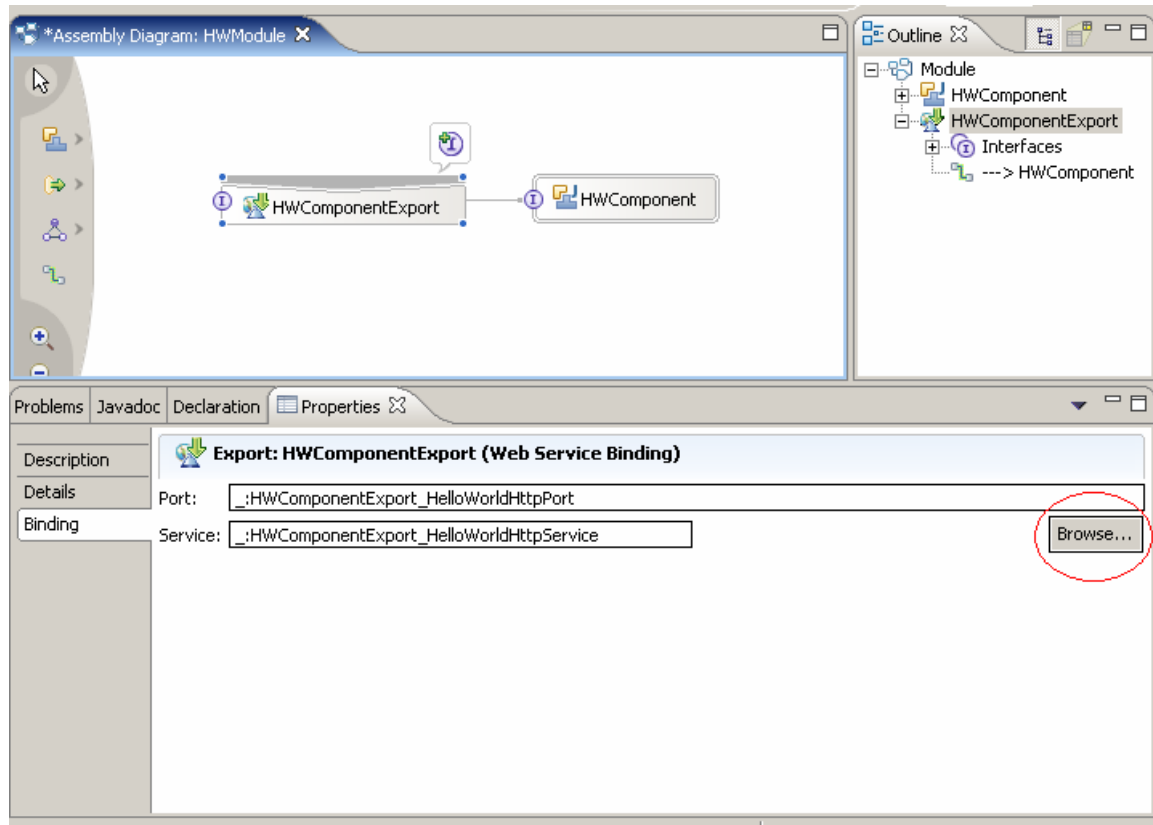
In this context, existing endpoint development refers to either of the following situations:

1. Re-implementing an existing Web services endpoint with SCA
2. Exposing Web services with other supported styles, for example, rpc/encoded and rpc/literal.

Both of these situations assume that an existing WSDL that contains the binding information for the Web service will be reused. You can create this WSDL using any other tools like the WSDL editor in Rational Application Developer

In this scenario, the prompt shown in the previous section for binding file generation should be answered with **No**, and the user will need to **Browse ...** to this existing WSDL file after creating the export to specify the binding information for the newly created endpoint:

