Dynamic Web service client

A dynamic client for publishing, discovering and invoking Web services

Skill Level: Intermediate

Rosaline Makar (rosaline@eg.ibm.com)
Software Engineer
IBM

Mahmoud Ouda (mouda@eg.ibm.com)
Software Engineer
IBM

22 Sep 2006

Learn how you can significantly speed the sometimes tedious and repetitive tasks of service publishing and inquiry with the dynamic Web service client. This all-in-one bundle encompasses components for UDDI registries, including publish and inquiry, using UDDIV3Client API, and Web services.

Section 1. Before you start

About this tutorial

This tutorial introduces a set of reusable components for publishing, inquiry and dynamic invocation of Web services. You'll see how these components were integrated to compose the Dynamic Web service client, also referred to as DWSC.

Objectives

The DWSC component provides a simple interface to:

1. Publish a Web service to a UDDI registry
2. Discover an already published Web service
3. Dynamically invoke a published Web service

The publishing and discovery are carried out using UDDI v3.0 Client API.

Prerequisites

You should have a basic knowledge of SOA, Web services, and Java technology. Basic technical experience in products listed in the system requirements section is required.

System requirements

In order to complete this tutorial you need the following list of tools:

1. IBM® WebSphere® Application Server v6.0 or later, or a test environment
2. DB2® Universal Database v8.2 or later. The DWSC may work on lower versions too.
3. IBM Rational® Application Developer v6.0 or later

Please refer to the Resources section for links to trial versions.

Section 2. Introduction

Technologies such as Service-Oriented Architecture (SOA) have evolved in answer to the increased demand on heterogeneous systems. This is a paradigm shift that enables complex criteria such as loose coupling, location transparency, interoperability, scalability and platform independence. Thus, SOA fulfills key requirements for today's business needs. Web services technology is a widely-accepted implementation of SOA. This tutorial presents a tool that encompasses components for publishing, discovery and dynamic invoking of Web services.

The DWSC provides a simplified interface to:

1. Publish a Web service to a UDDI registry
2. Discover an already published Web service
3. Dynamically invoke a Web service

The ordinary publish, discover, connect and invoke cycle is listed below. You might be familiar with it if you are involved in Web services development.

1. The service provider creates a Web service, and publishes its interface and access information to the service Registry (UDDI).
2. The service registry makes the Web service interface and implementation access information available to potential service requestors.
3. The service requestor locates entries in the registry using various find operations and then binds to the service provider in order to invoke one of its Web services.

**Figure 1. Web services components**

The Web service client component replaces the previous service requestor role as shown in Figure 2.

**Figure 2. Dynamic Web service client**
As Figure 2 shows, it is assumed that a service has already been published on a UDDI. You can publish one using the UDDIClient box (shown later in this tutorial).

1. The user supplies a service name through the Web service client
2. The UDDIClient inquires the service name within the target UDDI
3. Target UDDI responds with the corresponding WSDL URL (Pointer to a valid WSDL Document as published by the service provider)
4. Using a WSDL2Java wrapper, the WSDL document is parsed and the corresponding Java code is generated and compiled
5. The Reflector applies Reflection on the generated Java classes to extract a list of invokable operations
6. The Web service client lists the invokable operations
7. User chooses the desired operation to invoke

Section 3. DWSC components
We’ll cover the three components of the dynamic Web service client shown in Figure 2.

- **UDDIClient component**
- **WSDL2Java component**
- **Reflector component**

**UDDIClient component**

This component can be used to publish and discover Web service to and from a UDDI registry using UDDI V3.0 client API

**Constructor:** public UDDIClientImpl()

**Interface methods:**

**Discovering Web service**

- public void initializeInquiryPorts(UDDIInfo uddiInfo), where *uddiInfo* is a bean holding WebSphere Application Server configuration
- public URI[] getBusiness(String businessName), where *businessName* is the business name to retrieve business key for
- public Service[] getServices(URI businessKey), where *businessKey* is the business key to retrieve services for
- public BindingTemplate[] getBindingTemplate(URI serviceKey), where *serviceKey* is the service key to retrieve binding templates for
- public OverviewDoc[] getOverviewDoc(TModel tmodel), where *tmodel* is the TModel to retrieve overview documents for
- public String getOverviewURL(OverviewDoc overviewDoc), where *overviewDoc* is the overview document to get WSDL URL for

