UNIT-V

WEB SERVICES

Definition of Web Services

A Web Service is a standards-based, language-agnostic software entity that accepts specially formatted requests from other software entities on remote machines via vendor and transport neutral communication protocols, producing application specific responses.

- Standards based
- Language agnostic
- Formatted requests
- Remote machines
- Vendor neutral
- Transport neutral
- Application specific responses

Web Services

- Web services are application components
- Web services communicate using open protocols
- Web services are self-contained and self-describing
- Web services can be discovered using UDDI
- Web services can be used by other applications
- HTTP and XML is the basis for Web services

Web Services have Two Types of Uses

Reusable application-components.

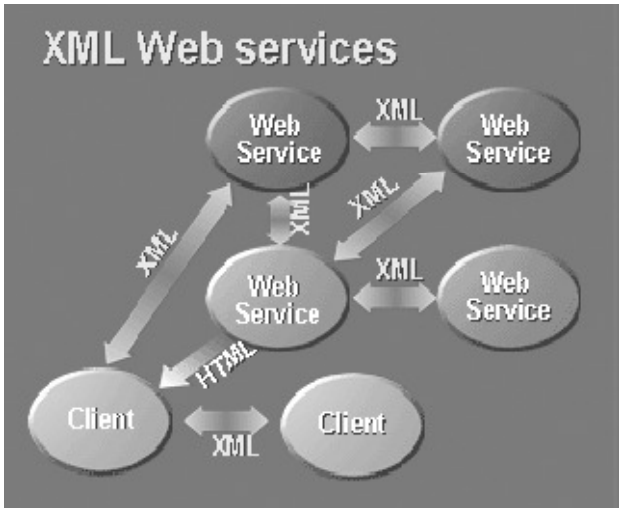
Web services can offer application-components like: currency conversion, weather reports, or even language translation as services.

Connect existing software.

Web services can help to solve the interoperability problem by giving different applications a way to link their data.

With Web services you can exchange data between different applications and different platforms.

ROLE OF WEB SERVICES IN DISTRIBUTED COMPUTING



Enabling Technologies

To achieve the inter-operable loose coupling that Web Services requires, there must be a set of standards that govern how communication between systems takes place. The likely direction for a language to describe services and to encapsulate messages is the Extensible Markup Language (XML).

Developers using XML can take advantage of a common parser that drastically reduces development time. With the current hype surrounding XML it is hard to believe that there are alternative languages, but one such alternative is the DAML family of languages built around the RDF specifications of the W3C.

DAML and RDF deliver advantages over XML such as description logic and inheritance. The likely technology for the transport of XML messages over the Internet is the Simple Object Access Protocol (SOAP).

SOAP is a better way to manage remote invocation over the Internet than distributed technologies such as DCOM, RMI and IIOP because it packages XML messages into "envelopes" and sends them over standard internet protocols such as HTTP and SMTP, taking advantage of well defined data formats and making it easier to operate over firewalls, which are normally already set up to cope with the standard HTTP port 80.

To implement discovery, there needs to be a way to define and query registries of Web Services. Universal Description, Discovery and Integration (UDDI) are such a mechanism. It uses SOAP messaging to publish, edit, browse and search a registry. The Web Services Description Language (WDSL) is an XML standard for describing a Web Service.

WSDL: Describing Web Services

The Web Services Description Language (WSDL) is an XML schema format that defines an extensible framework for describing Web services interfaces. WSDL was developed primarily by Microsoft and IBM and was submitted to W3C by 25 companies.

WSDL is at the heart of the Web services framework, providing a common way in which to represent the data types passed in messages, the operations to be performed on the messages, and the mapping of the messages on to network transports.

WSDL is, like the rest of the Web services framework, designed for use with both procedure-oriented and document-oriented interactions. As with the rest of the XML technologies, WSDL is so extensible and has so many options that ensuring compatibility and interoperability across differing implementations may be difficult. If the sender and the receiver of a message can share and understand the same WSDL file the same way, however, interoperability can be ensured.

WSDL is the XML format that describes what a Web service consists of

WSDL is divided into three major elements:

