

Tutorial: Using Web Services

This tutorial describes how the Financial Brokerage Service (FBS) sample application uses *Web Services* to access functionality made available by other applications.

This tutorial assumes that you are familiar with the FBS and have installed and configured the required software as described in *About the Financial Brokerage Service*.

Contents

1. Concepts
2. Design
3. Required Software
4. Setup
5. Implementation
6. Resources
7. Feedback



Concepts

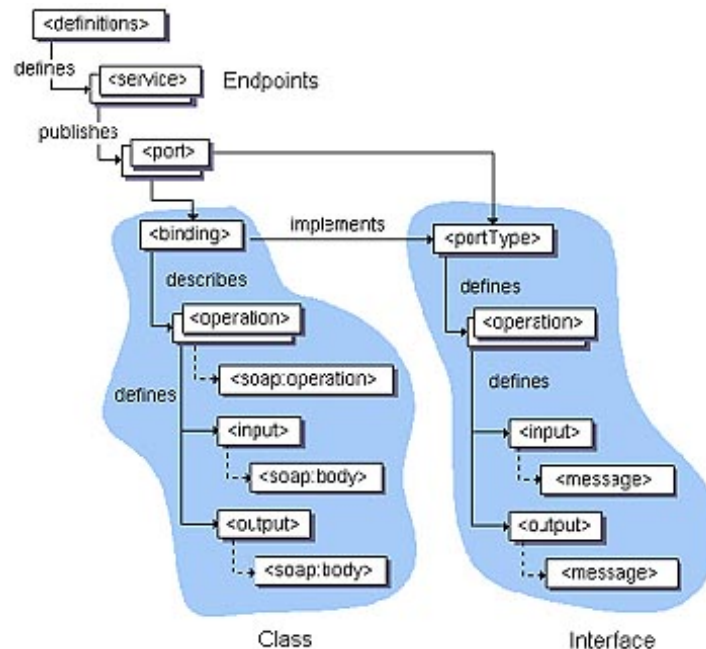
A Web Service (commonly called a *service*) comprises some content, or some process, or both, with an open programmatic interface. Simple examples: currency converter, stock quotes, dictionary. More complex examples: travel planner, procurement workflow system. A service has the following characteristics:

- Internet-based application that performs a specific task and complies with a standard specification.
- An executable that can be expressed and accessed using XML and XML messaging.
- Can be published, discovered, and invoked dynamically in a distributed computing environment.
- Is platform- and language-independent.

Two key parts of the Web Services infrastructure are WSDL, a language for describing Web Services; and SOAP, a protocol that enables applications an Web Services to communicate.

About WSDL

The Web Services Description Language (WSDL) is an XML language for describing the syntax of Web Service interfaces and their locations. The WSDL specification calls it "an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information." Within that context, the diagram below illustrates the elements that are present in a WSDL document, and shows how they are related.



Programmers or automated development tools can create WSDL files to describe a service and can make the description available over the Internet. Client-side programmers and development tools can use published WSDL

descriptions to obtain information about available Web Services and to build and create proxies or program templates that access available services.

A WSDL document has a `definitions` element that contains the `types`, `message`, `portType`, `binding`, and `service` elements as described in the following table.

definitions	Defines one or more services. A <code>definition</code> element supports the following attributes: <ul style="list-style-type: none">• <code>name</code> is optional.• <code>targetNamespace</code> is the logical namespace for information about this service. WSDL documents can import other WSDL documents, and setting <code>targetNamespace</code> to a unique value ensures that the namespaces do not clash.• <code>xmlns</code> is the default namespace of the WSDL document, and it is set to <code>http://schemas.xmlsoap.org/wsdl/</code>. All the WSDL elements, such as <code><definitions></code>, <code><types></code> and <code><message></code> reside in this namespace.• <code>xmlns:xsd</code> and <code>xmlns:soap</code> are standard namespace definitions that are used for specifying SOAP-specific information as well as data types.• <code>xmlns:tns</code> stands for <i>this namespace</i>.
types	Provides information about any complex data types used in the WSDL document. When simple types are used, the WSDL document does not need this section.
message	An abstract definition of the data being communicated.
operation	An abstract description of the action supported by the service.
portType	An abstract set of operations supported by one or more endpoints.
binding	Describes how the operation is invoked by specifying concrete protocol and data format specifications for the operations and messages.
port	Specifies a single endpoint as an address for the binding, thus defining a single communication endpoint.
service	Specifies the port address(es) of the binding. The service is a collection of network endpoints or ports.