Figure 5. Changing the binding for a Web services export



Web services supported import development patterns

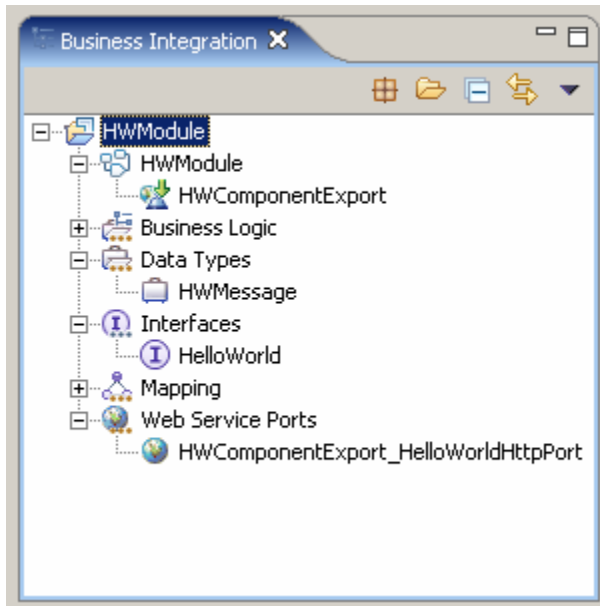
Reusing existing services exposed through the Web is a fundamental pattern in WebSphere Process Server and involves the creation of a Web services import to represent the imported Web service.

The following subsections provide details for the common patterns that customers will see when importing Web services using SCA.

Importing Web Services SCA exports

SCA components exposed by a Web services export can be reused in another module like any other ordinary, standard Web service. The only difference here is the tooling support for SCA exports (drag-and-drop [DND] function for creation of a corresponding import) through the use of the Business Integration view.

Figure 6. Business Integration view for HWModule



The DND function is limited in WebSphere Process Server to just the creation of the import and will not copy the required client files (that is, referenced WSDL and the XSD files) that are required. You will need to copy these over manually to the project that you're dropping the export into.

For example, in the Hello World example, we have a Web services export named HWComponentExport, which is dependent on the following files:

- HellWorldComponentExport1_HelloWorldInterfaceHttp_Service.wsdl – WSDL file containing the endpoint information for the Web service export.
- hw\intf\HelloWorld.wsdl – WSDL file describing the interface for our Web service, e.g. the portType definition.
- hw\data\HWMessage.xsd – XSD file describing the BO used by our Web service.

Hint: If you took a look at the corresponding EJB staging project under ejbModule/META-INF/wsdl, all the required files will be there. In our example it's all the files in the wsdl directory since we only have one Web services export in the HWModule module.

Listing 4. Source view of StockQuote WSDL file

```
</portType>
<binding name="StockQuotesSoap" type="s0:StockQuotesSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />
  <operation name="GetStockQuotes">
    <soap:operation
      soapAction="http://swanandmokashi.com/GetQuotes" style="document" />
    <input name="GetQuotesIn">
      <soap:body use="literal" />
    </input>
    <output name="GetQuotesOut">
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
<service name="StockQuotes">
  <port name="StockQuotesSoap" binding="s0:StockQuotesSoap">
    <soap:address
      location="http://www.swanandmokashi.com/HomePage/WebServices/StockQuotes.asmx" />
  </port>
</service>
</definitions>
```

Use the import from File System to import the WSDL file. After importing the WSDL file, you should see (in the Business Integration view) an entry under Web Service Ports:

Important: You should NOT “import” the Java classes (for example, service interfaces) from previous WSDL2Java since the type mappings in WebSphere Process Server Version 6.0 uses SDO, whereas in the past (for example, WebSphere Business Integration-Server Foundation 5.1.x) the mappings were to/from Java beans based on the JAX-RPC specification.

Figure 8. Business Integration view showing a Web service port



Now, you should be able to DND this StockQuotesSoap port onto the Assembly Diagram to create a Web services import file.

WebSphere Process Server and WebSphere Integration Developer limitations and restrictions

The set of restrictions and limitations below are assumed temporary for v6.0.0.0 unless explicitly noted.

Limitation: Manipulation of types containing “anys” will be with Java

WebSphere Integration Developer v6.0.0.0 will not support the manipulation of "any" types (for example, soapenc:anyType), but will enable the flow and assignment (that is, one can assign an element of type "any" to another variable of type "any") of these types. If manipulation is needed, you need to do it via Java code that is, Java snippets, Java custom maps, Java implementations, and so on.