- Data type definitions
- Abstract operations
- Service bindings

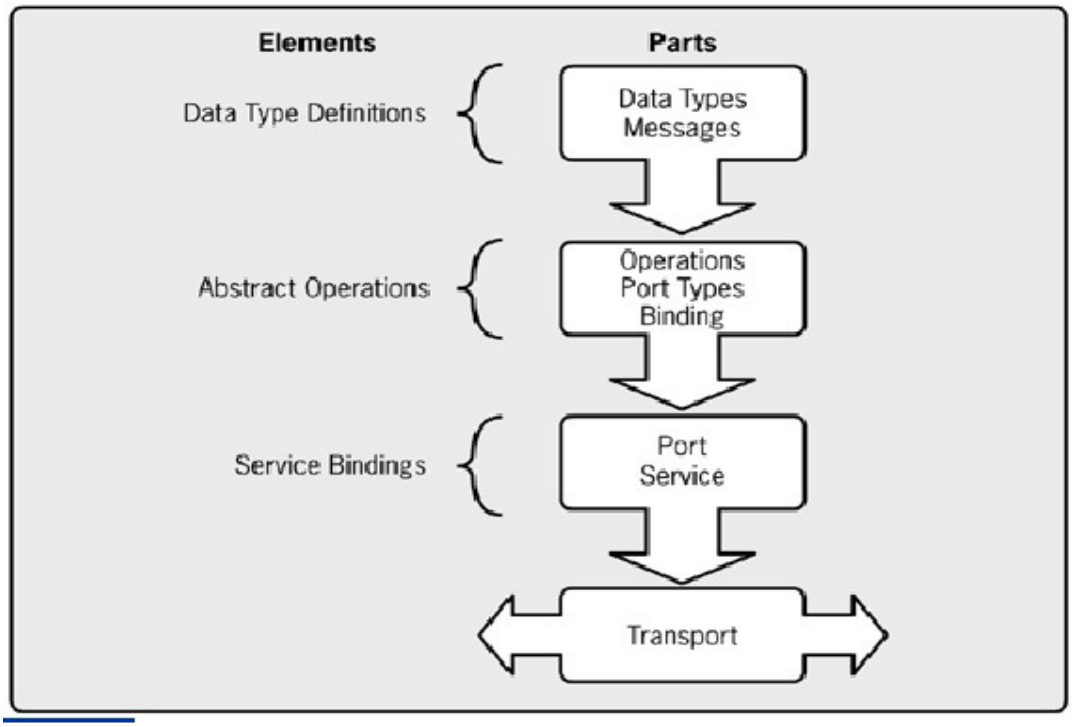
WSDL has three major elements, according to level of abstraction

Each major element can be specified in a separate XML document and imported in various combinations to create a final Web services description, or they can all be defined together in a single document. The data type definitions determine the structure and the content of the messages. Abstract operations determine the operations performed on the message content, and service bindings determine the network transport that will carry the message to its destination.

WSDL elements can be defined in separate documents

Figure shows the elements of WSDL, layered according to their levels of abstraction, which are defined independently of the transport, specifically so that multiple transports can be used for the same service. For example, the same service might be accessible via SOAP over HTTP and SOAP over JMS. Similarly, data type definitions are placed in a separate section so that they can be used by multiple services. Major WSDL elements are broken into subparts.

Figure: WSDL consists of three major elements and seven parts.



The definition parts include data type definitions, messages, and abstract operations, which are similar to interface definitions in CORBA or DCOM. Messages can have multiple parts and can be defined for use with the procedure-oriented interaction style, the document-oriented interaction style, or both. Through the abstraction layers, the same messages can be defined and used for multiple port types. Like the other parts of WSDL, messages also include extensibility components—for example, for including other message attributes.

WSDL interfaces are like CORBA or DCOM interfaces

WSDL data type definitions are based on XML schemas, but another, equivalent or similar type definition system can be substituted. For example, CORBA Interface Definition Language (IDL) data types could be used instead of XML schema data types. (If another type definition system is used, however, both parties to a Web services interaction must be able to understand it.)

Web service data types are based on XML schemas but are extensible to any other mechanism

The service bindings map the abstract messages and operations onto specific transports, such as SOAP. The binding extensibility components are used to include information specific to SOAP and other mappings. Abstract definitions can be mapped to a variety of physical transports. The WSDL specification includes examples of SOAP one-way mappings for SMTP (Simple Mail Transfer Protocol), SOAP RPC mappings for HTTP, SOAP mappings to HTTP GET and POST, and a mapping example for the MIME (multipurpose Internet messaging extensions) multipart binding for SOAP.

Abstract messages and operations are mapped to specific transports

XML namespaces are used to ensure the uniqueness of the XML element names used in each of the three major WSDL elements. Of course, when the WSDL elements are developed separately and imported into a single complete file, the namespaces used in the separate files must not overlap. Associated schemas are used to validate both the WSDL file and the messages and operations defined within the WSDL file.

Namespaces ensure WSDL element names' uniqueness

It's safe to say that WSDL is likely to include many extensions, changes, and additions as Web services mature. Like SOAP, WSDL is designed as an extensible XML framework that can easily be adapted to multiple data type mappings, message type definitions, operations, and transports. For example, IETF (Internet Engineering Task Force) working groups are proposing a new protocol standard—Blocks Extensible Exchange Protocol (BEEP)—to define a useful connection-oriented transport. (HTTP, by contrast, is inherently connectionless, making it difficult to resolve quality-of-service problems at the transport level.) Companies interested in using Web services for internal application or integration may choose to extend WSDL to map to more traditional protocols, such as DCOM or IIOP (Internet Inter-Orb Protocol).

SOAP: ACCESSING WEB SERVICES

So far, you have defined the data (XML) and expressed the abstraction of the service necessary to support the communication and processing of the message (WSDL). You now need to define the way in which the message will be sent from one computer to another and so be available for processing at the target computer. The SOAP specification defines a messaging framework for exchanging formatted XML data across the Internet. The messaging framework is simple, easy to develop, and completely neutral with respect to operating system, programming language, or distributed computing platform. SOAP is intended to provide a minimum level of transport on top of which more complicated interactions and protocols can be built.

