Level: Advanced
Works with: QuickPlace
Updated: 02-Jun-2003

by
Raj
Balasubramanian

A *Web Service* is a set of APIs that can be located and run over the Web. Web Services provide a common interface to access complex application functions. This offers the promise of integrating disparate, distributed systems within and across an enterprise. Web Service technology is supported by several major software vendors, including IBM and Microsoft, and has garnered increasing interest within the Web development community. For more information about Web Services, see the **developerWorks Web Services zone**.

Existing applications can be made to run as Web Services, including Lotus Web applications like QuickPlace. QuickPlace 3.0 introduces two important new features that are Web Services-accessible:

- My Places, a custom list of Places to which a specific user belongs
- Place Catalog, which stores the Places and the membership information

Together these two features provide a single point of entry into a QuickPlace environment that potentially contains multiple Place servers.

Using the techniques described in this article, you can expose the My Places functionality as a Web Service. This allows Web clients to retrieve the Places of a given user from a variety of programming languages, including Python, Java, COM (compliant languages), Perl, and others. This article starts with a quick overview of how Web Services work and how to use the Web Services Description Language (WSDL), the XML format for defining Web Services. We then describe how to create the XML code and Domino agent required to make My Places functionality available as a Web Service. (You can download all the code shown in this article from the **Sandbox**.) This article assumes that you're an experienced Domino and XML programmer.

A quick overview of Web Services

Traditionally, the most common methods for remote communication were based on Remote Procedure Call (RPC) technology. Well-known examples include NRPC (Notes RPC), IIOP (Internet Inter-Orb Protocol), RMI (Remote Method Invocation), and DCOM (Distributed Component Object Model) to name a few. All these employ a specific format and language to describe the data that's being transported. They also use specific security paradigms to establish and validate identity. Libraries specific to each of these protocols need to be deployed on the client in order to support them. This can become cumbersome in complex, multi-application environments.

Web Services offer a way to converge the disparate RPC models into a common language that various systems can understand and support while leveraging existing Web technologies. This allows a service coded in J2EE to be accessed and invoked by a Visual Basic client on a Windows machine or by a PHP page running on a Linux

machine. To make this possible, Web Services technology includes encapsulation, message passing, dynamic binding, service description, and querying.

The major components of Web Services technology are:

- *Storage and retrieval*
  The standard for this is Universal Description, Discovery, and Integration (UDDI). This article does not concern itself with Web Services storage and retrieval; for more information, see the **UDDI Web site**.
- *Description*
  The standard is Web Services Description Language (WSDL).
- *Implementation/communication*
  The standard is Simple Object Access Protocol (SOAP).

The common language used by all three of these components is XML. The next sections further describe WSDL and SOAP.

WSDL

WSDL is an XML format for describing Web Services as collections of communication endpoints capable of exchanging messages. WSDL definitions provide information for distributed systems and serve to automate the details involved in inter-application communications.

WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. A WSDL document uses the following elements to define a Web Services:

- Types, a container for data type definitions using a type system such as XML Schema Definition (XSD)
- Message, an abstract, type definition of the data being communicated (both request and response)
- Operation, an abstract description of an action supported by the service
- Port Type, an abstract set of operations supported by one or more endpoints
- Binding, a concrete protocol and data format specification for a particular port type
- Port, a single endpoint defined as a combination of a binding and a network address
- Service, a collection of related endpoints

For detailed information on WSDL, see the **Web Services Description Language specification**.