Workaround

Use Java code to manipulate "any" types when needed using the following sample code:

Listing 5. Sample code from HowToSOAPArrayClient class in anySample.zip

```
// Invoke an operation that takes a SOAP array of strings
// A SOAP array is just a complex type containing an
// element named any with maxOccurs="unbounded"
{
    // Create the input DataObject and get the SDO sequence for
    // the any element
    DataObject arrayOfString = dataFactory.create("http://soapinterop.org/", "ArrayOfString");
    sequence sequence = arrayOfString.getSequence("any");

    // Get the SDO property for the sample element that we want
    // to use here to populate the sequence

    // we have defined this element in an XSD file, see
    // SampleElements.xsd
    DataObject sampleElements = dataFactory.create("http://sample/elements", "DocumentRoot");
    Property property = sampleElements.getType().getProperty("sampleStringElement");

    // Add the elements to the sequence
    sequence.add(property, "Hello");
    sequence.add(property, "World");
}
```

The entire code example can be found in the anySample.zip file.

Limitation: WebSphere Process Server components can only reference components described by WSDL interfaces

WebSphere Process Server components such as Process and Mediation components, can only reference (and wire to) other components that have WSDL interfaces. Components that use Java to describe their interface (in v6.0.0.0 only Java implementations can do this) cannot be wired from WebSphere Process Server components.

Workaround

Create a POJO that exposes a WSDL interface and calls out to the Java type component.

Limitation: Web service dependencies to artifacts in library projects are not refreshed after an update

The user may be operating with inconsistent artifacts used in Web services (export/import) scenarios if the Web services exposed WSDL depends on artifacts (for example, XSDs and/or other WSDLs) that are located in library projects.

If you update or change the library artifacts after creating the Web service import/export, the server will still be operating with the old artifacts. The server won't be in sync with what's in the workspace until the next full build of the workspace.

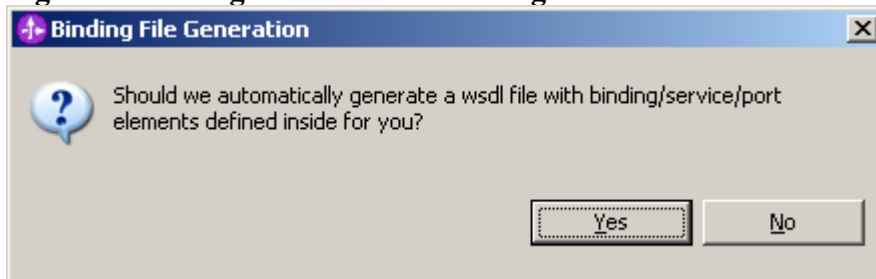
Workaround

After changing a library artifact that is used by a Web service export/import, manually execute a **project => clean** or **project => build All** action.

Limitation: Changing the name of a Web services export requires binding regeneration

