Wendy Conti

WebSphere Application Server V6.1 Web Services Problem Determination

Web services-related problems can occur when your application acts as a Web services client to a remote Web service or as a Web service that is accessed by external clients. Symptoms of a Web services problem would include:

- Web services errors and exceptions that appear in the JVM™ logs
- Unexpected Web services behavior
- Problems with Web services tooling, such as the Java2WSDL and WSDL2Java scripts
- Error or warning messages that begin with `WSWS`.

This paper covers problems that occur during development of Web services applications (WSDL2Java errors) and deployment of these applications to WebSphere® Application Server. It also covers problems that occur in the Web services engine.

It does not cover problems with Web services gateway, service integration bus, WSIF, or Web services security.

It is applicable to WebSphere Application Server on distributed platforms, i5/OS®, and z/OS®.

Introduction

WebSphere Application Server V6.1 provides extensive support for Web services standards. It can host both Web service client and provider applications. Problems can occur due to coding errors, invalid requests, deployment code generation, or the runtime configuration.

This activity covers the following types of Web services problems:

- Problems that occur during development
- Problems deploying Web services applications
- Problems that occur during runtime

If you believe that you have a Web services gateway, service integration bus, WSIF, or Web services security problem, go to “The next step” on page 47 for information about online resources and contacting IBM®.

Determine problem type

Problems can occur during development, deployment, or runtime of Web services applications.

Symptoms of a problem during development

Two IBM products are targeted for development for WebSphere Application Server applications: Rational® Application Developer and Application Server Toolkit.

Problems that occur during the development phase of Web service clients and providers for WebSphere Application Server are often a result of the use of the WSDL2Java tool. WSDL2Java is used for developing and deploying Java™-based Web services applications from a WSDL file — both client and provider. This tool can be used stand-alone. It is also invoked by the development products and WebSphere deploy tools when working with Web services applications.

Symptoms of a WSDL2Java problem include error or warning messages with the WSWS prefix, or generated code that does not compile.

“Problems that occur during development” on page 5 focuses on problems that occur during the use of WSDL2Java when developing Web services applications.

Symptoms of a problem during deployment

Web services applications that are deployed to WebSphere Application Server require special deployment code to be generated. This code can be generated in

a variety of ways, for example, with the WSDL2Java tool, by Rational Application Developer, or by selecting an option during installation. However, the basis for each of these methods is, once again, the WSDL2Java tool.

Symptoms of a Web services deployment problem generally appear in the SystemOut log or, if using the **wsdeploy** command, in the command window. The first indication is a `WSWS0038E: Error from Web services deploy tool` message.

"Problems deploying Web services applications" on page 16 focuses on problems that occur when deploying a Web service client or provider application to WebSphere Application Server.

Symptoms of a problem during runtime

Problems that occur when running Web service clients or providers can happen for many reasons. Examples include (but are not limited to) an improper call to the provider, application code errors, runtime configuration errors, or a change in runtime conditions (for example, a provider URL that has changed).

Symptoms of a Web services runtime problem can appear on the client or the provider, either in a log or on the client display. How the problem presents will vary depending on factors such as where the problem occurred (client or provider), the type of client, how the application handles errors, and so on. Primary symptoms include messages with a WSWS prefix or unexpected results in the application.

"Problems that occur during runtime" on page 24 focuses on Web services problems that occur in WebSphere Application Server, where the application in WebSphere Application Server is the Web service provider or Web service client.

WSDL2Java

A WSDL file acts as a contract between client and provider. Client and provider agree to send SOAP messages structured according to what the WSDL describes. WSDL2Java is used for developing Java-based Web service client and provider applications from a WSDL file.

Web service provider applications can run in the EJB™ and Web container. Web service client applications can also run in the EJB and Web container, as well as the J2EE™ client container. These types of clients are commonly known as *managed clients*. A Web service client can also be a standalone Java application; this is commonly known as an *unmanaged client.*

But whether client or provider and whether it runs in a container or not, any application developed using WSDL2Java should comply with the JAX-RPC

specification. It is when applications violate JAX-RPC that unexpected results occur.

WSDL2Java can be invoked directly by the developer. It runs behind the scenes in Rational Application Developer and the Application Server Toolkit when you develop Web services applications and prepare them for deployment. It is also used when you invoke WebSphere Application Server deploy tools for a Web services application.

WSDL2Java generates the following development code:

- The service endpoint interface
- Java types that map to JAX-RPC supported schema types defined/ referenced in the WSDL
- Exceptions that map to WSDL faults
- JAX-RPC mapping file that describes how the Java and XML are mapped to each other
- webservices.xml - the Web service provider deployment descriptor file
- ejb-jar.xml - the deployment descriptor for a Web service client running in the EJB container
- web.xml - the deployment descriptor for a Web service client running in the Web container
- WebSphere-proprietary deployment descriptor files
  - ibm-webservices-bnd.xmi and ibm-webservices-ext.xmi if provider
  - ibm-webservicesclient-bnd.xmi and ibm-webservicesclient-ext.xmi if client.

WSDL2Java also generates deployment code:

- The WebSphere-proprietary *_Helper, *_Deser, and *_Ser classes that bind the application to the engine and allow deserialization and serialization of messages. If the application contains exceptions, <Exception>_DeserProxy classes are also generated.
- If the application is a client, WebSphere-proprietary *ServiceInformation, *Locator, and *Stub classes.

Important: Do not modify deployment code that results from WSDL2Java. If you have modified the deployment code and are having problems, re-generate the deployment code and test the application without modification. Modified deployment code is not supported by IBM.

Problems that occur during development

This topic covers problems that occur as a result of using WSDL2Java during the development of Web service clients and providers for WebSphere Application Server.

WSDL2Java allows you to develop Java API for XML-based RPC (JAX-RPC) Web service applications. You do not have to use the WebSphere WSDL2Java tool to create a JAX-RPC application to run in WebSphere, but the application must comply with JAX-RPC Specification Version 1.1 and the Web Services for J2EE Specification Version 1.1.

- The JAX-RPC 1.1 specification can be downloaded at:

  http://jcp.org/aboutJava/communityprocess/final/jsr101/index2.html

- The Web services for J2EE specification (JSR 109) can be downloaded at:

  http://java.sun.com/webservices/reference/apis-docs/index.jsp

This activity does not include problems during runtime of applications developed with WSDL2Java. For that see “Problems that occur during runtime” on page 24.

Identify symptoms

Symptoms of a WSDL2Java problem covered in this topic include:

- Error or warning messages with the WSWS prefix occur when WSDL2Java is executed.

  When WSDL2Java is used from the command line, the error messages appear in the command window as it executes. When WSDL2Java errors result during wizard execution in Rational Application Developer, the error is displayed in a new error window.

- WSDL2Java generates code that does not compile.

  The compiler will report errors that occur during the compile of WSDL2Java-generated code. If you are using Rational Application Developer or the Application Server Toolkit, you will see the messages in the workbench. If using the command-line (javac), you will see the messages displayed to that interface (the DOS prompt or korn shell).

Symptoms discussed in the information center

You will find information about additional symptoms related to WSDL2Java in the WebSphere Application Server Information Center.

If you see your symptom in the following list, please refer to the relevant article in the Information Center.

- Compiler errors

  Web services compiled bindings troubleshooting tips:

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/rwbs_trbjavacompiler.html

  Java code to Web Service Description Language (WSDL) mapping cannot be reversed to the original Java code.

- Command-line tools symptom list

  Web services command-line tools troubleshooting tips:

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/rwbs_trbcommand.html

  - The .Net client does not reflect a Web service method with Vector-type parameters.

    The following exception is displayed:

    ```
    System.InvalidOperationException: Method
    AnnuityInteropService.wsListAnnuityByHolder can not be reflected.

    System.InvalidOperationException: There was an error reflecting
    wsListAnnuityByHolderResult.