**Publishing Web service**

- public void PublishService(AuthToken authToken, String businessName, String serviceName, String tModelName, String accessPointValue, String wsdlURL)
- public void PublishService(AuthToken authToken, String businessName, String serviceName)
- public void PublishService(AuthToken authToken, String businessName, String serviceName, String tModelName)
WSDL2Java component

Generate Java skeletons from a WSDL document using the WSDL2JavaGenerator.

Constructor: public WSDL2JavaGenerator(String wsdlURL, String libPath, String targetPath), where wsdlURL is a valid URL for a WSDL document, targetPath is where the java skeletons will be generated, whereas libPath is the path of the libraries required to generate the Java sources.

Interface methods:

Generate Java source code from a WSDL document

- public void generateJavaSource(boolean overwrite), where overwrite causes this method to overwrite old generated java code if exists.

Reflector component

Reflects a currently compiled class, decomposing it to its public methods.

Constructor: public JavaReflector(String path, boolean recursive), where path is the root path where reflection starts and recursive causes recursion in subdirectories.

Interface methods:

Retrieve class info

- public ArrayList retrieve(int typeFlag, int shallowFlag), where this method reflects classes within a specific directory, and returns an ArrayList of invokable Operations.

Invoke a service operation

- public static Object invoke(Operation operation), invokes a Web service operation and returns the operation result.
  Note: This method is defined in class Invoker.

Section 4. DWSC use cases

We will discuss the different use cases of the dynamic Web service client Which include:

- System Administrator use case
• Service Provider use case
• Service Requestor use cases

System Administrator use case

Figure 3. System Administrator use case

The System administrator configures UDDI on the Websphere Application Server.

Service Provider use case

Figure 4. Service Provider use case

The Service Provider publishes Web services to the UDDI using the business name, the service name and the WSDL URL.

Service Requestor use cases

Figure 5. Service Requestor use cases
• Generate Web service client from a WSDL use case: Given a valid WSDL document URL, the corresponding Java classes are generated.

• Compile the generated Java sources use case: Given a directory name; all Java files within that directory are compiled to produce Java classes.

• Reflector use case: Given a service interface; all invokable service operations are extracted.

• Invoke Web service operation use case: Invokes a service operation and waits for a return (if it exists)

We've made a pretty JSP client that uses the DWSC, Figure 6 shows the sequence diagram for it. Details for the JSP client will be shown later in this tutorial.

**Figure 6. Putting it all together**
Section 5. Environment preparation

To set up the dynamic Web service environment, we'll go through these steps:

1. Create new WebSphere Application Server profile
2. Create a DB2 distributed Database for UDDI registry
3. Create data source for the UDDI registry
4. Deploy UDDI registry Application
5. Set up library folder
6. Set properties file values

1. Create new WebSphere Application Server profile
   - Go to WAS\Install_root\profiles\default\firststeps
   - Run firststeps.bat
   - Click on Profile creation wizard as shown in Figure 7

Figure 7. WebSphere Application Server first steps
First steps

- **Installation verification**
  Confirm that your server is installed and that it can start properly.

- **Start the server**
  Start the server and its applications.

- **Administrative console**
  Install and administer applications.

- **Profile creation wizard**
  Create a profile.

- **Information center for WebSphere Application Server**
  Learn more about WebSphere Application Server.

- **Migration wizard**
  Migrate WebSphere Application Server V4 or V5 to V6.0.

- **Exit**

Edit profile name, and make it the default profile then click **Next** as shown in Figure 8.

**Figure 8. Profile Name**
Set the profile directory, such that the path is as short as possible and does not contain white spaces, then click Next as shown in Figure 9.

**Figure 9. Profile directory**

Observe the node name and save it in a text file for later use, then click Next as
shown in Figure 10.

**Figure 10. Node and host names**

![Profile creation wizard](image)

Edit the values in the text boxes to be just like the default values given in the left hand side, then click **Next** as shown in **Figure 7**.