SOAP provides the communication mechanism to connect Web services

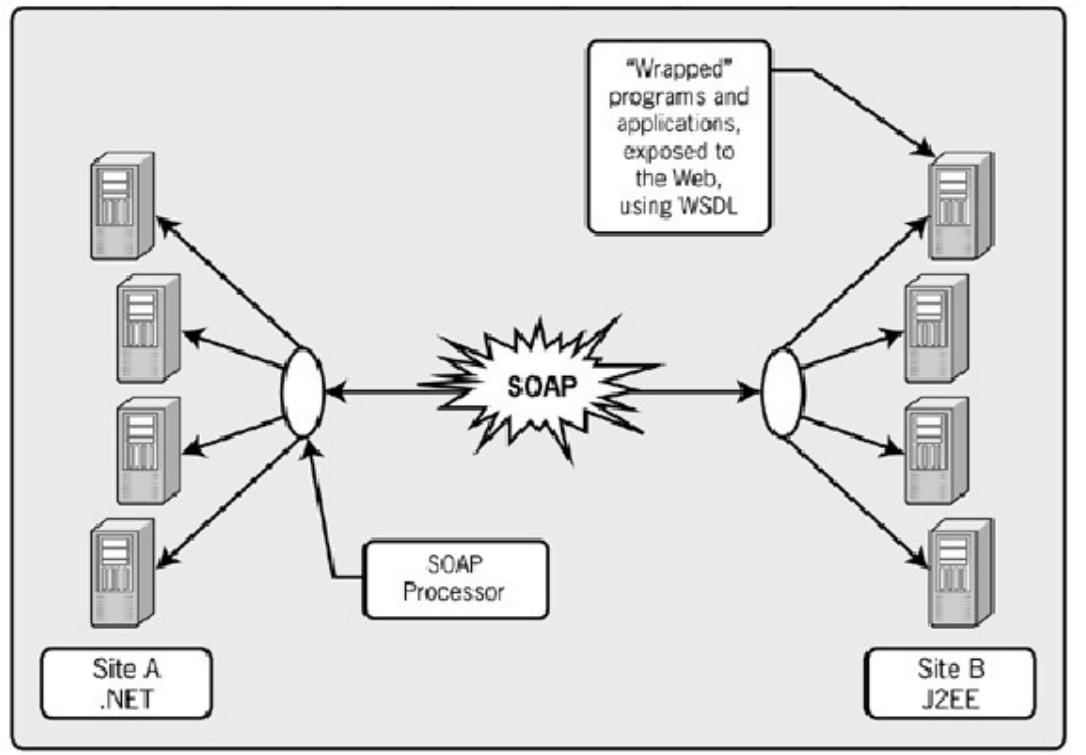
SOAP is fundamentally a one-way communication model that ensures that a coherent message is transferred from sender to receiver, potentially including intermediaries that can process part of or add to the message unit. The SOAP specification contains conventions for adapting its one-way messaging for the request/response paradigm popular in RPC-style communications and also defines how to transmit complete XML documents. SOAP defines an optional encoding rule for data types, but the end points in a SOAP communication can decide on their own encoding rules through private agreement. Communication often uses literal, or native XML, encoding.

SOAP is the XML way of defining what information gets sent and how

As shown in Figure, SOAP is designed to provide an independent, abstract communication protocol capable of bridging, or connecting, two or more businesses or two or more remote business sites. The connected systems can be built using any combination of ha rdware andsoftware that supports Internet access to existing systems such as .NET and J2EE.

The existing systems typically also represent multiple infrastructures and packaged software products. SOAP and the rest of the XML framework provide the means for any two or more business sites, marketplaces, or trading partners to agree on a common approach for exposing services to the Web.

Figure: SOAP messages connect remote sites.



SOAP has several main parts:

•**Envelope:** Defines the start and the end of the message
•**Header:** Contains any optional attributes of the message used in processing the message,either at an intermediary point or at the ultimate end point
•**Body:** Contains the XML data comprising the message being sent
•**Attachment:** Consists of one or more documents attached to the main message (SOAPwith Attachments only)
•**RPC interaction:** Defines how to model RPC-style interactions with SOAP
•**Encoding:** Defines how to represent simple and complex data being transmitted in themessage

SOAP messages contain an envelope, a header, and a body

Only the envelope and the body are required.

UDDI: PUBLISHING AND DISCOVERING WEB SERVICES

The UDDI framework defines a data model in XML and SOAP application programming interfaces (APIs) for registering and discovering business information, including the Web services a business publishes. UDDI is produced by an independent consortium of vendors, founded by Microsoft, IBM, and Ariba, to develop an Internet standard for Web service description registration and discovery. Microsoft, IBM, Hewlett-Packard, and SAP are hosting the initial deployment of a public UDDI service, which is conceptually patterned after DNS, the Internet domain name service that translates Internet host names into TCP addresses. In reality, UDDI is much more like a replicated database service accessible over the Internet.

UDDI registers and publishes Web service definitions

UDDI is similar in concept to a Yellow Pages directory. Businesses register their contact information, including such details as phone and fax numbers, postal address, and Web site. Registration includes category information for searching, such as geographical location, industry type code, business type, and so on. Other businesses can search the information registered in UDDI to find suppliers for parts, catering services, or auctions and marketplaces. A business may also discover information about specific Web services in the registry, typically finding a URL for a WSDL file that points to a supplier's Web service.