    System.InvalidOperationException: The Form property might not be
    unqualified when an explicit namespace property is available.
    ```

  - Multiprotocol port component restrictions with Web Services for J2EE Version 1.0 and 1.1 Specifications:

    ```
    Error in <module> : CHKW6030E: Implementation class <class>
    referred to by port components<port1> and <port2>. (JSR109 1.0:
    7.1.2).
    ```

  - Avoiding application errors after uninstalling an interim fix, a fix pack, or a refresh pack.
  - Using a proxy server to access the Internet while running the WSDL2Java command causes your connection to time out.
  - An emitter failure error occurs when running the WSDL2Java command on a WSDL document containing a JMS-style endpoint URL:

    ```
    WSWS3099E: Error: Emitter failure. Invalid endpoint address in
    port <x> in service <y>: <jms-url-string>
    ```

Collect and diagnostics

Collect the following diagnostic data:

- WSWSxxxxE error messages that resulted from issuing the WSDL2Java command
- Error messages issued as a result of using a Web services wizard in Rational Application Developer or Application Server Toolkit wizard (for example, the Web services client wizard)
- Output of the compilation (for example, messages that result from using javac)

Evaluate the error messages

Information about messages with the WSWS prefix, including suggested actions, can be found in the WebSphere Information Center in the Messages section under References. Review the user response for WSWS messages at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.messages.doc/com.ibm.ws.webservices.multiprotocol.resources.mpMessages.html

Note that this is the first of five WSWS message topics.

If you see a message other than WSWS3457W or WSWS3205E, go to “The next step” on page 47 for information about performing online searches and gathering information required by IBM support.

WSWS3574E, WSWS3205E

The messages shown in Example 1 indicate that a relative URI has been used. If you see these messages, go to “targetNamespace is a relative URI” on page 12.

Example 1 WSWS3574E, WSWS3205E

```
WSWS3574E: ---------- FATAL ERRORS ENCOUNTERED ----------
   GENERATION OF ARTIFACTS HAS BEEN SUSPENDED.
   See messages to follow for more details.
WSWS3205E: Error: Type {relative.test.com}sayHelloRequestType is
referenced but not defined.
```

The same error in Rational Application Developer would appear during the execution of a Web services wizard. It would appear in an error box (Figure 1).

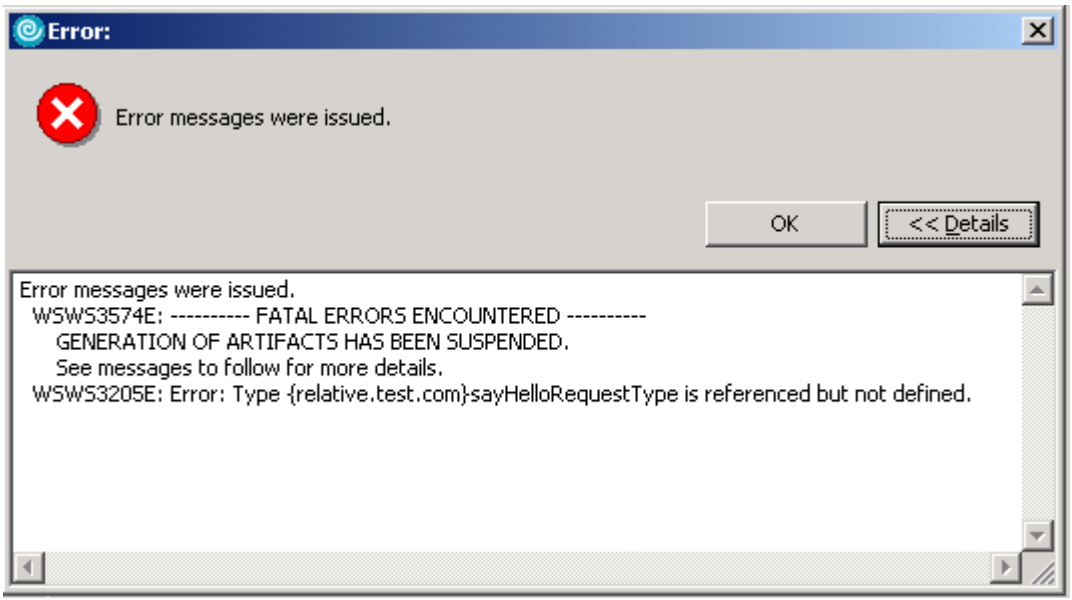

Figure 1 WSWS3547E issued in Rational Application Developer

WSWS3099E

If you have the emitter failure error `WSWS3099E: Error: Emitter failure. Invalid endpoint address in port <x> in service <y>: <jms-url-string>` when running the WSDL2Java command on a WSDL document containing a JMS-style endpoint URL, see the “Web services command-line tools troubleshooting tips” WebSphere Information Center article for more information, available at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/rwbs_trbcommand.html

Evaluate compiler output

Compiler output should be scanned for indications of an error. Some errors are self-explanatory, but others may require a search for known problems.

Illegal start of type

Example 2 shows an example of compiler output with errors. In this example, the compiler is complaining about the period character immediately preceding TestHelloResponse, TestHelloRequest, and Test.

This particular problem was caused because a non-traditional URI was used in the WSDL targetNamespace. For more information about the use of non-traditional URIs, see “targetNamespace contains a non-traditional URI scheme” on page 10.

Example 2 Compiler output with errors

```
Test.java:10: illegal start of type
    public .TestHelloResponse testHello(.TestHelloRequest parameters)
throws java.rmi.RemoteException, .HelloWorldException;
           ^
Test.java:10: <identifier> expected
    public .TestHelloResponse testHello(.TestHelloRequest parameters)
throws java.rmi.RemoteException, .HelloWorldException;

^
TestService.java:10: illegal start of type
    public .Test getTest() throws javax.xml.rpc.ServiceException;
           ^
TestService.java:10: <identifier> expected
    public .Test getTest() throws javax.xml.rpc.ServiceException;
                                                                 ^
TestService.java:14: illegal start of type
```

```
    public .Test getTest(java.net.URL portAddress) throws
javax.xml.rpc.ServiceException;
           ^
TestService.java:14: <identifier> expected
    public .Test getTest(java.net.URL portAddress) throws
javax.xml.rpc.ServiceException;

^
6 errors
```

Where to go from here: If you have not identified a specific symptom in those discussed here, go to “The next step” on page 47 for information about performing an online search of Web services problems and for a list of useful technical articles. If these resources do not help you resolve the problem, contact IBM support.

targetNamespace contains a non-traditional URI scheme

If a WSDL schema targetNamespace contains a non-traditional URI scheme, WSDL2Java can generate code that does not compile.

For information about URI syntax see *RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax* at:

http://www.faqs.org/rfcs/rfc2396.html

Example 3 shows an example of WSDL that contains a non-traditional URI in its namespace declarations and embedded schema targetNamespace.

Example 3 WSDL containing non-traditional URI

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="hhttp://test.com"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:intf="hhttp://test.com"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
   <schema targetNamespace="hhttp://test.com"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
       <element name="testHelloResponse">
         <complexType>
           <sequence>
             <element name="testHelloReturn" nillable="true" type="xsd:string"/>
           </sequence>
         </complexType>
```

```
      </element>
      <element name="testHelloRequest">
        <complexType>
          <sequence>
           <element name="name" nillable="true" type="xsd:string"/>
          </sequence>
        </complexType>
    </element>
</schema>
```

Resolve the problem

Previously, WSDL2Java did not consider non-traditional URI schemes when mapping namespaces to package names. PK35427 fixes WSDL2Java so that it can properly parse namespaces with such non-traditional URI schemes and transform them into package names, instead of just an empty string.

The solution is to apply fix pack 6.1.0.9, which includes PK35427, on the WebSphere installation where WSDL2Java is invoked. Then invoke WSDL2Java to re-generate the classes. The newly generated classes should not contain any compilation errors.

Workaround

To work around the compilation error, change the WSDL and schema targetNamespace to use a traditional URI scheme, such as http, file, or urn. For example, from:

```
hhttp://test.com
```

to:

```
http://test.com
```

Then invoke WSDL2Java to re-generate the classes. The resulting classes should not contain any compilation errors.

More about this problem

Consider the WSDL schema shown in Example 3 on page 10. Example 4 shows the portion of the WSDL that references these schema types.

Example 4 WSDL referencing the schema

```
<wsdl:message name="testHelloRequest">
    <wsdl:part name="parameters" element="intf:testHelloRequest"/>
  </wsdl:message>
  <wsdl:message name="testHelloResponse">
```

```
  <wsdl:part name="parameters" element="intf:testHelloResponse"/>
</wsdl:message>
<wsdl:portType name="Test">
  <wsdl:operation name="testHello">
    <wsdl:input name="testHelloRequest" message="intf:testHelloRequest"/>
    <wsdl:output name="testHelloResponse" message="intf:testHelloResponse"/>
  </wsdl:operation>
</wsdl:portType>
```

From this WSDL, WSDL2Java would map the complexTypes of elements testHelloRequest and testHelloResponse to Java types, and generate them in no package. Hence, any other classes that reference these Java types would incorrectly use the notation shown in Example 5.

Example 5 WSDL2Java generated code that will not compile