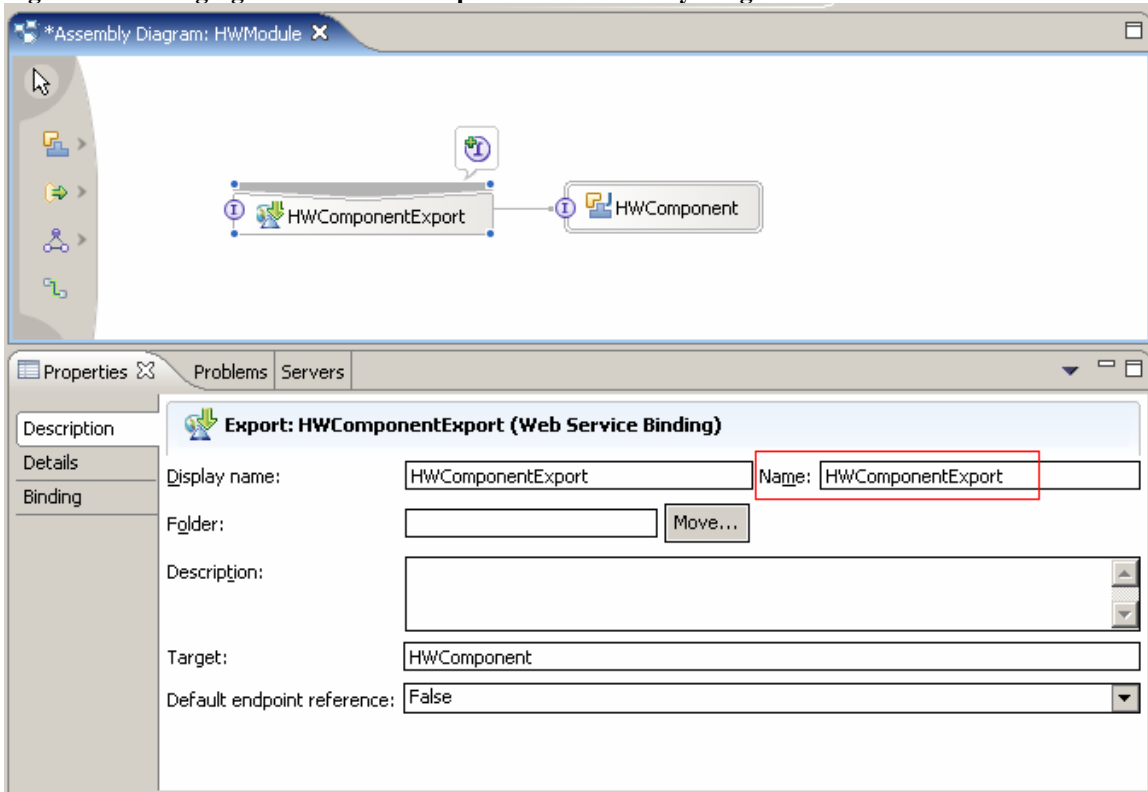
When you choose to create a Web services export for a component and choose **Yes** on the **Binding File Generation** dialog, a default name is given to the export comprising the name of the export's target plus "Export".

Figure 9. Binding File Generator dialog window



Do not change this name, because the J2EE artifacts will be inconsistent with the new name you provide.

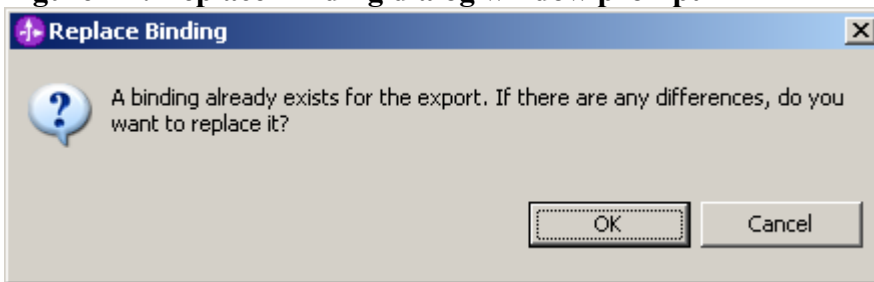
Figure 10 . Changing the name of an Export in the Assembly Diagram



Workaround

If, however, there's a requirement for you to change the Web service export name from its default, then the workaround is to regenerate the Web services export after the name change by right-clicking on the export (for example, HWComponentExport) and choosing **Replace Binding** and click **OK** when you see the following dialog popup:

Figure 11 . Replace Binding dialog window prompt



Restriction: Default WSDL generation pattern is doc-literal Wrapped

Currently, the Interface wizard is restricted to producing only interfaces that are doc-literal wrapped following the direction set forth by the [WS-I](#) body. The Web services export binding file generator also only generates to this pattern. WebSphere Integration Developer and WebSphere Process Server promote the usage of doc-lit wrapped WSDL

files and thus, for v6.0.0.0 generate only doc-literal wrapped WSDL files. If you wish to generate WSDL files that conform to other styles, (that is, doc/literal, rpc/encoded, and rpc/literal) then you will need to use other tools to do this WSDL file creation.

Tip: Other useful editors from RAD are listed in the appendix under “Other Useful Tools in WebSphere Integration Developer”

Limitation on types with no "targetNamespace" usage

Types that do not have a targetNamespace (that reside in library projects) will not be resolved in the WebSphere Test Environment (WTE) if you use the default **Run server with resources within the workspace** set property.

The use of no target namespaces is a pattern for type reuse and is expected to be a common pattern that customers will want to use. The runtime supports this, but there is a limitation in the WTE when using the default server setting of **Run server with resources with the workspace** though.