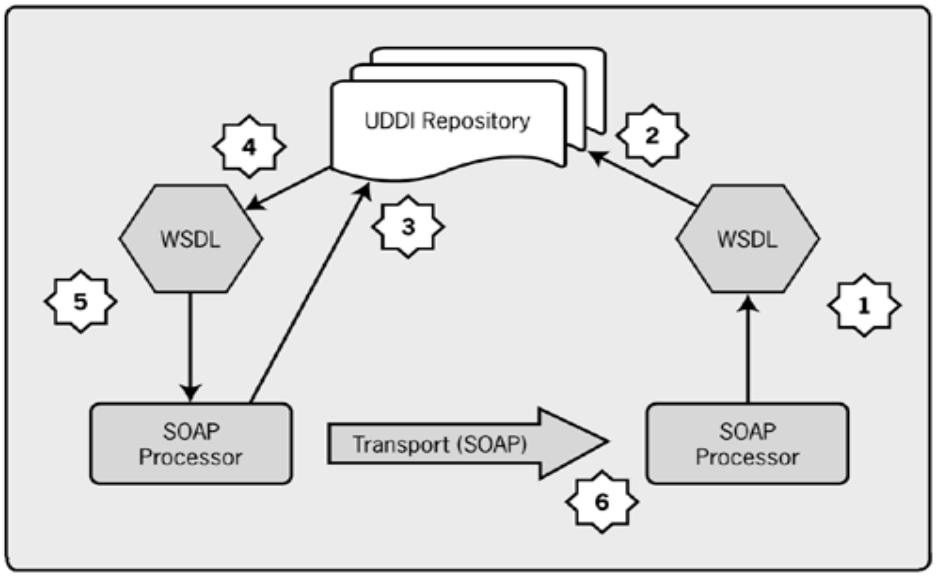
UDDI is a directory of Web services

Businesses use SOAP to register themselves or others with UDDI; then the registry clients use the query APIs to search registered information to discover a trading partner. An initial query may return several matches from which a single entry is chosen. Once a business entry is chosen, a final API call is made to obtain the specific contact information for the business.

UDDI uses SOAP for registering and discovering information

Figure shows how a business would register Web service information, along with other, more traditional contact information, with the UDDI registry. A business first generates a WSDL file to describe the Web services supported by its SOAP processor (1) and uses UDDI APIs to register the information with the repository (2). After a business submits its data to the registry, along with other contact information, the registry entry contains a URL that points to the SOAP server site's WSDL or other XML schema file describing the Web service. Once another business's SOAP processor queries the registry (3) to obtain the WSDL or other schema (4), the client can generate the appropriate message (5) to send to the specified operation over the identified protocol (6). Of course, both client and server have to be able to agree on the same protocol—in this example, SOAP over HTTP—and share the same understanding, or semantic definition of the service, which in this example is represented via WSDL. With the widespread adoption of these fundamental standards, however, this common understanding of WSDL seems ensured.

Figure: The UDDI repository can be used to discover a Web service.



USING DATABASE WEB SERVICES

The following sections describe how to use database Web services:

- Overview of Database Web Services
- Type Mapping Between SQL and XML
- Developing Database Web Services Using Oracle JDeveloper

Overview of Database Web Services

In heterogeneous and disconnected environments, there is an increasing need to access stored procedures, data and metadata, through Web service interfaces. Database Web service technology enables Web services for databases. It works in two directions:

- Database Call-in—Access database resources as a Web service
- Database Call-out—Consuming external Web services from the database itself

Database Call-in

Turning the Oracle database into a Web service provider takes advantage of your investment in Java stored procedures, PL/SQL packages, Advanced Queues, pre-defined SQL queries and DML.

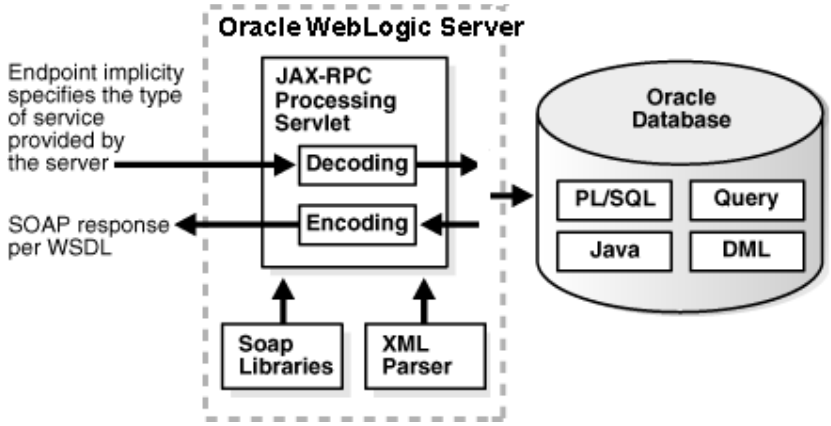
Note:
Creating Web services out of Query, Java, DML, and Advanced Queues is not supported in this release.

Client applications can query and retrieve data from Oracle databases and invoke stored procedures using standard Web service protocols. There is no dependency on Oracle specific database connectivity protocols. Applications can employ any cached WebLogic Server connection. This approach is very beneficial in heterogeneous, distributed, and non-connected environments.

Since database Web services are a part of WebLogic Web Services, they can participate in a consistent and uniform development and deployment environment. Messages exchanged between the Web services exposed database and the Web service client can take advantage of all of the management features provided by WebLogic Web Services, such as security, reliability, auditing and logging.

The following figure illustrates Web service call-in.