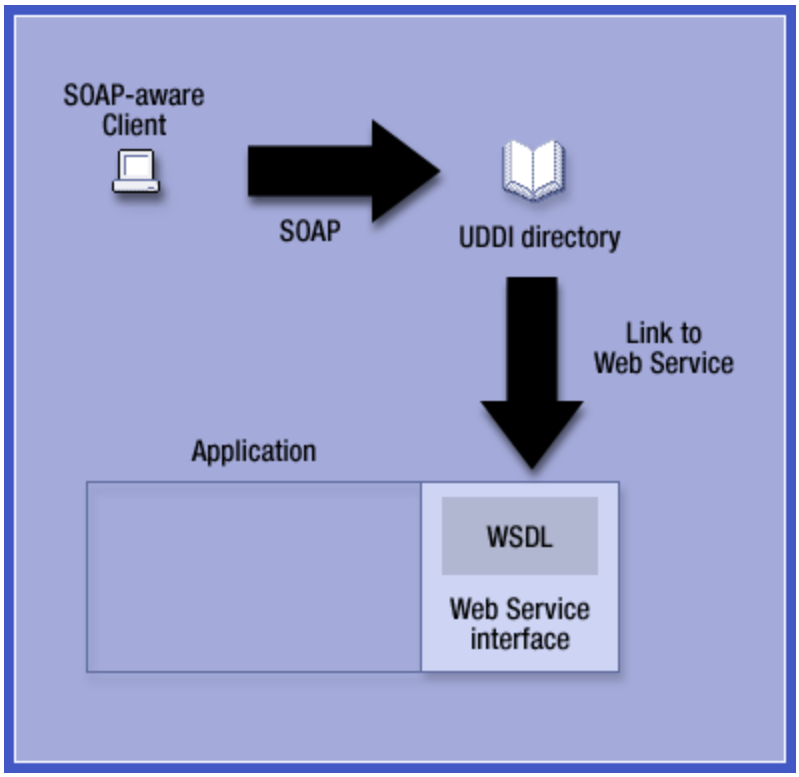
SOAP

Simple Object Access Protocol (SOAP) is a protocol for exchanging information in decentralized, distributed environments. The SOAP protocol consists of an envelope for describing the content of a message and how to process it, encoding rules for expressing instances of application-defined data types, and a standard for representing remote procedure calls and responses. Because SOAP can use HTTP as the transport mechanism (in addition to SMTP, Jabber, and BEEP), it can work seamlessly across existing enterprise firewalls. For more information, see the **SOAP specification**.

Putting it together

The first step in enabling an application as a Web Service is to provide an implementation of the application that can be accessed by a SOAP-aware client. This implementation of the application can then be considered a Web Service its simplest form. The client sends a SOAP request message, and the Web Service processes the request and sends back a SOAP response. For this to succeed, the client must know the Web Service implementation address as well as the parameters that can be passed within the message. To ensure that the proper implementation is invoked and that appropriate parameters are used, the client can request a WSDL for the Web Service (if it's available). WSDL describes the request parameters, response parameters, and the address for the Web Service implementation. Using this information, the client can then send the SOAP request to the correct address with appropriate parameters.

Taking this one step further, we can publish this newly created Web Service and client's query for this service in a UDDI directory. This is similar to so-called "yellow pages" of Web Services. The published services can be categorized and associated with a company and can have more metadata describing the service. The client can then search for a desired Web Service and upon finding one, use the link for the WSDL to use the Web Service:



Creating the My Places Web Service

In this section, we develop a Web Service to deliver My Places (a list of all QuickPlaces to which a given user belongs) and to produce a WSDL for the Web Service. This WSDL then needs to be published in a UDDI directory to make it available to SOAP-aware Web clients. (Publishing the WSDL in a UDDI directory is not covered in this article; for information, see the **UDDI home page**.)

You can create the My Places Web Service by using a number of technologies. These include:

- *Java*
  Create the Web Service and host it on an Apache or WebSphere server. Then code a servlet to handle SOAP messages and host it on the QuickPlace server.
- *.Net*
  To do this, develop the Web Service and host it on .Net.
- *LotusScript agent*
  There are three ways to implement the Web Service using LotusScript: Create an agent using the Microsoft SOAP Toolkit (COM-based), build a parser to handle SOAP messages, or develop a SOAP LSX and use it within the agent.

Because the first method is probably the quickest and easiest method, we will use it in this article. Note that this requires the Microsoft SOAP Toolkit 3.0, which you can **download here**.

The LotusScript agent handles the client SOAP request as follows:

1. The client sends an HTTP SOAP request that includes the user name associated with the requested My Places. The target of the request is the agent's URL.
2. The agent initializes the MS SoapReader30 object. It then loads the SOAP message from the request and extracts the user name.
3. LotusScript database function calls open the Place Catalog database and query for the user name. If the user name exists, a list of documents (My Places) is returned. The Place Name and the Place Server Name are extracted from each document.
4. Print statements build a SOAP message from the returned query results. This message is then sent to the client as an HTTP SOAP response.

5. The client receiving this SOAP response parses and displays the data in the message.

The myplaceservice agent
The My Places Web Service is deployed as a Domino agent in a Domino database that resides on the Place Catalog servers. In our example, we name the agent myplaceservice and the Domino database MyPlaces.nsf. The agent is a Run Once agent.

The agent code is split into the four main sections. The first consists of basic variable/object declarations:

```
Sub Initialize
    Dim s As New notessession
    Dim db As notesdatabase
    Dim curdb As notesdatabase
    Dim pdoc As notesdocument
    Dim coll As notesdocumentcollection
    Dim v As notesview
    Dim doc As notesdocument
    Dim soap As Variant
    Dim SOAPin As String
```

In addition to the standard Notes object declarations, we also define holders for the SOAPReader Object (in the soap variant) and a string variable to hold contents being read.