```
public interface Test extends java.rmi.Remote {
    public .TestHelloResponse testHello(.TestHelloRequest parameters)
    throws java.rmi.RemoteException;
}
```

The Java compiler would complain about the period character immediately preceding the Java types TestHelloResponse and TestHelloRequest.

Support preparation

If the work around or applying fix pack 6.1.0.9 does not resolve the problem, collect the following data for use by IBM support:

- WSDL and any referenced schema files
- WSDL2Java command invocation, or a description of how Rational Application Developer or the Application Server Toolkit was used to generate the non-compiling classes

Go to “The next step” on page 47 for information about performing an online search of Web services problems and for a list of useful technical articles. If these resources do not help you resolve the problem, contact IBM support.

targetNamespace is a relative URI

The error is not due to a product defect. It occurs when the WSDL contains an embedded schema with a targetNamespace set to a relative URI. The FATAL ERROR message just indicates that WSDL2Java will stop generating code. It does not indicate a fatal error with the operating system — just that WSDL2Java will abort.

Resolve the problem

Note: WSDL2Java requires that targetNamespaces be absolute URIs.

Change the WSDL so that it uses absolute URIs and repeat the WSDL2Java process.

More about this problem

When invoking WSDL2Java on the command line, you will get an error similar to Example 6 displayed to the console.

Example 6 WSWS3574E, WSWS3205E messages

```
WSWS3574E: ---------- FATAL ERRORS ENCOUNTERED ----------
   GENERATION OF ARTIFACTS HAS BEEN SUSPENDED.
   See messages to follow for more details.
WSWS3205E: Error: Type {relative.test.com}sayHelloRequestType is
referenced but not defined.
java.io.IOException: FATAL ERRORS encountered by WSDL2Java tool.
   at com.ibm.ws.webservices.wsdl.Parser.generate(Parser.java:467)
   at com.ibm.ws.webservices.wsdl.Parser.generate(Parser.java:394)
   at com.ibm.ws.webservices.wsdl.Parser.run(Parser.java:299)
   at com.ibm.ws.webservices.wsdl.WSDL2.run(WSDL2.java:318)
   at com.ibm.ws.webservices.tools.WSDL2Java.main(WSDL2Java.java:492)
   at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
   at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.ja
va:64)
   at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccesso
rImpl.java:43)
   at java.lang.reflect.Method.invoke(Method.java:615)
   at com.ibm.ws.bootstrap.WSLauncher.main(WSLauncher.java:263)
```

This error happens because the WSDL contains an embedded schema with a targetNamespace set to a relative URI.

Example

In this example, the WSDL definitions element contains the namespace declarations shown in Example 7.

Example 7 WSDL namespace declarations

```
<wsdl:definitions targetNamespace="http://helloworld.com"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns1="relative.test.com"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 ...>
```

Then the WSDL embeds an XML schema (Example 8).

Example 8 Embedded XML schema

```
<wsdl:types>
 <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="relative.test.com">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
    <complexType name="sayHelloRequestType">
      <sequence>
         <element name="msg" type="xsd:string"/>
         <element name="id" type="xsd:int"/>
      </sequence>
    </complexType>
    <complexType name="sayHelloResponseType">
      <sequence>
        <element name="msg" type="xsd:string"/>
    <element name="id" type="xsd:int"/>
      </sequence>
    </complexType>
  </schema>
</wsdl:types>
```

Notice the embedded schema's targetNamespace, relative.test.com. The schema complexTypes are then referenced later in the WSDL messages, as shown in Example 9.

Example 9 WSDL messages

```
<wsdl:message name="sayHelloRequest">
        <part name="parameter" type="tns1:sayHelloRequestType"/>
  </wsdl:message>
  <wsdl:message name="sayHelloResponse">
    <wsdl:part name="parameter" type="tns1:sayHelloResponseType"/>
  </wsdl:message>
```

WSDL2Java requires that targetNamespaces be absolute URIs. Since the WSDL has a targetNamespace of http://helloworld.com and the embedded schema has a specified targetNamespace of relative.test.com, WSDL2Java tries to make that relative URI absolute by appending it to the base URN — the WSDL targetNamespace.

In other words, given this WSDL targetNamespace http://helloworld.com and schema targetNamespace relative.test.com, WSDL2Java would resolve the targetNamespace of the embedded schema as:

```
http://helloworld.comrelative.test.com
```

However, the prefix tns1 is set to the relative URI relative.test.com, and the schema complexTypes are referred to using that prefix shown (see Example 9).

In essence, tns1:sayHelloRequestType means the same as {relative.test.com}sayHelloRequestType.

The WSDL messages reference the schema types using namespace relative.test.com, but WSDL2Java resolved the targetNamespace of those schema types as http://helloworld.comrelative.test.com. Therefore, it issues the error message:

```
WSWS3205E: Error: Type {relative.test.com}sayHelloRequestType is
referenced but not defined.
```

One way to solve the error is to set the tns1 prefix to the expected URI, http://helloworld.comrelative.test.com, as shown in Example 10.

Example 10 Modified namespace declaration

```
<wsdl:definitions targetNamespace="http://helloworld.com"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns1="http://helloworld.comrelative.test.com"

 ...>
```

But, the namespace http://helloworld.comrelative.test.com is most likely not what was intended. So the best guideline is to avoid relative URIs altogether, as in Example 11.

Example 11 targetNamespace modified to be absolute URI

```
<wsdl:types>
 <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://relative.test.com">
    …
  </schema>
</wsdl:types>
```

Support preparation

If changing the schema targetNamespace URI from relative to absolute does not alleviate the WSWS3205E error, collect the following data for use by IBM support:

- WSDL
- WSDL2Java command invocation, or a description of how Rational Application Developer or the Application Server Toolkit was used when the error was encountered

Go to “The next step” on page 47 for information about performing an online search of Web services problems and for a list of useful technical articles. If these resources do not help you resolve the problem, contact IBM support.

Problems deploying Web services applications

This activity covers Web services problems that occur when deploying a Web service client or provider application to WebSphere Application Server. Specifically, it covers problems that occur in WSDL2Java.

Important: Do not modify deployment code that results from WSDL2Java. If you have modified the deployment code and are having problems, re-generate the deployment code and test the application without modification. Modified deployment code is not supported by IBM.

A Web services application deployed to WebSphere Application Server must include the various, proprietary deployment code generated by WSDL2Java. This deployment code can be generated by:

- The WSDL2Java tool
- Selecting the -deployws option when installing the application using wsadmin
- Selecting the Deploy WebServices option when installing the application using the administrative console
- The wsdeploy tool
- Using Rational Application Developer or the Application Server Toolkit

However, keep in mind that all of these methods use WSDL2Java to generate the code.

The WSDL2Java options to deploy the application depend on the application type:

- If the application is a Web service implementation, then WSDL2Java is invoked with the -role deploy-server option.
- If the application is a Web service client, then WSDL2Java is invoked with the -role deploy-client option.
- For both client and server role, the -introspect option is also enabled.

Identify symptoms

Symptoms of a Web services deployment problem generally appear in the SystemOut log or, if using the **wsdeploy** command, in the command window. The first indication is the following message:

```
WSWS0038E: Error from Web services deploy tool
```

This message is followed by more specific error messages in the following format:

```
WSWSxxxxE
```

Where to go: If you are experiencing WSWS3467E, go to “targetNamespace contains a non-traditional URI scheme” on page 10.

If you do not know the message or are experiencing any other symptoms related to Web services deployment, go to “Collect diagnostics” on page 18.

Collect diagnostics

Collect SystemOut log. For information about collecting SystemOut see “Collecting JVM logs” on page 44.

Analyze SystemOut

Scan the SystemOut log for WSWSxxxxE messages. See Example 12.

Example 12 WSWSxxxxE messages in SystemOut

```
[1/19/07 8:59:24:984 MST] 00000065 WSDeployTask  E   WSWS0038E: Error
from Web services deploy tool:
WSWS3574E: ---------- FATAL ERRORS ENCOUNTERED ----------
   GENERATION OF ARTIFACTS HAS BEEN SUSPENDED.
   See messages to follow for more details.