Figure: Web Service Calling in to the Database



Description of "Figure: Web Service Calling in to the Database"

The following steps describe the process shown in the previous figure:

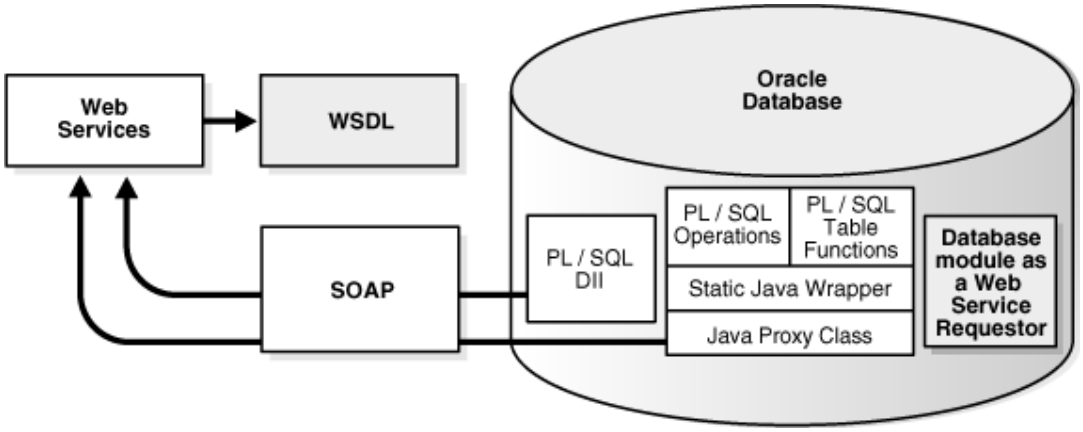
1. A request for a type of database service arrives at the application server. The service endpoint implicitly specifies the type of service requested.
2. The JAX-RPC processing servlet references the SOAP libraries and XML parser to decode the request.
3. The servlet passes the request to the classes that correspond to the exposed database operations. The generated classes can represent PL/SQL packages, queries, DML, AQ Streams, or Java classes in the database.
4. The database passes the response to the JAX-RPC processing servlet, which references the SOAP libraries and XML parser to encode it.
5. A SOAP response formed in accordance with the WSDL is returned to the client.

Database Call-out

You can extend a relational database's storage, indexing, and searching capabilities to include Web Services. By calling a Web Service, the database can track, aggregate, refresh, and query dynamic data produced on-demand, such as stock prices, currency exchange rates, or weather information. An example of using the database as a service consumer would be to call an external Web service from a predefined database job to obtain inventory information from multiple suppliers, then update your local inventory database. Another example is that of a Web Crawler: a database job can be scheduled to collate product and price information from a number of sources.

The following figure illustrates database call-out.

Figure: Calling Web Services from Within the Database



Description of "Figure: Calling Web Services from Within the Database"

The following steps describe the process shown in the previous figure:

- SQL and PL/SQL call specs—Invoke a Web service through a user-defined function call either directly within a SQL statement or view, or through a variable.
- Dynamic Web service invocation using the UTL_DBWS PL/SQL package. A Call object can be dynamically created based on a WSDL and subsequently, Web services operations can be invoked.

  Oracle Database PL/SQL Packages and Types Reference provides more information on using the UTL_DBWS PL/SQL package.

- Pure Java static proxy class—Generate a client proxy class which uses JAX-RPC. This method simplifies the Web service invocation as the location of the service is already known without needing to look up the service in the UDDI registry. The client proxy class does all of the work to construct the SOAP request, including marshalling and unmarshalling parameters.

- Pure Java using DII (dynamic invocation interface) over JAX-RPC—Dynamic invocation provides the ability to construct the SOAP request and access the service without the client proxy.

Which method to use depends on whether you want to execute from SQL or PL/SQL, from Java classes, or whether the service is known ahead of time (static invocation) or only at runtime (DII).

Type Mapping Between SQL and XML

The following sections describe the type mappings between SQL and XML for call-ins and call-outs when the Web service is known ahead of time (static invocation).

When the Web service is known at runtime you can use only the Dynamic Invocation Interface (DII) or the UTL_DBWS PL/SQL package. For more information on using the JAX-RPC DII, see the API at the following Web address: http://java.sun.com/j2ee/1.4/docs/#api.

SQL to XML Type Mappings for Web Service Call-Ins

In a database Web service call-in, a SQL operation, such as a PL/SQL stored procedure or a SQL statement, is mapped into one or more Web service operations. The parameters to the SQL operation are mapped from SQL types into XML types.

Note:
The reason there may be more than one operation is because OracleAS Web Services may be providing additional data representation choices for the SQL values in XML, such as different representations of SQL result sets.

The following table illustrates the SQL-to-XML mappings for Web service call-ins. The first column lists the SQL types. The second column of the table, XML Type (Literal), shows SQL-to-XML type mappings for the default literal value of the use attribute. The third column, XML Type (Encoded), shows the mappings for the encoded value of the use attribute. The literal and encoded values refer to the rules for encoding the body of a SOAP message.