The following listing shows the WSDL code for the Exchange Web Service used by the FBS.

```

<?xml version = '1.0' encoding = 'windows-1252'?>
<!--Generated by the Oracle9i JDeveloper Web Services WSDL Generator-->
<!--Date Created: Tue Apr 23 15:09:13 IST 2002-->
<definitions
  name="Exchange"
  targetNamespace=

"http://tempuri.org/jdeveloper/generated/oracle/otnsamples/ibfbs/admin/ejb/Exchange"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns=

"http://tempuri.org/jdeveloper/generated/oracle/otnsamples/ibfbs/admin/ejb/Exchange"
  xmlns:ns1=

"http://tempuri.org/jdeveloper/generated/oracle/otnsamples/ibfbs/admin/ejb/Exchange/schema">
  <types>
    <schema
      targetNamespace=

"http://tempuri.org/jdeveloper/generated/oracle/otnsamples/ibfbs/admin/ejb/Exchange/schema"
      xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" />
    </types>
    <message name="processOrder0Request">
      <part name="toAddress" type="xsd:string"/>
      <part name="symbol" type="xsd:string"/>
      <part name="tradeDate" type="xsd:dateTime"/>
      <part name="tradeType" type="xsd:string"/>
      <part name="qty" type="xsd:int"/>
    </message>
    <message name="processOrder0Response">
      <part name="return" type="xsd:int"/>
    </message>
    <portType name="ExchangePortType">
      <operation name="processOrder">
        <input name="processOrder0Request" message="tns:processOrder0Request" />
        <output name="processOrder0Response" message="tns:processOrder0Response" />
      </operation>
    </portType>
    <binding name="ExchangeBinding" type="tns:ExchangePortType">
      <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
      <operation name="processOrder">
        <soap:operation soapAction="" style="rpc" />
        <input name="processOrder0Request">
          <soap:body use="encoded"
            namespace="oracle.otnsamples.ibfbs.admin.ejb.Exchange"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
        </input>
        <output name="processOrder0Response">
          <soap:body use="encoded"
            namespace="oracle.otnsamples.ibfbs.admin.ejb.Exchange"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
        </output>
      </operation>
    </binding>
    <service name="Exchange">

```

```
<port name="ExchangePort" binding="tns:ExchangeBinding">
  <soap:address
location="http://incq212e.idc.oracle.com:8888/ibfbs/ExchangeService"/>
  </port>
</service>
</definitions>
```

About SOAP

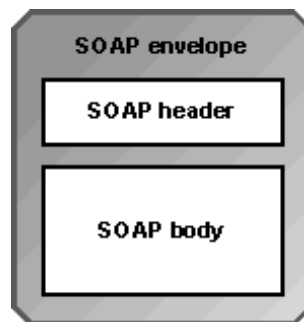
The Simple Object Access Protocol (SOAP) is a lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. SOAP supports different styles of information exchange, including:

- Remote Procedure Call style (RPC), which allows for request-response processing, where an endpoint receives a procedure oriented message and replies with a correlated response message.
- Message-oriented information exchange, which supports organizations and applications that need to exchange business or other types of documents where a message is sent but the sender may not expect or wait for an immediate response.

SOAP has the following features:

- Protocol independence
- Language independence
- Platform and operating system independence
- Support for SOAP XML messages incorporating attachments (using the multipart MIME structure)

A SOAP message consists of a SOAP envelope that encloses two data structures, the SOAP header and the SOAP body, and information about the name spaces used to define them. The header is optional; when present, it conveys information about the request defined in the SOAP body. For example, it might contain transactional, security, contextual, or user profile information. The body contains a Web Service request or reply to a request in XML format. The high-level structure of a SOAP message is shown in the following figure.



Here is a SOAP message that corresponds to the WSDL listed above:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:processOrder
xmlns:m="http://tempuri.org/jdeveloper/generated/oracle/otnsamples/ibfbs/admin/ejb/Exchange">
      <m:toAddress>abcd@yahoo.com</m:toAddress>
      <m:symbol>ORCL</m:symbol>
      <m:tradeDate>2001-12-15T13:20:00-05:00</m:tradeDate>
      <m:tradeType>B</m:tradeType>
      <m:qty>50</m:qty>
    </m:processOrder>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP messages, when used to carry Web Service requests and responses, can conform to the WSDL definition of available Web Services. WSDL can define the SOAP message used to access the Web Services, the protocols over which such SOAP messages can be exchanged, and the Internet locations where these Web Services can be accessed. The WSDL descriptors can reside in UDDI or other directory services, and they can also be provided via configuration or other means such as in the body of SOAP request replies.