.. error messages

[1/19/07 8:59:25:000 MST] 00000065 WSDeployTask E WSWS0038E: Error from
Web services deploy tool: Error: FATAL ERRORS encountered by WSDL2Java
tool.
```

Evaluate SystemOut

Information about messages with the WSWS prefix, including suggested actions, can be found in the WebSphere Information Center in the Messages section under References.

If you see `WSWS3467E: WSDL2Java validation error`, go to “targetNamespace contains a non-traditional URI scheme” on page 10.

Otherwise, go to “The next step” on page 47 for information about performing online searches and gathering information required by IBM support.

Invalid public constructor

WSWS3467E occurs when the validation process that occurs during the WSDL2Java command has detected an invalid constructor. Specifically, it has detected an exception that is missing a constructor that takes the same number of parameters that correspond to the number of elements inside of the element's complexType.

The WSDL2Java command (invoked during the deploy process) uses the -introspect option, which directs the WSDL2Java command to examine and validate existing Java classes against the JAX-RPC 1.1 specification and the WSDL, and to generate code that is compatible with those existing Java classes.

WSWS3467E errors are a result of an error detected by this validation.

Resolve the problem

Correct the constructor and re-deploy the application.

More about this problem

The JAX-RPC 1.1 specification describes Java APIs for XML-based Remote Procedure Call (RPC) mechanisms. It is one of the central specifications that the Web services engine follows. Its description of WSDL and XML to Java mapping is crucial to understanding and resolving this problem.

Example

In this example, the user receives the error message shown in Example 13.

Example 13 WSWS3467E

```
[1/19/07 8:59:24:984 MST] 00000065 WSDeployTask  E   WSWS0038E: Error
from Web services deploy tool:
WSWS3574E: ---------- FATAL ERRORS ENCOUNTERED ----------
   GENERATION OF ARTIFACTS HAS BEEN SUSPENDED.
   See messages to follow for more details.

[1/19/07 8:59:24:984 MST] 00000065 WSDeployTask E WSWS0038E: Error from
Web services deploy tool: WSWS3467E: ---------- WSDL2Java VALIDATION
ERROR ----------
        Existing Fault class is invalid: "com.test.HelloWorldException"
        Does not implement a valid public constructor:
"HelloWorldException", containing 3 parameters.

[1/19/07 8:59:25:000 MST] 00000065 WSDeployTask E WSWS0038E: Error from
Web services deploy tool: Error: FATAL ERRORS encountered by WSDL2Java
tool.
```

First, examine HelloWorldException, the exception that the message claims is invalid. See Example 14.

Example 14 HelloWorldException

```
package com.test;

public class HelloWorldException extends java.lang.Exception {
  private java.lang.String message;
  private java.lang.Integer id;
  private java.lang.String detail;

  public HelloWorldException(java.lang.String message) {
    this.message = message;
  }

  public java.lang.String getMessage() {
    return message;
  }

  public java.lang.Integer getId() {
    return id;
  }
```

```
  public java.lang.String getDetail() {
    return detail;
  }

}
```

The error message specifically mentions the exception's constructor. Notice that the constructor takes one parameter. Yet the message states that in order for the constructor to be valid, the constructor must take three parameters. What is the basis for this claim?

Examine the WSDL file. Remember, when a Web services application is deployed, WSDL2Java is used *behind the scenes* to examine existing classes in the application, validate them against the WSDL, and verify that they comply with JAX-RPC.

That schema type is an anonymous complexType and is defined as shown in Example 15.

Example 15 hellowWorldException element - anonymous complexType definition

```
<element name="helloWorldException">
    <complexType>
     <sequence>
   <element name="message" nillable="true" type="xsd:string"/>
   <element name="id" nillable="true" type="xsd:int"/>
   <element name="detail" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
```

The helloWorldException element is referenced in the message with the same name, as shown in Example 16.

Example 16 helloWorldException message defined in WSDL

```
<wsdl:message name="helloWorldException">
    <wsdl:part name="fault" element="impl:helloWorldException"/>
  </wsdl:message>
```

And this message is referenced as a fault in the WSDL operation, testHello, as shown in Example 17.

Example 17 WSDL operation containing WSDL fault

```
<wsdl:operation name="testHello">
  <wsdl:input name="testHelloRequest"
         message="impl:testHelloRequest"/>
  <wsdl:output name="testHelloResponse"
         message="impl:testHelloResponse"/>
  <wsdl:fault name="helloWorldException"
         message="impl:helloWorldException"/>
</wsdl:operation>
```

Note: The JAX-RPC specification says this about the mapping of WSDL faults to Java exceptions:

> A wsdl:fault is mapped to either a java.rmi.RemoteException (or its subclass), service specific Java exception (described later in this section) or a javax.xml.rpc.soap.SOAPFaultException.

JAX-RPC 1.1 goes on to elaborate on service specific exceptions:

> A service specific Java exception (mapped from a wsdl:fault and the corresponding wsdl:message) extends the class java.lang.Exception directly or indirectly. The single message part in the wsdl:message (referenced from the wsdl:fault element) may be either a type or an element. If the former, it can be either a xsd:complexType or a simple XML type. **Each element inside the xsd:complexType is mapped to a getter method and a parameter in the constructor of the Java exception. Mapping of these elements follows the standard XML to Java type mapping**.

The full specification can be downloaded from:

http://java.sun.com/xml/downloads/jaxrpc.html

In the case of our example, the WSDL fault is clearly mapped to a service-specific exception. The bolded text highlights the remarks relevant to the WSWS3467E WSDL2Java validation error. The HelloWorldException class is missing a constructor that takes three parameters that correspond to the three elements inside of the helloWorldException element's complexType.

This point is illustrated by contrasting the exception constructor (Example 18) with its mapped complexType (Example 19).

Example 18 HelloWorldException class constructor - incorrect

```
public HelloWorldException(java.lang.String message) {
    this.message = message;
    }
```

Example 19 helloWorldException element and complexType definition

```
<element name="helloWorldException">
    <complexType>
     <sequence>
   <element name="message" nillable="true" type="xsd:string"/>
   <element name="id" nillable="true" type="xsd:int"/>
   <element name="detail" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
```

To be compliant with JAX-RPC, HelloWorldException should also implement a constructor like that shown in Example 20.

Example 20 HelloWorldException class constructor - correct