**Figure 11. Port value assignment**
Uncheck **Run the application server as a Windows service**, then click **Next** as shown in Figure 12.

**Figure 12. Windows service definition**
Now the wizard should finish successfully.

2. Create a DB2 distributed Database for UDDI registry

Please refer to Creating a DB2 distributed Database for UDDI registry. Don't go through the whole set of tasks, only "Setting up a customized UDDI node" is needed, initializing UDDI node and other tasks are not.

3. Create a data source for the UDDI registry

Go to WAS_profile_root(for example C:\WAS2\server2), go to firststeps folder, click on firststeps.bat.

Click on start the server, and after the server is started successfully, click on Administrative console.

Expand Security as shown in Figure 13.
Select **Global Security**.

Expand **JAAS Configuration** (on the right).

Click on **J2C authentication**.

**Figure 13. Security setup**

![Security setup diagram]

Click **New**.

Fill the following Details as shown in Figure 14:

- **Alias**: A short name, e.g. UDDIAlias
- **User ID**: The database user ID
- **Password**: The Password for the above ID
- **Description**: Can be anything

**Figure 14. J2C authentication alias setup**
Click **Apply** and then **Save** the changes to the master configuration.

Expand **Resources**.

Select **JDBC Providers**.

Select **Server: server1** to create JDBC provider on server level as shown in Figure 15.

Click **New**.

**Figure 15. JDBC provider setup**
Fill the following details as shown in Figure 16:

- **Database**: DB2
- **Provider Type**: DB2 Legacy CLI-based Type 2 JDBC Driver
- **Implementation Type**: Connection Pool data source
- Save the changes

**Figure 16. JDBC provider configuration**

Expand **Environment**.

Select **Websphere Variables**.

Click on **DB2_JDBC_DRIVER_PATH**.

As shown in Figure 17, edit the **value** to be `DB2installRoot\java`, Note that the default DB2 install root on Windows platform is: C:\program files\IBM\SQLLIB, so a candidate value would be C:\program files\SQLLIB\IBM\java.

Save the changes.

**Figure 17. JDBC environment variable setup**
Create the data source for the UDDI registry:

1. Click **Resources** then **JDBC providers**
2. Select **DB2 Legacy CLI-based Type 2 JDBC Driver**
3. Under **Additional Properties**, select **Data Sources** (not the **Data sources (Version 4)** option)
4. Click **New** to create data source
5. Fill the details as described in the Figure 18, note that the **JNDI name** has to be **datasources/uddids**
6. Then click on **apply** and save the changes to the master configuration.
7. To make sure that the data source can connect to the database correctly, click on **Test Connection**

**Figure 18. Data source setup**
4. Deploy UDDI registry Application

Run **uddiDeplpy.jacl** script to deploy UDDI registry application, Please refer to
Deploying the UDDI registry application

After the script has successfully completed, restart the Server, open the administrative console and check that the UDDI appears in the UDDI nodes list as shown in Figure 19.

Figure 19. UDDI nodes

5. Set up libraries folder

Copy the following files from the `<WAS_Install root>\lib` to `C:\LibsFolder`:

- webservices.jar
- bootstrap.jar
- ras.jar
- commons-logging-api.jar
- commons-discovery.jar
- wccm_base.jar
- emf.jar
- j2ee.jar
- deployutils.jar
- wsd14.jar
- classloader.jar
- ffdc.jar
- wsexception.jar

In addition to xercesImpl.jar, xml-apis.jars, available from the Downloads section. They are also downloadable from apache.org.

6. Set properties file values

Fill in your environment specific properties in `config.properties` file and place the properties file in the WebSphere Application Server profile folder (the place where you just created the new WAS profile), as shown here:
**Targetpath:**

- If you are deploying on **Websphere Application Server without using an IDE** then the TargetPath is the concatenation of `<WAS_profile>\installedapps\<nodename>\<earName>\<warName>\WEB-INF\classes`.
- If you are **using Rational application developer** then the TargetPath is `<Workspace>\DynamicWebServiceClient\WebContent\WEB-INF\classes`.