The SOAP specification provides a standard way to encode requests and responses. It describes the structure and data types of message payloads using XML Schema. The way that SOAP is used for the message and response of a Web Service is:

- The SOAP client uses an XML document that conforms to the SOAP specification and which contains a request for the service.
- The SOAP client sends the document to a SOAP server, and the SOAP servlet running on the server handles the document using, for example, HTTP or HTTPS.
- The Web service receives the SOAP message, and dispatches the message as a service invocation to the application providing the requested service.
- A response from the service is returned to the SOAP server, again using the SOAP protocol, and this message is returned to the originating SOAP client.

SOAP provides a way to leverage the industry investment in XML. Also, since SOAP is typically defined over "firewall friendly" protocols such as HTTP and SMTP, the industry investment in firewall technology is leveraged as well. Thus, by defining SOAP as an essential part of Web Services, the industry will likely enjoy volume production use of Web Services far sooner than if other strategies had been employed.





Design

OTN developers used Web Services to access data provided by other, completely separate applications. This approach makes the FBS itself smaller and therefore easier to maintain. It also makes the FBS more flexible, because it's relatively easy to swap one Web Service for another.

The FBS uses Web Services to

- Fetch stock quotes.
- Fetch financial news.
- Communicate with other FBS instances

To watch a viewlet that shows how the FBS uses Web Services, browse to:

- http://otn.oracle.com/sample_code/tutorials/fbs/files/viewlets/ibfbswebservices_viewlet.html

Fetching Stock Quotes

Rather than build a module to fetch stock quotes, OTN developers used an existing Web Service. The Web Service WSDL URL is

`http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl`.

Oracle9i JDeveloper includes a wizard that takes a WSDL URL and generates the Java stub code required to access the Web Service.

Fetching Financial News

Similarly, the FBS uses another Web Service to fetch financial news. However, instead of using a WSDL URL, the FBS accesses the Web Service via APIs in a library provided by Google. This library is required to build the FBS.

Communicating with Other FBS Instances

Each instance of the FBS represents a unique stock exchange. When an FBS user buys or sells

a particular stock, the order is sent to the corresponding stock exchange as a simplified message. If the stock is not traded in the current exchange, the FBS forwards the order to another FBS, and so on until the trade is executed. FBS instances pass orders around via Web Services.





Required Software



This tutorial presents several code examples. If you want to study them in context, download and install the FBS source code. If you also want to build and run the FBS, you will need the software listed in the Required Software section of *About the Financial Brokerage Service*.





Setup



If you use Oracle9i JDeveloper as your IDE, add the `J2EE` library to your project. For coding details, see the Implementation section.

To configure your system to build and run the FBS sample application, see the Setup section of *About the Financial Brokerage Service*.





Implementation

Rather than build a module to fetch stock quotes, OTN developers used an existing Web Service. The Web Service WSDL URL is `http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl`, and that's just about all the developers need to access the service. Here's why: Oracle9i JDeveloper provides a wizard that can parse the WSDL file and generate the Java code that a client needs to invoke the service's methods. In this case, the generated Java class is `StockQuoteServiceStub`. It provides code for connecting to the service and wrapper methods that a client can call to interact with the service. Following is some of the code from the generated stub class that connects to the service and wraps the `getQuote` method.

```
/**
 * Generated by the Oracle9i JDeveloper Web Services Stub/Skeleton Generator.
 * Date Created: Fri Apr 19 15:45:47 IST 2002
 * WSDL URL: http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl
 *
 * net.xmethods.services.stockquote.StockQuote web service
 */
public class StockQuoteServiceStub {
    public String endpoint = "http://66.28.98.121:9090/soap";
    private OracleSOAPHTTPConnection m_httpConnection = null;
    public StockQuoteServiceStub() {
        m_httpConnection = new OracleSOAPHTTPConnection();
    }
    public Float getQuote(String symbol)
        throws Exception {
        Float returnVal = null;
        URL endpointURL = new URL(endpoint);
        Call call = new Call();
        call.setSOAPTransport(m_httpConnection);
        call.setTargetObjectURI("urn:xmethods-delayed-quotes");
        call.setMethodName("getQuote");
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
        Vector params = new Vector();
        params.addElement(new Parameter("symbol", String.class, symbol, null));
        call.setParams(params);
        Response response = call.invoke(endpointURL,
            "urn:xmethods-delayed-quotes#getQuote");
        if(!response.generatedFault()) {
            Parameter result = response.getReturnValue();

```

```

    returnVal = (Float)result.getValue();
}
else {
    Fault fault = response.getFault();
    throw new SOAPException(fault.getFaultCode(), fault.getFaultString());
}
return returnVal;
}