```
public HelloWorldException(
           java.lang.String message,
           java.lang.Integer id,
           java.lang.String detail) {
        this.message = message;
        this.id = id;
        this.detail = detail;
    }
```

Support preparation

If changing the exception constructor does not resolve the problem, collect the following data for use by IBM support:

- The complete Web services application that fails to deploy, including the WSDL file and the Java source of the service specific exception.
- A description of how the Web services application was deployed — for instance, invoking the wsdeploy tool or installing the application using the administrative console.

- If deploying the application using the administrative console, enable the Web services trace to capture the WSWS3467E deployment error. Instructions for enabling the Web services trace can be found in “Enabling the Web services trace” on page 46.
- If wsdeploy is used, then collect a simple text capture of the error sent to standard output.

Go to “The next step” on page 47 for information about performing an online search of Web services problems and for a list of useful technical articles. If these resources do not help you resolve the problem, contact IBM support.

Problems that occur during runtime

This topic covers problems that occur in Web services applications running in WebSphere Application Server.

- WebSphere application as the Web service client

  When a WebSphere application invokes a Web service, it becomes a Web service client. The Web service being invoked can be another WebSphere application or may be hosted on any other type of system that provides Web services. The system hosting the Web service is the Web service provider.

  Problems invoking a Web service can be a result of several things, for example:

  - The application acting as the Web service provider is not available or is broken.
  - The URL for the Web service provider has changed.
  - The Web service client code has a problem.
  - The Web service request is routed through a gateway or enterprise service bus with a problem.

- WebSphere application as the Web service provider

  When a WebSphere application that provides a Web service experiences a problem, it may behave incorrectly or become unavailable. In addition to any error symptoms generated by the application, this type of problem usually creates error symptoms at the Web service client. The Web service client may or may not be another WebSphere application.

Check system integrity

When a WebSphere Application Server application is the Web service client and receives an error, the simplest thing to do initially is to ensure that the Web service provider is available and that the URL used by the requester is correct.

Clients attempting to access a provider with an incorrectly specified URL commonly occurs when applications are moved from development environments to production ones. Ports and host names most likely change, and clients must be updated with the new Web service target URL.

Tip: When the Web service provider is an application running in WebSphere Application Server, you can test the availability of the service by pointing a browser to the URL. If the test is successful, you should see two lines. The first contains the full QName of WSDL port, and the second the message `Hi there, this is a Web service!`

Identify symptoms

Symptoms of a Web services problem appear in a variety of ways depending on the problem, the client's environment, and how the client was designed. For example, if invoking an unmanaged client from the command-line console, the error messages would be displayed to that interface. If the client is a Web application that writes to an output stream that appears in the Web browser, then the errors would be displayed to the browser. On the provider system, the messages will appear in the application server SystemOut log.

Messages will, with a WSWS prefix indicate a Web services related error. Problems may also occur that do not produce a Web services error message but will appear as unexpected results in the application. A trace of the SOAP messages often shows the error.

Specific symptoms covered in this activity include:

- Client system
  - javax.xml.rpc.ServiceException with message code WSWS5055E or WSWS5014E.

    For more information about this symptom see "targetNamespace contains a non-traditional URI scheme" on page 10.
  - WSWS3713E: Connection to the remote host localhost failed. Received the following error: `Connection refused`.

    Verify that the host and port used to call the provider are correct.

- Client or provider system
  - org.xml.sax.SAXException: WSWS3047E

    For more information about this symptom see “Improperly formed SOAP request or response” on page 38.
  - A SOAP message with an unexpectedly empty body

    For more information about this symptom see “Serialized SOAP message does not contain children in body” on page 35.

Collect diagnostics

If the symptoms are not apparent, you will need to collect the following:

- If the problem occurs at the client:
  - Messages displayed.
  - Logs that the client application writes messages to.
  - If the client is a managed client running in the server, collect SystemOut log.
- If the problem occurs at the provider: SystemOut log.

  For information about collecting the SystemOut log see “Collecting JVM logs” on page 44.
- TCPMonitor trace.

  Using TCPMonitor to view messages will not always be applicable or relevant in every case. Sometimes, runtime problems occur before the SOAP message can be serialized and sent across the wire. Or problems may have nothing to do with how a message appears.

  However, for certain problems, it is important to have a copy of the SOAP request/response exchange. In particular, capture the trace when:
  - There is an issue with an HTTP header, for example, it is not set to the expected value or it does not appear at all.
  - There appears to be a deserialization problem. The message text or Java stack trace usually contains the word *deserialization* if this is the case.
  - There is an issue with a SOAP message.

  The message exchange can be captured by recreating the problem and tracing it using TCPMonitor. For a quick reference on using TCPMonitor, see “Collecting trace data with TCPMonitor” on page 45.

  You may want to analyze the SystemOut log first before deciding whether you should recreate the problem and collect this trace.

Glossary terms:

- *Serialization* is the process of converting Java objects to XML.
- *Deserialization* is the process of converting XML to Java objects.

Analyze diagnostics

To analyze the diagnostics:

1. Scan the SystemOut log for errors.
2. If captured, scan the TCPMonitor trace output for the message causing the problem.

If your specific symptom is not addressed here, go to “The next step” on page 47 for information about performing online searches and gathering information required by IBM support.

Analyze SystemOut

In the SystemOut log, look for and make a note of any of the following:

- Error messages with WSWS as the prefix.

  Review the user response for WSWS messages at:

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.messages.doc/com.ibm.ws.webservices.multiprotocol.resources.mpMessages.html`

  Note that this is the first of five WSWS message topics.
- javax.xml.rpc.ServiceException
- org.xml.sax.SAXException
- Java stack trace with any other errors

javax.xml.rpc.ServiceException, WSWS5055E, WSWS5014E

The following messages indicate a problem when using the *wsdl_service*Locator class generated by WSDL2Java to obtain a stub object for invoking the Web service.

- `javax.xml.rpc.ServiceException: WSWS5055E: Cannot find a Service for namespace null`
- `javax.xml.rpc.ServiceException: WSWS5014E: An error occurred instantiating the generated Stub class class: java.lang.reflect.InvocationTargetException`

Prior to fix pack 6.0.2.19 for V6.0.2 and fix pack 6.1.0.9 for V6.1, this can be due to a bug in the WSDL2Java tool that occurs when port names contain non-alphanumeric characters.

For more information see “Port name problem” on page 31.

org.xml.sax.SAXException: WSWS3047E

WSWS3047E can appear on the client or provider system. If you see this message see “Improperly formed SOAP request or response” on page 38.

When a Web service client sends a request to a WebSphere Application Server Web service, you might observe a SOAP fault like in Example 21 as a response.

Example 21 WSWS3047E SOAP fault

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <soapenv:Fault>
         <faultcode>soapenv:Server.generalException</faultcode>
         <faultstring><![CDATA[org.xml.sax.SAXException:
            WSWS3047E: Error: Cannot deserialize element name of bean
             com.test.TestHello.
            Message being parsed:
            <soapenv:Envelope
                    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
             xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
               <soapenv:Header/>
                 <soapenv:Body>
                  <p585:testHello xmlns:p585="http://test.com">
                    <p585:name>Wendy</p585:name>
                  </p585:testHello>
                 </soapenv:Body>
              </soapenv:Envelope>]]>
         </faultstring>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

On the WebSphere Web service provider, the error might be manifested in the server's SystemOut. See Example 22.

Example 22 WSWS3047E on the Web service provider

```
[3/7/07 16:43:13:708 CST] 00000022 UserException E   WSWS3228E: Error: Exception:
WebServicesFault
 faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.generalException
 faultString: org.xml.sax.SAXException: WSWS3047E: Error: Cannot deserialize element
name of bean com.test.TestHello.

Message being parsed:
<soapenv:Envelope  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header/>
  <soapenv:Body>
    <p585:testHello xmlns:p585="http://test.com">
      <p585:name>Wendy</p585:name>
    </p585:testHello>
  </soapenv:Body>
</soapenv:Envelope>

faultActor: null faultDetail:

org.xml.sax.SAXException: WSWS3047E: Error: Cannot deserialize element name of bean
com.test.TestHello. Message being parsed:
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
    <p585:testHello xmlns:p585="http://test.com">
      <p585:name>Wendy</p585:name>
    </p585:testHello>
  </soapenv:Body>
</soapenv:Envelope>
```

Similarly, the WSWS3047E error might occur on the client, if it is running in the WebSphere environment.

If the WebSphere Web service client is a managed one running in the Web or EJB container, you might also see in the server's SystemOut a stack trace beginning with text similar to Example 23.

Example 23 WSWS3047E on the Web service client

```
? 00000028 MCUtils       1 com.ibm.ws.webservices.engine.utils.MCUtils
interceptDeserializationException interceptDeserializationException:
org.xml.sax.SAXException: WSWS3047E: Error: Cannot deserialize element
name of bean com.test.TestHello. org.xml.sax.SAXException: WSWS3047E:
Error: Cannot deserialize element name of bean com.test.TestHello.
```

If the client is an unmanaged one and using the WebSphere environment, you might see the stack trace displayed in standard out or standard err (such as the command-line console). During development cycles, developers commonly use Rational Application Developer to invoke unmanaged test clients, and might see the WSWS3047E stack trace displayed in the RAD console.

Where to go from here:

- If the problem appears to involve deserialization, the SOAP message format, or the HTTP header, collect a TCPMonitor trace of the problem.
- If you found errors not listed here, go to “The next step” on page 47.

Analyze TCPMonitor output

To analyze the TCPMonitor output:

1. Scan the output to find the message that caused the problem.
2. Examine the message to make sure that it is formed properly according to the WSDL.

Empty SOAP body

A SOAP request or response with unexpected content can cause problems in a Web service client or provider.

One problem that you may experienced is a SOAP message with an unexpectedly empty body, described by a WSDL that defines a document/literal message with no parts. If you have this situation, go to “Serialized SOAP message does not contain children in body” on page 35. See Example 24.

Example 24 Message with empty SOAP Body

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body/>
</soapenv:Envelope>
```

Port name problem

WSWS5055E and WSWS5014E can indicate a reported error in the WSDL2Java tool that occurs when the WSDL port name contains characters other than alphanumerics (for example, periods or dashes).

Resolve the problem

This problem was reported by APAR PK35399 and is fixed in fix pack 6.1.0.9 for V6.1 and fix pack 6.0.2.19 for V6.0.2. Under this APAR, WSDL2Java was changed so that it generated a correct *wsdl_service*Locator class.

Apply the fix pack, then re-generate client deployment code, specifically, the *wsdl_service*Locator class.

Quick test

As a test to determine whether your problem is the same as the problem discussed here, you can modify the *wsdl_service*Locator class, bearing in mind that doing so is not an approved practice and just serves as a temporary work around until the true solution can be applied.