Table:1 SQL-to-XML Type Mappings for Web Services Call-ins

SQL Type	XML Type (Literal)	XML Type (Encoded)
INT	Int	int
INTEGER	Int	int

SQL Type	XML Type (Literal)	XML Type (Encoded)
FLOAT	double	double
NUMBER	decimal	decimal
VARCHAR2	string	string
DATE	dateTime	dateTime
TIMESTAMP	dateTime	dateTime
BLOB	byte[]	byte[]
CLOB	String	String
LONG	String	String
RAW	byte[]	byte[]
Primitive PL/SQL indexby table	Array	Array
PL/SQL Boolean	boolean	boolean
PL/SQL indexby table	complexType	complexType
PL/SQL record	complexType	complexType
REF CURSOR (*name*Beans)	Array	Array
REF CURSOR *name*XML)	Any	test_xml
REF CURSOR nameMLRowSet	swaRef	test_xml
SQL object	complexType	complexType
SQL table	complexType	complexType
SYS.XMLTYPE	any	test_xml

Note:
If National Language Support (also known as "NLS" or "Globalization Support") characters are used in a SQL SYS.XMLTYPE value, they may not be properly handled.

A query or a PL/SQL function returning REF CURSOR will be mapped into the three methods listed below, where *name* is the name of the query or the PL/SQL function.

- *name*Beans—This method returns an array, where each element is an instance of an XSD complex type that represents one row in the cursor. A complex type sub-element corresponds to a column in that row.
- *name*XMLRowSet—This method returns a swaRef or text_xml response that contains an OracleWebRowSet instance in XML format.
- *name*XML—this method returns an XML any or text_xml response that contains an Oracle XDB row set.

Both OUT and IN OUT PL/SQL parameters are mapped to IN OUT parameters in the WSDL file.

Note that Table -1 provides two different mappings: one for literal and another for encoded use. The default mapping is literal. From a database Web service's perspective, there is no special reason why encoded should be used. The mapping for encoded is provided in case you encounter scenarios which call for the encoded use setting. All of the descriptions in this chapter assume that you will be using the literal use setting unless otherwise specified.

XML-to-SQL Type Mapping for Web Service Call-outs

In database Web services call-outs, XML types are mapped into SQL types. The following table lists the XML-to-SQL type mappings used in call-outs.

Table:2 XML-to-SQL Type Mappings for Web Service Call-outs

XML Type	**SQL Type**
Int	NUMBER
Float	NUMBER
Double	NUMBER
decimal	NUMBER
dateTime	DATE
String	VARCHAR2
byte[]	RAW
complexType	SQL OBJECT
Array	SQL TABLE
test_xml	XML Type

ACCESSING A WEB SERVICE THROUGH ASP.NET

Introduction

Web services signal a new age of trivial distributed application development. While Web services are not intended nor do they have the power to solve every distributed application problem, they are an easy way to create and consume services over the Internet. One of the design goals for Web Services is to allow companies and developers to share services with other companies in a simple way over the Internet. Web services take Web applications to the next level.

Using Web services, your application can publish its function or message to the rest of the world.

Web services use XML to code and decode your data and SOAP to transport it using open protocols.

With Web services, your accounting departments Win 2K servers' billing system can connect with your IT suppliers UNIX server.

Using Web services, you can exchange data between different applications and different platforms.

With Microsoft .NET platform, it is a simple task to create and consume Web Services. In this article, I am going to show how to call a published Web service inside a Web project. I use a test published Web service; Extentrix Web Services 2.0 Application Edition that Extentrix published for the developer community to help them in testing and developing.

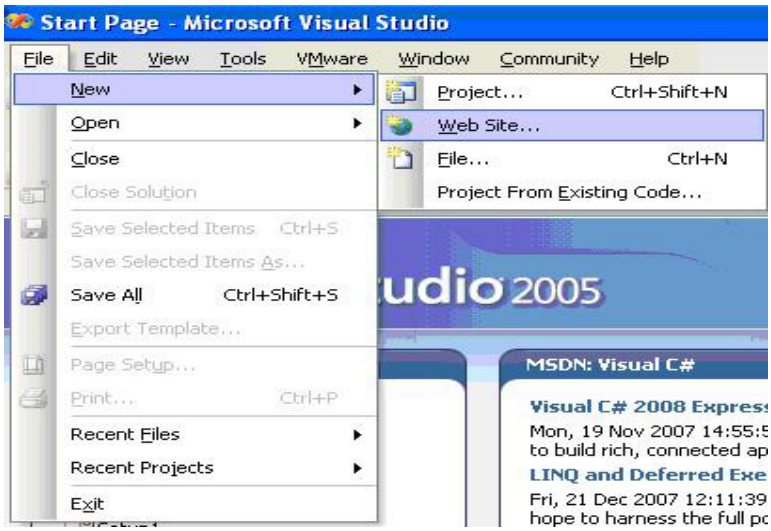
In general Extentrix Web Services for Citrix Presentation Server helps you get information about a published application for a specific client with the specified details, server types, and client types. It also returns the ICAfile description to be used to launch an application with a given parameter and checks the user's credentials and returns true if they are valid.