The second section of code uses the MSSoapReader Object to read the incoming SOAP request:

```
    Set soap = CreateObject("MSSOAP.SoapReader30")

    Set doc = s.documentcontext
    SOAPin = doc.GetItemValue("Request_content")(0)

    soap.LoadXML SOAPin
    scs$ = soap.RPCParameter("dn").text
    MethodName = soap.RPCStruct.baseName
    NameSpace ="Domino"
```

Next we initialize the variant (soap) by creating the SOAPReader object and reading the current Web document, then extracting the request content. This is assigned to the SOAPin string we declared in the previous section. Then we use the LoadXML method of the SOAPReader object to load the request content into the variant. By doing this, we can then use the RPCParameter property of the SOAPReader to specify the attribute we need to fetch (in this case, dn). We then extract the text value of the attribute. After this is completed, we can extract the method that was called by the client by using the RPCStruct.baseName property of the SOAPReader. If the method called was correct, then we query the Place Catalog for the list of Places to which the user has membership. We return the My Places list as a string of URLs of the Places, separated by semicolons. The returned string is modified if the proper method wasn't called by the initiating SOAP client; in which case, we add the SOAP Fault:

```
    If MethodName = "GetMyQPs" Then
        Set curdb=s.currentdatabase
        Set db=s.getdatabase(curdb.server,"PlaceCatalog.nsf")
        Set v = db.getview("PlacesByMember")
        Set coll=v.getalldocumentsbykey(scs$)
        Set pdoc = coll.getfirstdocument
        ch$ = ""
        While Not pdoc Is Nothing
            If Len(ch$) < 1 Then
                ch$="http://"+pdoc.PlaceServerName(0)+"/"+pdoc.PlaceName(0)
            Else
                ch$=ch$+";"+"http://"+pdoc.PlaceServerName(0)+"/"+pdoc.PlaceName(0)
```



```
            End If

                Set pdoc=coll.getnextdocument(pdoc)
            Wend
            response=ch$
        rr$ = |<m:| & MethodName & "Response"  & | xmlns:m="| & NameSpace & |"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">| & _
        |<Answer xsi:type="xsd:string">| & response & |</Answer>| & _
        |</m:| & MethodName & |Response>|

    Else
        response = "Invalid Method Name used"
        rr$ = |<SOAP-ENV:FAULT><faultcode>Client.Authentication</faultcode> <faultstring>Invalid Methd
        Name</faultstring> <faultactor>http://localhost/</faultactor> <detail xsi:type="xsd:string"/>
        </SOAP-ENV:Fault> |
    End If
```

After this is completed, we then build the response string:

```
    Print "Content-Type: text/xml"
    strTmp = |<?xml version="1.0" encoding="UTF-8" standalone="no"?>| &_
    |<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">| & _
    |<SOAP-ENV:Body>| & _
    rr$ & _
    |</SOAP-ENV:Body>| & _
    |</SOAP-ENV:Envelope>|

    Print strTmp
    End
```

The My Places Web Service in action

After you create the Web Service for My Places, a SOAP-aware client can send a request to it similar to the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <m:GetMyQPs xmlns:m="http://localhost/">
        <dn>cn=admin,o=ibmtest</dn>
    </m:GetMyQPs>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The message is in XML format with the key information passed being the user name. This appears in the line bracketed by the <dn> attribute in the preceding code, which we've highlighted in bold text. The request is processed by our myplaceservice agent Web Service and sends a SOAP response back to the client:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
    <m:GetMyQPsResponse xmlns:m="http://localhost/">
        <Answer xsi:type="xsd:string">http://localhost/myqp1,http://localhost/testqp2</Answer>
    </m:GetMyQPsResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The response is also in XML format. The key information passed to the client to process appears in the bold lines of the preceding code bracketed by the <Answer> attribute.

Conclusion

We hope you found this article useful. Creating a Web Services interface for My Places extends the power of QuickPlace and makes its functionality more accessible to Web users. And this process can help you understand basic Web Services concepts, which can be applied to creating services for other applications as well.

ABOUT THE AUTHOR

Raj Balasubramanian is a Consulting IT Architect for IBM Software Services for Lotus. He works on customer engagements delivering application and infrastructure related projects. His interests range from anything technical to history and physics. During his copious spare time, he enjoys talking about robots with his sons.