Workaround

The workaround is to change the server's setting to **Run server with resources on Server** from **Run server with resources with the workspace**.

Figure 12 . Changing a setting using the Server Overview

Server Overview

General
Specify the host name and other settings.

Server name:
Host name:
Runtime:

Server
Enter settings for the server.

WebSphere profile name:
Update server status interval (in milliseconds):

Server connection type and admin port

RMI (Better performance)
ORB bootstrap port: 2809

SOAP (More firewall compatible)
SOAP connector port:

Enable hot method replace in debug mode
 Enable universal test client
 Terminate server on workbench shutdown

Publishing
Modify the publishing settings.

Run server with resources within the workspace
 Run server with resources on Server

Enable automatic publishing
Publishing interval (in minutes):

Security

Network Deployment

Limitation: WSDL binding section not honored in v6

Binding information specified in the WSDL file is not honored in v6.0.0.0. For example, if you specified that a particular part be placed in the header of the SOAP envelope, then this will not be honored.

Listing 6. Source view showing the binding section of the StockQuote WSDL

```
<operation name="queryService">
  <soap:operation soapAction="queryService"
    style="document" />
  <input>
    <soap:body
      namespace="http://www.tempuri.org/MyService" parts="parameters"
      use="literal" />
    <soap:header message="tnsl:queryServiceSoapIn"
      namespace="http://www.tempuri.org/MyService" part="context"
      use="literal" />
  </input>
  <output>
    <soap:body
      namespace="http://www.tempuri.org/MyService"
      parts="queryServiceResult" use="literal" />
  </output>
  <fault name="fault">
    <soap:fault name="fault"
      namespace="http://www.tempuri.org/MyService" use="literal" />
  </fault>
</operation>
```

Workaround

An iFix is available and will be available only for the scenario where the parts in the input stanza are located in the same message definition meaning that the signature of the operation contains the parts specified here (for example, explicit headers).

Conclusion

Web services and XML play a key role in SCA and WebSphere Process Server. This article has outlined the ways in which WSDL and XML are observed and leveraged in the externals for the creation of the various kinds of SCA services that make up the core components of WebSphere Process Server. Also described is the role of Web services as a way to export a service, defined and implemented using SCA, and the way in which Web services, that already exist, are leveraged as part of an overall solution.

What wasn't explained is that many of the internal files for SCA metadata are also described via XML. Some of the SCA components implementation types are XML described as well. WSDL and XML are central to the WebSphere Process Server and key concepts to master as you begin to develop services and solutions on this platform.

Resources

- Java 2 Platform, Enterprise Edition (J2EE): <http://java.sun.com/j2ee/>
- IBM developerWorks Web services site: <http://www.ibm.com/developerworks/webservices/>
- WS-I Basic Profile v1.0: <http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html>
- SCA Homepage: <http://www.ibm.com/developerworks/library/specification/ws-sca/>
- SDO Homepage: <http://eclipse.org/emf/sdo.php>
- XML Schemas: Best Practices: <http://www.xfront.com/BestPracticesHomepage.html>
- WebSphere Process Server site: <http://www-306.ibm.com/software/integration/wps/>
- WebSphere Process Server Information Center: <http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp>
- WebSphere Business Integration zone: <http://www.ibm.com/developerworks/websphere/zones/businessintegration/>

About the authors

Anh-Khoa Phan is a Software Engineer at IBM, Rochester who currently works on WebSphere Business Integration. You can reach Anh-Khoa at anhkhoa@us.ibm.com.

Eric Herness is currently the Chief Architect for Websphere Business Integration and is from the IBM Rochester, MN, USA development lab. He is senior member of the WebSphere Foundation Architecture Board and a core member of the Software Group Architecture Board. Eric has been involved with product architecture and product development in object technology and distributed computing for over 15 years. You can reach Eric at herness@us.ibm.com.

Trademarks

- IBM, Rational, and WebSphere are trademarks or registered trademarks of IBM Corporation in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- Other company, product, and service names may be trademarks or service marks of others.

IBM copyright and trademark information: <http://www.ibm.com/legal/copytrade.phtml>