The *wsdl_service*Locator.getPort2NamespaceMap() method should be modified so that the XML QName local part of the WSDL port name is used as the key to the namespace value.

For example, given the following WSDL port definition:

```
<wsdl:port name="Test-Service" binding="impl:TestSoapBinding">
<wsdlsoap:address
location="http://localhost:9080/TestHello/Test"/>
</wsdl:port>
```

change from:

```
port2NamespaceMap.put(
"TestService",
"http://schemas.xmlsoap.org/wsdl/soap/");
```

to:

```
port2NamespaceMap.put(
"Test-Service",
"http://schemas.xmlsoap.org/wsdl/soap/");
```

Save this modified class and then re-invoke client. The WSWS5055E error should no longer occur.

Important: This is a temporary check only. If the client is re-deployed, then this modified class will be overwritten.

Support preparation

If applying the fix pack and re-generating client deployment code does not resolve the problem, collect the following data for use by IBM support:

- Web services trace containing the javax.xml.rpc.ServiceException: WSWS5055E or WSWS5014E error. See “Enabling the Web services trace” on page 46 for instructions on collecting the trace.
- Client application test case, including the WSDL file and Java source.

Go to “The next step” on page 47 for information about performing an online search of Web services problems and for a list of useful technical articles. If these resources do not help you resolve the problem, contact IBM support.

More about this problem

Consider the WSDL file excerpt in Example 25.

Example 25 WSDL service definition

```
<wsdl:service name="Test-Service">
    <wsdl:port name="Test-Service" binding="impl:TestSoapBinding">
      <wsdlsoap:address
location="http://localhost:9080/TestHello/Test"/>
    </wsdl:port>
  </wsdl:service>
```

When invoked to generate client development code, WSDL2Java generates an interface that extends the javax.xml.rpc.Service interface, and that maps to the WSDL service. In this example, that interface is called com.test.TestService, named after the WSDL service.

When emitting client deployment code, WSDL2Java emits a WebSphere proprietary class that implements this interface. The convention used in naming this class is the name of the interface mapped to WSDL service (again, in this

example, com.test.TestService) followed by *Locator* — in this example, com.test.TestServiceLocator.

You can use the *wsdl_service*Locator class to obtain a client-side stub implementation of the remote service endpoint.

Example 26 shows how a client application might use the *wsdl_service*Locator class to obtain the stub.

Example 26 Using the *wsdl_service*Locator class to obtain the stub

```
import com.test.Test;
import com.test.TestServiceLocator;
try {
Test stub = new TestServiceLocator().getTestService();
...
```

Test is the service endpoint interface (what will actually be returned is an instance of another WSDL2Java-generated deployment class, TestSoapBindingStub, which implements Test).

Examine the javax.xml.rpc.ServiceException stack trace in Example 27.

Example 27 javax.xml.rpc.ServiceException stack trace

```
Caught exception javax.xml.rpc.ServiceException: WSWS5055E: Cannot find
a Service for namespace null.
javax.xml.rpc.ServiceException: WSWS5055E: Cannot find a Service for
namespace null.
   at
com.ibm.ws.webservices.multiprotocol.utils.ServiceManager.getServiceFor
Namespace(ServiceManager.java:105)
   at
com.ibm.ws.webservices.multiprotocol.AgnosticService.getGeneratedStub(A
gnosticService.java:525)
   at
com.ibm.ws.webservices.multiprotocol.AgnosticService.doGetPort(Agnostic
Service.java:462)
   at
com.ibm.ws.webservices.multiprotocol.AgnosticService.getStub(AgnosticSe
rvice.java:405)
   at
com.test.TestServiceLocator.getTestService(TestServiceLocator.java:68)
   at
com.test.TestServiceLocator.getTestService(TestServiceLocator.java:63)
   at com.test.TestClient.main(TestClient.java:12)
```

Since it is referenced in a stack frame in the javax.xml.rpc.ServiceException stack trace, line 68 of TestServiceLocator might offer possible clues. Line 68, shown in Example 28, is a call from getTestService() to getStub().

Example 28 Call from getTestService() to getStub()

```
com.test.Test _stub =
            (com.test.Test) getStub(
                testServicePortName,
                (String)
                getPort2NamespaceMap().get(testServicePortName),
                com.test.Test.class,
                "com.test.TestSoapBindingStub",
                portAddress.toString());
```

getStub() is implemented in a parent class of TestServiceLocator, com.ibm.ws.webservices.multiprotocol.AgnosticService. There is never any need for you to examine internal engine code. However, for the purposes of understanding this problem, it is meaningful to know that the second parameter that AgnosticService.getStub() takes is the namespace of the WSDL port.

The way that getTestService() obtains that namespace is by calling the getPort2NamespaceMap() method, implemented in TestServiceLocator:

```
...(String) getPort2NamespaceMap().get(testServicePortName)...
```

The testServicePortName variable is declared and initialized earlier in the class:

```
private java.lang.String testServicePortName = "Test-Service";
```

Notice how the string Test-Service matches the value of the WSDL port, defined as a child of the WSDL service:

```
<wsdl:port name="Test-Service" … >
```

Now look at the implementation of the getPort2NamespaceMap() method in Example 29.

Example 29 getPort2NamespaceMap()

```
private java.util.Map port2NamespaceMap = null;
protected synchronized java.util.Map getPort2NamespaceMap() {
if (port2NamespaceMap == null) {
port2NamespaceMap = new java.util.HashMap();
port2NamespaceMap.put(
"TestService",
"http://schemas.xmlsoap.org/wsdl/soap/");
}
```

```
return port2NamespaceMap;
}
```

getPort2NamespaceMap() creates a map that uses the WSDL port as a key and namespace as a value. It returns this map. Recall that getTestService() invokes a get() on this returned map, passing in testServicePortName as a key to get the namespace value:

```
...(String) getPort2NamespaceMap().get(testServicePortName)...
```

But remember that testServicePortName resolves to "Test-Service", yet the getPort2NamespaceMap() method never added a mapping using "Test-Service" as a key. It used the string "TestService" instead:

```
port2NamespaceMap.put(
"TestService",
"http://schemas.xmlsoap.org/wsdl/soap/");
```

Therefore, the returned namespace value is null, which is why the Web services engine throws the `javax.xml.rpc.ServiceException: WSWS5055E: Cannot find a Service for namespace null` exception.

The fix for the APAR corrects WSDL2Java so that it generates the *wsdl_service*Locator class to use the XML QName local part of the WSDL port.

Serialized SOAP message does not contain children in body

This problem occurs when a document/literal WSDL defines a message without a part. At run time, the serialized SOAP message does not contain any children under the SOAP Body.

Consider a document/literal WSDL with the following message defined:

```
<wsdl:message name="testHelloRequest"/>
```

This message is used as input for the operation testHello shown in Example 30.

Example 30 Input message in WSDL operation

```
<wsdl:operation name="testHello">
<wsdl:input name="testHelloRequest" message="intf:testHelloRequest"/>
…
</wsdl:operation>
```

A Web service client developed from such a WSDL and running in the WebSphere environment would send a message like that shown in Example 31.

Example 31 Message with empty SOAP Body

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body/>
</soapenv:Envelope>
```

Notice that the SOAP Body element is empty.

Resolve the problem

If the SOAP Body was not intended to be empty, correct the WSDL, and re-develop and re-deploy the application.

More about this problem

The WSDL defines a document/literal message, as specified in the binding. See Example 32.

Example 32 WSDL defining a document/literal message

```
<wsdl:binding name="TestSoapBinding" type="intf:Test">
    <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="testHello">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="testHelloRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="testHelloResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="helloWorldException">
        <wsdlsoap:fault name="helloWorldException" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
```

Also remember the input message, testHelloRequest? It does not contain a part:

```
<wsdl:message name="testHelloRequest"/>
```

Basic Profile Version 1.1: The Basic Profile Version 1.1 says this on the matter of child elements in document/literal messages:

> 4.7.8 Child Element for Document-Literal Bindings
>
> WSDL 1.1 is not completely clear what, in document-literal style bindings, the child element of soap:Body is.
>
> R2712 A document-literal binding MUST be serialized as an ENVELOPE with a soap:Body whose child element is an instance of the global element declaration referenced by the corresponding wsdl:message part.

In order for the request SOAP Body to contain any elements, the input message must contain a part that references an element, globally defined in the schema. See Example 33.

Example 33 Input message, now with a part

```
<wsdl:message name="testHelloRequest">
    <wsdl:part name="parameters" element="impl:testHello"/>
  </wsdl:message>
```

The testHello element is declared as in Example 34.

Example 34 testHello element

```
<element name="testHello">
    <complexType>
     <sequence>
      <element name="name" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
```

Remember that elements can be declared using anonymous complexTypes. The complexType definition specifies that element testHello contains a child element, called name.

It is also important to understand that changing the WSDL often necessitates changing the Java code — including development and deployment code. In this particular example, a change to the input message changed the signature of the mapped Java method, testHello(), and a slight change in the implementation in both client and service applications. Web services deployment code also had to be re-generated.

Example 35 shows the ensuing SOAP request message after the applications were re-built and re-deployed.

Example 35 SOAP request message with SOAP Body containing child elements

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <p585:testHello xmlns:p585="http://test.com">
         <name>Wendy</name>
      </p585:testHello>
   </soapenv:Body>
</soapenv:Envelope>
```

In this example, the document/literal request message has an empty SOAP Body because the input message has no part. However, this cause and effect pattern is not limited to just the request, but also to the response. In other words, if the output message similarly contains no part, then the response message's SOAP Body will also not contain any elements.