**libpath:** full path of a folder where you will place the dependency libs. for example, "C:\LibsFolder"

Check out the sample `config.properties` file.

### Listing 1. sample config.properties

```plaintext
#Start config.properties
#Assume the new WAS profile was created under c:\was\was1
#C:\was\was1\config.properties: this file

#Case: Deployment on WAS
targetpath=C:\WS2\Server2\installedApps\IBM-FEB839BA8FANode02Cell\DynamicWebServiceClientEAR.ear\DynamicWebServiceClient.war\WEB-INF\classes

#Case: Development environment
targetpath=C:\\RAD-WS\\DynamicWebServiceClient\\WebContent\\WEB-INF\\classes

#Required libraries path
libpath=C:\ab\lib\

#End
```

---

### Section 6. Running the DWSC

In this section we will discuss two different ways to run the dynamic Web service client:

1. **Deploying on WebSphere Application Server version 6.0 and higher**
2. **Running inside Rational Application Developer**

**Deploying on WebSphere Application Server version 6 and higher**

**Deploy the dynamic Web service client**
1. Go to <WAS_profile root>(for example C:\WAS2\server2), go to firststeps folder and run firststeps.bat.

2. If the server is not started yet, click on start the server, and after the server is started successfully, click on Administrative console.

3. Expand Applications on the left, then click on Install New Applications as shown in Figure 20.

4. In the Local file System, browse to DynamicWebServiceClientEAR.ear, and then click Next until you reach the finish button.

Figure 20. Installing EAR on application server

5. When the EAR is deployed successfully, click on Save to master Configuration as shown in Figure 21

6. Now, make sure that the EAR is running on the server

Figure 21. Saving changes of successfully installed EAR
Deploy the Web service

Repeat the previous steps with HelloWebServiceEAR.ear found in the Download section in order to deploy the Web service on the application server which represents the service provider.

Publish Web service to the UDDI.

Running inside Rational Application Developer

1. Launch Rational Application Developer.
2. Create a new workspace without white spaces in the workspace location.
3. Import the project Interchange HelloWebService.zip and DynamicWebServiceClient.zip. Both can be found in the Download section.
4. You will need uddiv3client.jar to compile this project. You'll find it located under <WAS_Install root>\lib.
5. Publish the Web service to the UDDI by running PublishTest.java after editing the values in it according to your WebSphere Application Server configuration.
6. Make sure you are running the new server profile created in step 1 of the
"Environment preparation" section.

To run the dynamic Web service client, open the Web browser and type:

http://localhost:9080/DynamicWebServiceClient/start

Section 7. How does it work?

The dynamic Web service client GUI is composed of five JSPs:

1. Init.jsp
2. Services.jsp
3. Operations.jsp
4. OperationsIO.jsp
5. Result.jsp

Figure 22. The relations between client JSPs

Init.jsp shown in Figure 23 is the starting page of the dynamic Web service client.

Figure 23. Init.jsp
The fields in Init.jsp that are used to find services inside certain business name:

- **Business Name**: The name of the business in which the Web service resides.
- **WAS User Name**: The WebSphere application server user name.
- **WAS Password**: The WebSphere application server password.
- **Inquiry URL**: The inquiry port is usually set as the default value in the previous figure.
- **Security Non-Secure URL**: The non secure port URL, it is usually set as the default value in Figure 24.

**If WebSphere Application Server security is turned ON, there are other fields**
that have to be filled:

- **Security Secure URL**: UDDI security port
- **WAS Profile Root**: The WAS profile URL

You can retrieve **Trust Store dir Path, Trust Store file Name, Trust Store Certificate, Trust Store Password, Key Store Dir Path, Key Store File Name, Key Store Password**: from the administrative console of Websphere Application Server as shown in Figure 24:

- Open the Administrative Console
- Go to Security, then SSL, and click on the DefaultSSLSettings.

**Figure 24. DefaultSSLSettings**
Services.jsp shown in Figure 25, gets its parameters from Init.jsp, and displays the available services (WSDL URLs) inside the given Business Name.

Figure 25. Services.jsp
Operations.jsp shown in Figure 26 gets the selected WSDL URL from Services.jsp, and displays the available operations in the specified Web service.