```

Because the FBS uses several Web Services, the developers chose to centralize most of the service-related client code in one class, `WebServicesHelper`. Following is some of the code from `WebServicesHelper` that calls the `getQuote` method provided by the remote Web Service.

```

public class WebServicesHelper {
...

    public WebServicesHelper() {
...

        // Initialize stock quote service stub
        stockService = new StockQuoteServiceStub();
    }

    public StockRate fetchStockRate(String symbol)
        throws WebServiceAccessException {
        float rate;
        try {
            rate = stockService.getQuote(symbol).floatValue();
        } catch (Exception ex) {
            throw new WebServiceAccessException(" Stock Quote service error : "
                + ex.toString());
        }
        return new StockRate(symbol, new Date(), rate, (float) (rate + 0.5));
    }
}

```

To demonstrate a different approach, the FBS accesses another third-party Web Service via an API provided by that third party. Specifically, the FBS uses Google Search APIs to fetch financial news. Here is the code from `WebServicesHelper.java` that makes the connection and fetches the data.

```

package oracle.otnsamples.ibfbs.admin.helper;
...
import com.google.soap.search.GoogleSearch;
import com.google.soap.search.GoogleSearchFault;

```

```

import com.google.soap.search.GoogleSearchResult;
import com.google.soap.search.GoogleSearchResultElement;
...
public class WebServicesHelper {
    private GoogleSearch search = null;
    ...
    public WebServicesHelper() {
        // Initialize Google Search and set the key
        search = new GoogleSearch();
        search.setKey(ConnectionParams.googleSearchKey);
        // Initialize stock quote service stub
        stockService = new StockQuoteServiceStub();
    }

    /**
     * This method retrieves the latest news for a particular symbol by accessing
     * the news webservice.
     *
     * @param symbol Stock symbol
     * @param maxResults maximum number of results for the symbol
     * @return news news for given symbol
     * @exception WebServiceAccessException if accessing webservice fails
     * @since 1.0
     */
    public Collection fetchLatestNews(String symbol, String companyName,
                                     int maxResults)
        throws WebServiceAccessException {
        ArrayList newsList = new ArrayList();
        News      news      = null;
        Date      today     = new Date();
        int julianDate = this.toJulian(today);
        // Set the search string
        search.setQueryString(new StringBuffer().append("intitle:")
                                                .append(companyName)
                                                .append(" finance stock news")
                                                .append(" daterange:")
                                                .append(julianDate-30)
                                                .append("-")
                                                .append(julianDate).toString());
        // Set the maximum number of results returned
        search.setMaxResults(maxResults);
        try {
            // Search
            GoogleSearchResult      searchResult = search.doSearch();
            GoogleSearchResultElement[] result   =

```

```
    searchResult.getResultElements();
// Number of matching results
int noofresults = searchResult.getEndIndex();
for (int i = 0; i < noofresults; i++) {
    news = new News(null, today, symbol, removeHTML(result[i].getTitle()),
        URLEncoder.encode(result[i].getURL()));
    newsList.add(news);
}
} catch (GoogleSearchFault ex) {
    throw new WebServiceAccessException(" Stock News service error : "
        + ex.toString());
}
return newsList;
}
```

Each instance of the FBS represents a unique stock exchange. When an FBS user buys or sells a particular stock, the order is sent to the corresponding stock exchange as a simplified message. If the stock is not traded in the current exchange, the FBS forwards the order to another FBS, and so on until the trade is executed. FBS instances pass orders around via Web Services. See `ExchangeStub.java`, `Exchange.java`, `ExchangeHome.java`, `ExchangeBean.java`, and `Exchange.wsdl` for implementation details.





Resources

This tutorial is part of a series, *J2EE and EJB 2.0 Techniques*, based on the Financial Brokerage Service (FBS) sample application. Following are links to resources that can help you understand and apply the concepts and techniques presented in this tutorial.

Resource	URL
Viewlet: Web Services in the FBS	http://otn.oracle.com/sample_code/tutorials/fbs/files/viewlets/ibfbswebservices_viewlet.html
OTN Web Services Center	http://otn.oracle.com/tech/webservices/content.html
Oracle9i Web Services Tutorials	http://otn.oracle.com/tech/webservices/htdocs/series/content.html





Feedback

If you have questions or comments about this tutorial, you can:

- Post a message in the OTN Sample Code discussion forum. OTN developers and other experts monitor the forum.
- Send email to the author. <mailto:Robert.Hall@oracle.com>

If you have suggestions or ideas for future tutorials, please send email to:

- <mailto:Raghavan.Sarathy@oracle.com>