Support preparation

If changing the WSDL and the application does not resolve the problem, collect the following data for use by IBM support:

- Complete test case, including WSDL and Java source
- Web services trace that reflects the sending of the message with the empty SOAP body

Go to “The next step” on page 47 for information about performing an online search of Web services problems and for a list of useful technical articles. If these resources do not help you resolve the problem, contact IBM support.

Improperly formed SOAP request or response

The WSWS3047E deserialization error is a common problem. It usually occurs in the case where either a WebSphere Web service requestor or provider is interoperating with a requestor or provider on another Web services platform. The incoming SOAP message, whether request or response, is not formed properly. The error does not happen because of a product defect.

Resolve the problem

Fix the message so that local elements are qualified or unqualified and in the proper namespace according to the schema. You should work with the Web services vendor to accomplish this task.

More about this problem

Consider the case where a non-WebSphere client sends a request to a WebSphere Web service. Examine the request message (Example 36) that resulted in the WebSphere Web service sending a SOAP fault as a response.

Example 36 Request message

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <p585:testHello xmlns:p585="http://test.com">
         <p585:name>Wendy</p585:name>
      </p585:testHello>
   </soapenv:Body>
</soapenv:Envelope>
```

This request is sent to the target Web service testHello, running on WebSphere.

The testHello Web service operation is defined in the WSDL as shown in Example 37.

Example 37 testHello Web service operation

```
<wsdl:operation name="testHello">
      <wsdl:input name="testHelloRequest"
message="impl:testHelloRequest"/>
      <wsdl:output name="testHelloResponse"
message="impl:testHelloResponse"/>
    </wsdl:operation>
```

In this example, since the deserialization problem occurs during processing of request, the input message is of interest. See Example 38.

Example 38 Input message

```
<wsdl:message name="testHelloRequest">
    <wsdl:part name="parameters" element="impl:testHello"/>
  </wsdl:message>
```

Now look at the element that the message part refers to. `testHello` is declared in the WSDL's embedded schema. See Example 39.

Example 39 testHello global element

```
<schema targetNamespace="http://test.com"
 xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <element name="testHello">
    <complexType>
     <sequence>
      <element name="name" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
  </schema>
```

testHello is a global element. It is declared immediately under the schema element, and not within a complexType definition. Therefore, it *exists* in that schema's target namespace, http://test.com. The testHello element appears correctly, according to the schema it is declared in:

```
...<p585:testHello xmlns:p585="http://test.com">...
```

Notice that testHello is prefixed with the string p585 and that p585 is set to namespace http://test.com.

Notice in the testHello element's anonymous complexType definition that the child element, name, is declared. Remember that according to the error message, the Web services engine complains that it cannot deserialize element name. Example 40 shows how the name element appears in the SOAP request.

Example 40 testHello element in SOAP request

```
…
      <p585:testHello xmlns:p585="http://test.com">
         <p585:name>Wendy</p585:name>
      </p585:testHello>
      ...
```

Notice that name is qualified, too, with prefix p585 that resolves to namespace http://test.com.

Note: The W3C recommendation, "XML Schema Part 0: Primer Second Edition," says the following about qualifying local elements:

> Qualification of local elements and attributes can be globally specified by a pair of attributes, elementFormDefault and attributeFormDefault, on the schema element, or can be specified separately for each local declaration using the form attribute. All such attributes' values may each be set to unqualified or qualified, to indicate whether or not locally declared elements and attributes must be unqualified.

In this case, name is a local element because it is not declared within the context of the schema — it is declared within a complexType definition. Because the schema in which `name` is declared does not explicitly have the elementFormDefault attribute, the default of *unqualified* is assumed. Therefore, name should not be qualified in the SOAP message.

Example 41 shows how you should not specify name (qualified with p585).

Example 41 Incorrect specification of name

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <p585:testHello xmlns:p585="http://test.com">
         <p585:name>Wendy</p585:name>
      </p585:testHello>
   </soapenv:Body>
</soapenv:Envelope>
```

Example 42 shows the correct way to specify name (unqualified).

Example 42 Correct specification of name

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <p585:testHello xmlns:p585="http://test.com">
         <name>Wendy</name>
      </p585:testHello>
   </soapenv:Body>
</soapenv:Envelope>
```

Alternatively, the use of a default namespace (Example 43) is another way that the message can be structured so that name is in the wrong namespace.

Example 43 Incorrect specification using the default namespace

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <testHello xmlns="http://test.com">
         <name>Wendy</name>
      </testHello>
   </soapenv:Body>
</soapenv:Envelope>
```

The appearance of the xmlns= attribute on parent element testHello indicates that the default namespace is http://test.com. Therefore, its child element name inherits this default namespace. But again, according to the schema, name does not have a namespace.

Instead, the message should appear as shown in Example 44 to be valid.