**Figure 26. Operations.jsp**

OperationIo.jsp shown in Figure 27 gets the selected Operation from Operations.jsp, and displays the input parameters of the specified operation.
Figure 27. OperationsIO.jsp

Result.jsp shown in Figure 28 gets the input parameters from OperationIO.jsp, processes them, invokes the operation, and displays the result.

Figure 28. Result.jsp
Section 8. Limitations

The APIs do not support Web services with array type input parameters. Web services that rely on primitive types (such as boolean, int, or double) are not supported either. Instead you may use Object wrappers such as Boolean, Integer, or Double.

Section 9. Acknowledgement

The authors would like to thank Ghaly Stefanos, advisory IT architect, for his
supervision of this article.

Section 10. Appendix

We will discuss in this section the key packages of the dynamic Web service client.

**com.ibm.eg.java**

- CommandExec class: A utility class that executes a command in a thread

**com.ibm.eg.java.util**

- ClassFilter class: A utility class that filters an array of class objects according to certain criteria. Currently, only the INHERITS_FILTER criterion is implemented. i.e. if a class implements certain class(es) it passes the filterClasses method.
- JavaReflector class: Reflects a currently loaded class.
- Operation class: Encapsulation of a port type operation.
- Param class: Encapsulation of a method parameter.
- JavaCompiler: A utility class to compile java code.
- Invoker: Invokes an initialized Operation object.

**com.ibm.eg.soa.webservices**

- WSDL2JavaGenerator class: Wraps WSDL2Java (generates Java skeletons for a WSDL document).
- WSDLExplorer class: Façade for the above packages. No need to instantiate any of the classes.

**com.ibm.eg.soa.uddi.test**

- FindTest class: is a tester class to find a service
- PublishTest class: is a tester class to publish a service

**com.ibm.eg.soa.uddi**

Service class: is a javabean that contains service parameters, It has the following attributes:

- Service Name
• Service Key: it is an autogenerated key resulted from publishing the service.

• Business Key: it is an autogenerated key resulted from publishing the business.

UDDIInfo class: is a javabean that contains parameters needed for publishing and finding a Web service. It has the following attributes:

• Security URL Non Secure
• Security URL Secure
• Inquiry URL
• Publish URL Non Secure
• Publish URL Secure
• Trust Store DirPath
• Trust Store Filename
• Trust Store Certificate
• Trust Store Password
• Key Store DirPath
• Key Store Filename
• Key Store Password
• User Install Root
• User Name
• password

UDDIClient interface: contains methods signatures needed for publishing, finding and deleting a Web service from UDDI registry.

UDDIClientImpl class: implements UDDIClient interface

**Figure 29. com.ibm,eg,soa,uddi classes**
# Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
<th>Download method</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWSC_Downloads.zip</td>
<td></td>
<td>10KB</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

Information about download methods
Resources

Learn

• Read "Understanding WSDL in a UDDI registry" (developerWorks, September 2001) for an introduction to using WSDL with UDDI registries.
• Read about Web Services Description Language (WSDL) 1.1 from W3C.
• See how to set up a customized UDDI node.
• See how to deploy the UDDI registry application.
• Learn about the IBM UDDI Version 3 Client for Java.
• Read the IBM Redbook "WebSphere Version 6 Web services Handbook Development and Deployment".

Get products and technologies

• Download a free trial version of WebSphere Application Server Version 6.0.
• Download a free trial version of Rational Application Developer for WebSphere Software.
• Download a free trial version of DB2 Universal Database Enterprise.

About the authors

Rosaline Makar
Rosaline Makar is a software engineer at IBM Egypt Cairo Technology Development Center (C-TDC). Her current work areas include Service-Oriented architecture (SOA) and Web services. She has a B.Sc. in Computer Engineering and currently preparing for a M.Sc. in Computer Science.

Mahmoud Ouda
Mahmoud Ouda is a software engineer at IBM Egypt Cairo Technology Development Center (C-TDC). His main interests are SOA, Web services, Security and e-business solutions. Ouda has a B.Sc. in Computer Science and is currently preparing for a M.Sc. in Computer Science.