Simple Steps to Consume a Web Service

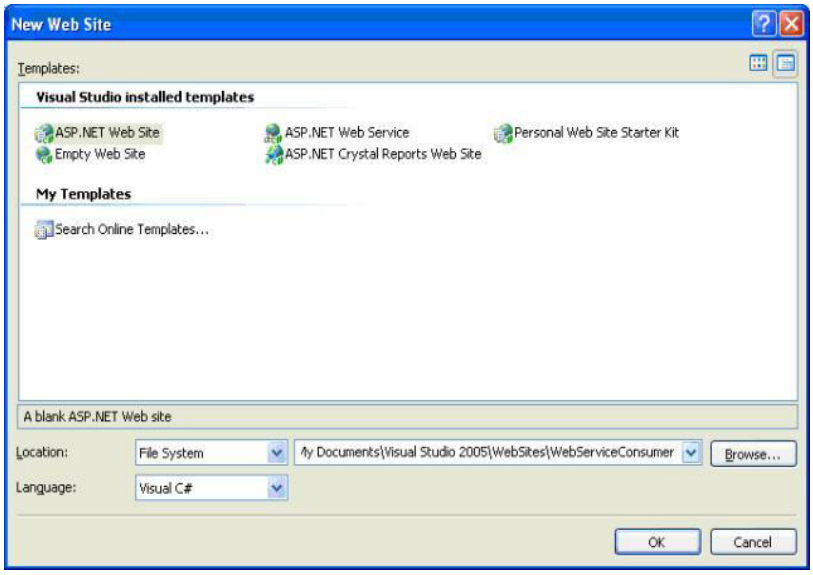
1. Create a Web Site project
2. Add a Web Reference
3. Call the Web services APIs inside the code

First Step: Create a Web Site Project

1. To create a new Web Site project, choose New from File menu, then choose Web Site as shown below:



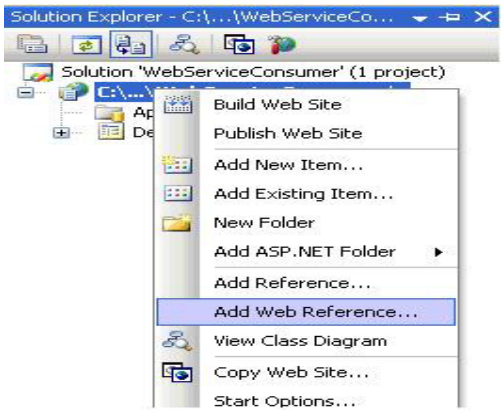
2. Choose ASP.NET Web Site. Name the project and click OK:



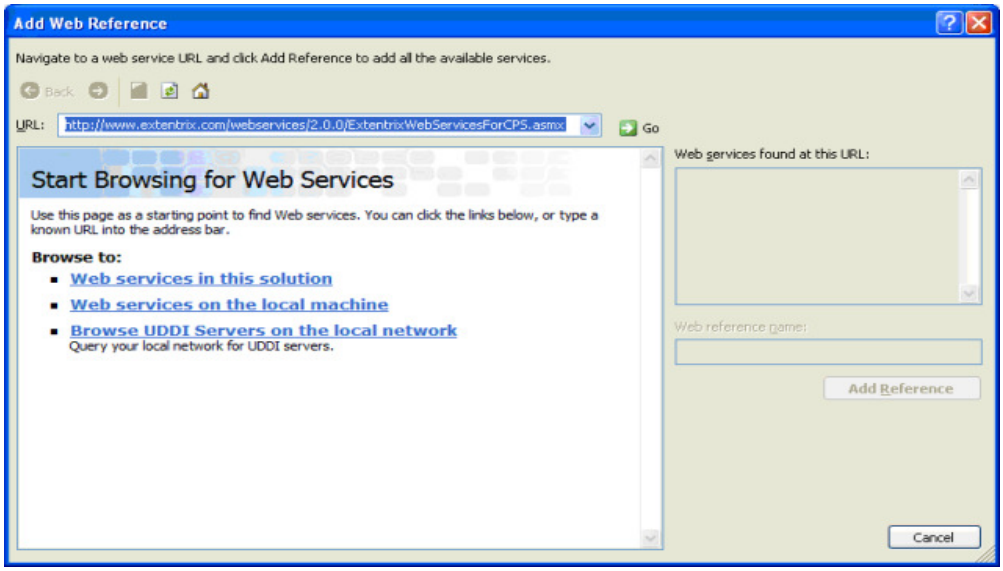
Second Step: Add a Web Reference

After creating the Web Site project, it's time to add a Web reference for our Web service.

1. In the solution explorer, right click the project node, choose Add Web Reference:



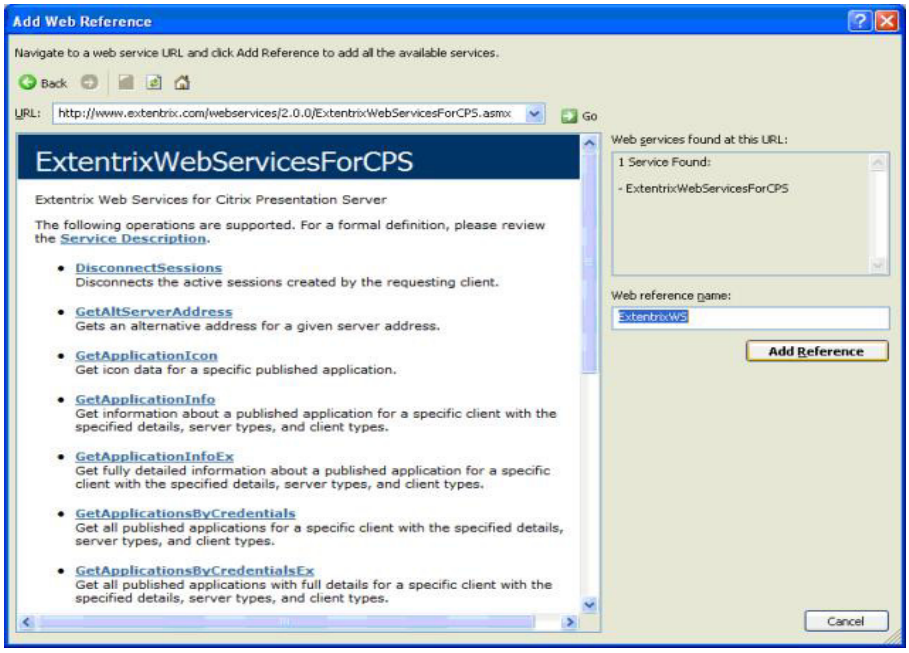
2. A new window with Add Web Reference title will be opened:



In the URL field, insert the URL for the Web service. In this tutorial, as I mentioned before, I'll use the test published Web services from Extentrix: "Extentrix Web Services 2.0 – Application Edition".

After clicking the Go button, you will see the Web services APIs.

3. Set a name for your Web service reference in the Web reference name field and click Add Reference:



Third Step: Call the Web Services APIs Inside the Code

After successfully adding to the Web service, now we are ready to call the Web services APIs inside our project.

1. First we need to add the added Web reference to our class. ExtentrixWS is the name of the added Web service from the previous step.

   using ExtentrixWS;

2. Create a proxy object for our added Web service reference, where ExtentrixWebServicesForCPS is the name of the Web Services.

   //define a Web service proxy object.
   private ExtentrixWS.ExtentrixWebServicesForCPS proxy;

3. As I explained before, we need credentials to pass to the Citrix Presentation Server. We will pass these credentials through the Web services APIs:

   //define a Citrix Presentation Server Credentials object
   private Credentials credentials;

   Initialize the proxy and the credentials objects:

   //initialize objects
   proxy = new ExtentrixWebServicesForCPS();
   credentials = new Credentials();

4. Set the values for Citrix credentials. I set the credentials values for the test of Extentrix Web Service:

```
//set credentials
//these values are according to Citrix testdrive presentation server
//for which Extentrix published a web service for developers to use it
//as a test web service.
    credentials.Password = "demo";
    credentials.UserName = "citrixdesktop";
    credentials.Domain = "testdrive";

//because it is a sample, we will use no encryption method.
//so the password will be sent as a clear text.
    credentials.PasswordEncryptionMethod = 0;

//set the domain type to windows domain
    credentials.DomainType = 0;
```

   Now we can call any Web services available. It is as simple as calling any ordinary function.

5. Call the GetApplicationsByCredentialsEx Web service. This web service takes the following parameters:
   - Credentials: Citrix Credential to access Citrix Presentation Server Farm
   - Client Name: Pass your machine name
   - Client IP: Pass your machine IP
   - Desired Details : Details you asked for
   - Server Types: Pass “all”
   - Client Types: Pass “all”

This API returns an array of ApplicationItemEx. This class will be built for you once you add the Web reference.

This class contains the published application properties. This Web service is used to get all the published applications, and then create an ImageButton for each application.

```
// 1) Get all the published applications list by calling GetApplicationsByCredentialsEx
//    web service.
// 2) create an ImageButton for each application
// 3) Create Image for the application
// 4) Add it to the AppList panel.
// 5) Set the event handler for each ImageButton, so when clicking it the associated
//    application will run calling the web service
ApplicationItemEx[] items = proxy.GetApplicationsByCredentialsEx
   (credentials, Request.UserHostName,
Request.UserHostAddress, new  string[] { "icon","icon-info"}, new string[]{ "all" },
new string[] { "all"});
```

```
//loop for each published application
for (int i = 0; i < items.Length; i++) {
//create the ImageButton
System.Web.UI.WebControls.ImageButton app = new
System.Web.UI.WebControls.ImageButton();

//set the Image URL to the created image
app.ImageUrl = createIcon(items[i].InternalName,items[i].Icon);

//set the ToolTip to the name of the published application
app.ToolTip = items[i].InternalName;

//add the ImageButton to the AppList panel
AppList.Controls.Add(app);

//set the event handler for the ImageButton.
app.Click += new
System.Web.UI.ImageClickEventHandler(this.OnApplicationClicked);
}
```

Finally, another example in calling a Web service is to launch the published application. In this example, in the event handler of the applications ImageButtons we have to launch the clicked application.

Get the ICA file content by calling LaunchApplication Web service. Then write the ICA file content to the response to launch the application.

```
private
void OnApplicationClicked (object sender, System.EventArgs e)
{
   ServicePointManager.Expect100Continue = false;

   // Get the event source object.
   System.Web.UI.WebControls.ImageButton app =
      (System.Web.UI.WebControls.ImageButton)sender;

   //Get the file ICAfile content by calling LaunchApplication web service.
   string = proxy.LaunchApplication(app.ToolTip, credentials, Request.UserHostName,
         Request.UserHostAddress);

   //Set the response content type to "application/x-ica" to run the file.
   Response.ContentType = "application/x-ica";

   //Run the application by writing the file content to the response.
   Response.BinaryWrite(Response.ContentEncoding.GetBytes(ica));
   Response.End();
}
```