Example 44 Correct specification using the default namespace

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <testHello xmlns="http://test.com">
         <name xmlns="">Wendy</name>
      </testHello>
   </soapenv:Body>
</soapenv:Envelope>
```

xmlns="" indicates that name exists in no default namespace at all.

The only way to resolve the WSWS3047E error is to fix the message so that local elements are qualified or unqualified, according to the schema. Additionally, whether elements are qualified using a prefix or default namespace, the

namespace must be the correct one. You should work with the Web services vendor to accomplish this task.

Support preparation

If you still observe the WSWS3047E error after fixing the message, collect the following data for use by IBM support:

- Capture of SOAP request and response. These can be obtained using the TCPMonitor tool. (See “Collecting trace data with TCPMonitor” on page 45.)
- Web services engine trace reflecting the "`org.xml.sax.SAXException: WSWS3047E: Error: Cannot deserialize element <element_name> of bean <bean_name>`" error. See “Enabling the Web services trace” on page 46 for information about collecting the trace.
- Complete test case. If you cannot provide the test case, then the WSDLs with all referenced schema files are needed.

Go to “The next step” on page 47 for information about performing an online search of Web services problems and for a list of useful technical articles. If these resources do not help you resolve the problem, contact IBM support.

Collecting diagnostic data

This section provides information for collecting diagnostic data useful in diagnosing Web services problems.

Collecting JVM logs

JVM logs, often referred to as SystemOut and SystemErr logs, are created for every WebSphere Application Server process (application server, cluster member, node agent, and deployment manager). They can be found in the following locations:

- WebSphere Application Server V6.x for z/OS

  The JVM logs are located in the address space output. Generally, the SYSPRINT card correlates to SystemOut.log, and SYSOUT correlates to SystemErr.log.

- WebSphere Application Server V6.x (distributed and i5/OS)

  The JVM log files are by default named SystemOut.log and SystemErr.log. The default location for the SystemOut and SystemErr logs is:

  – *profile_root*/logs/*server_name*/SystemOut.log
  – *profile_root*/logs/*server_name*/SystemErr.log

The location of application server logs is configurable.

a. Select **Troubleshooting** → **Logs and Trace** in the navigation bar.
b. Click the server name.
c. Select **JVM logs**.

This page shows the location of the log file.

Collecting trace data with TCPMonitor

You can use the TCPMonitor tool to observe and capture SOAP messages exchanged between a Web service client and Web service provider. TCPMonitor redirects messages from a port, records the messages, and forwards the messages to another port.

1. In the WebSphere bin directory, create a tcpmon script specific for your platform (a batch script for Windows® and a shell script for UNIX®).
   - For a Windows script, the file would contain the following:

     ```
     call "setupCmdLine.bat"
     %JAVA_HOME%\bin\java
     -Djava.ext.dirs=%WAS_EXT_DIRS%;%WAS_HOME%\plugins
     com.ibm.ws.webservices.engine.utils.tcpmon
     ```

   - On a UNIX system, the file would look like the following:

     ```
     . ./setupCmdLine.sh
     ${JAVA_HOME}/bin/java
     -Djava.ext.dirs=${WAS_EXT_DIRS}:${WAS_HOME}/plugins
     com.ibm.ws.webservices.engine.utils.tcpmon
     ```

2. Run the script in a command-line console. A GUI should appear.
3. In the GUI, specify the following:
   - Listen Port

     Port that you want TCPMonitor to listen on. This port name should be unused.

   - Target Hostname

     Hostname of Web service provider.

   - Target Port

     Port of the Web service provider.

4. Click **Add**.

   TCPMonitor is now ready to intercept SOAP requests and responses.

   The Web service client should now redirect its request through TCPMonitor. The new endpoint URL given to the client should contain the host name

where TCPMonitor is running and the listen port specified in the TCPMonitor Listen Port text field.

For example, consider the following case:

– The Web service provider is accessible at endpoint http://hostA:9080/Hello/HelloWorld.

– TCPMonitor is running on hostB and is listening on port 9085.

To redirect the client request through TCPMonitor, point the client to endpoint http://hostB:9085/Hello/HelloWorld.

5. Invoke the client. You should observe both the SOAP request and response in the two panels of the GUI. The messages can be saved to a text file by clicking the **Save** button.

Enabling the Web services trace

You can enable and gather Web services runtime trace for an unmanaged client, a managed client running in the J2EE client container, managed clients running on the Web or EJB containers, or Web service providers running on the Web or EJB containers.

Enabling and gathering trace for an unmanaged client

To enable and gather trace for an unmanaged client, do the following:

1. Create a trace properties file by copying the *app_server_root*\properties\TraceSettings.properties file to the same directory as your client application Java archive (JAR) file.
2. Edit the properties file and change the traceFileName value to the location for the trace data output (for example, traceFileName=c:\\temp\\myAppClient.trc).
3. Edit the properties file to remove com.ibm.ejs.ras.*=all=enabled and add `com.ibm.ws.webservices.*=all=enabled`.
4. Add the following options to the Java command line that is used to run the client:

   ```
   -DtraceSettingsFile=trace_properties_file
   -Djava.util.logging.manager=com.ibm.ws.bootstrap.WsLogManager
   -Djava.util.logging.configureByServer=true
   ```

   Where *trace_properties_file* represents the name of the properties file that you created in the previous steps. For example:

   ```
   java -DtraceSettingsFile=TraceSettings.properties
   -Djava.util.logging.manager=com.ibm.ws.bootstrap.WsLogManager
   -Djava.util.logging.configureByServer=true myApp.myAppMainClass
   ```

Enabling and gathering trace data for a managed client

To enable and gather trace for a managed client running in the J2EE client container, invoke the launchClient command-line tool with the following options:

```
-CCtrace=com.ibm.ws.webservices.*=all=enabled
-CCtracefile=traceFileName
```

For example:

```
app_server_root\bin\launchClient MyAppClient.ear
-CCtrace=com.ibm.ws.webservices.*=all=enabled
-CCtracefile=myAppClient.trc
```

Enabling the trace from the administrative console

To enable and gather trace for managed clients running in the Web or EJB container, or Web service providers running in the Web or EJB containers, do the following:

1. In the administrative console, expand **Servers** → **Application Servers** → *server_name*.
2. Select **Diagnostic Trace Service**.
3. Set Maximum File Size to 200 MB and select the **File radio** button in the General properties section. Set the appropriate number of historical trace files.
4. Navigate to **Change Log Detail Levels** in the Additional properties section.
5. Clear the current trace string and add the following trace string to the General properties field for general Web services issues:

   ```
   *=info:com.ibm.ws.webservices.*=all
   ```

By default, the trace is logged to a file called trace.log in the same location as SystemOut.log and SystemErr.log, *profile_root*/logs/*server_name.*

Disabling the trace

To restore trace state back to normal, use this same process and clear the trace string. Restart the server.

The next step

The symptoms and problem areas included in this activity are some that you are more likely to experience. However, there are other things that can go wrong.

Search online support

If you are sure that the problem is a result of developing, deploying, or executing a Web services application, there are tasks that you can do before contacting IBM support. First, you should review the documentation that you have gathered for errors that were not addressed in this paper and search support sites for information or fixes. Look for current information available from IBM support on known issues and resolutions on the following IBM support page:

http://www-1.ibm.com/support/search.wss?rs=180&tc=SSEQTP&tc1=SSCR4XC

Review troubleshooting articles

Look also at the WebSphere Information Center documentation for additional resources for diagnosing and fixing issues with Web services applications. The following list provides the URL for the distributed platform Network Deployment topic. If you are using a different package or platform, you can find the comparable topic by using the title (enclosed in double quotation marks) as a search argument.

- *Troubleshooting Web Services*

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/twbs_troubleshootwbs.html

  This topic includes a number of links to sub-topics, both within the text and at the bottom, to specific diagnostic information for different types of Web services problems.

- *Web services serialization and deserialization troubleshooting tips*

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/rwbs_trbserialize.htm

- *Trace and logging for WSIF*

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/twsf_tracelog.html

- *UDDI registry troubleshooting*

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/twsu_probdet.html

- *Universal Discovery, Description, and Integration, Web Service, and SOAP component troubleshooting tips*

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/rtrb_svsccomp.html

- Explanation of messages

  The WebSphere information center has a message reference that provides message text and user response suggestions for each message prefix type.

  The information center is at:

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp

  Navigate to ***WebSphere package*** → **Reference** → **Messages** → ***message_prefix***.

Resources for learning

For general information about Web services applications for WebSphere Application Server, see:

- *Web services* in the WebSphere Information Center, including links to tutorials, samples, and resources for learning:

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/welc6tech_wbs.html

- *Web Services Handbook for WebSphere Application Server 6.1*

  http://www.redbooks.ibm.com/abstracts/sg247257.html

Understanding the following specifications is important when diagnosing problems with Web services applications:

- The JAX-RPC 1.1 specification

  http://developers.sun.com/techtopics/webservices/reference/api/index.html

- The Web services for J2EE specification (JSR 109)

  http://jcp.org/en/jsr/detail?id=109

- Basic Profile Version 1.1

  http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

  This specification describes how Web services should behave in order to interoperate with each other. It is key in solving problems that arise from scenarios where the client and provider are on different Web services vendors (one on WebSphere and another on .Net, for example).

- SOAP 1.1

  http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

  This specification describes how SOAP 1.1 messages should appear.

- SOAP with Attachments API for Java™ (SAAJ) Specification 1.1

  https://saaj.dev.java.net/

  SAAJ is an alternative programming model to JAX-RPC.
- Web Services Description Language (WSDL) 1.1

  http://www.w3.org/TR/wsdl

  This outlines how a compliant WSDL should be structured.
- XML Schema

  http://www.w3.org/2001/XMLSchema

Contact IBM

If these steps do not resolve your problem, gather additional information and raise a problem record with IBM. If instructions on what to gather were not included for your particular problem area, use the MustGather information for your problem to collect documentation: MustGather: Web services engine and tooling problems for WebSphere Application Server V6.1, V6, V5.1 and V5.

http://www-1.ibm.com/support/docview.wss?rs=180&context=SSCR4XC&q1=MustGatherDocument&uid=swg21198363&loc=en_US&cs=utf-8&lang=en

The following URL contains a list of all of the MustGather documentation for Web services problems involving the Web services engine, WSIF, or Web services gateway:

http://www-1.ibm.com/support/search.wss?rs=180&tc=SSEQTP&tc1=SSCR4XC&q=MustGatherDocument

The following URL contains a list of the MustGather documentation for WS-Security:

http://www-1.ibm.com/support/search.wss?rs=180&tc=SSEQTP&tc1=SSCNPY4&q=MustGatherDocument

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM application programming interfaces.

This document REDP-4306-00 was created or updated on May 22, 2007.

Send us your comments in one of the following ways:
- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099, 2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Redbooks (logo) ®	z/OS®	Rational®
i5/OS®	IBM®	WebSphere®

The following terms are trademarks of other companies:

EJB, Java, JVM, J2EE, SOAP with Attachments API for Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.