Building Web services using Lotus Domino 6

Presented by the Lotus Developer Domain

http://www.lotus.com/ldd

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

Section 1. Introduction

Should I take this tutorial?

You should take this tutorial if you want to develop Web services with Lotus Domino 6 and want to quickly learn how to use the Domino platform for building Web services. The tutorial assumes you understand how to navigate Domino Designer 6 and have a working knowledge of LotusScript. You do not need a background in Web services or an understanding of technologies such as the Simple Object Access Protocol (SOAP) and the Web Services Description Language (WSDL). The tutorial provides a brief introduction to each of these technologies. The tutorial also walks you through the underlying code, so you can quickly grasp how a Domino agent can be used to process SOAP messages.

What is this tutorial about?

Lotus has made a commitment to embrace Web services as a way to extend Notes and Domino's collaborative features across the enterprise. During this tutorial, we'll explore how the Domino 6 platform can be utilized today for building and deploying server-based Web services. Our discussion will be centered on a fictitious book distributor's need to make its book catalog available to its customers. The goal is to provide a basic understanding of Web services and to develop a framework that can be used within your own Domino applications. The sections in this tutorial present and review the following topics:

- **Overview of Web services**: Surveys the concepts and technologies that make up Web services and explores Lotus' strategy to embrace Web services.
- **Book distributor scenario**: Provides some context for how Web services can be utilized. The needs for a fictitious book distributor are examined. Additionally, the basic concepts of a SOAP message are presented.
- **Implementing the Domino agent**: Provides the details of how to develop and deploy Web services using Domino Designer 6.
- **Generating the WSDL file**: Provides a brief introduction to WSDL with an example WSDL file for the developed Web service.
- **Using the test client**: Presents a sample Domino database and Java application to illustrate the core concepts of Web services.

Tools

The following tools are necessary if you want to run the examples in this tutorial:

- Lotus Domino Designer 6. Download a *free trial version*.
- Lotus Domino 6 server for Windows. Download a *free trial version*.
- Download *Java 2 Runtime Environment 1.4.1_01*.

The sample code, bookcatalog.zip, for this tutorial is also available for download.

About the author

Jeff Gunther, a *Studio B* author, is the General Manager and founder of Intalgent Technologies, an emerging provider of software products and solutions utilizing the Lotus Notes/Domino and Java 2 Enterprise Edition (J2EE) platforms. Jeff Gunther has been a part of the Internet industry since its early, "pre-Mosaic" days. Prior to starting Intalgent, Gunther was the Director of Internet/Groupware Development at Southern Illinois Healthcare. He managed software development teams responsible for both end-user applications and mission-critical groupware infrastructure and applications. Additionally, he has professional experience in all aspects of the software lifecycle including specific software development expertise with Lotus Notes/Domino, Java/J2EE, DHTML, XML/XSLT, database design, and handheld devices.

Section 2. Overview of Web services

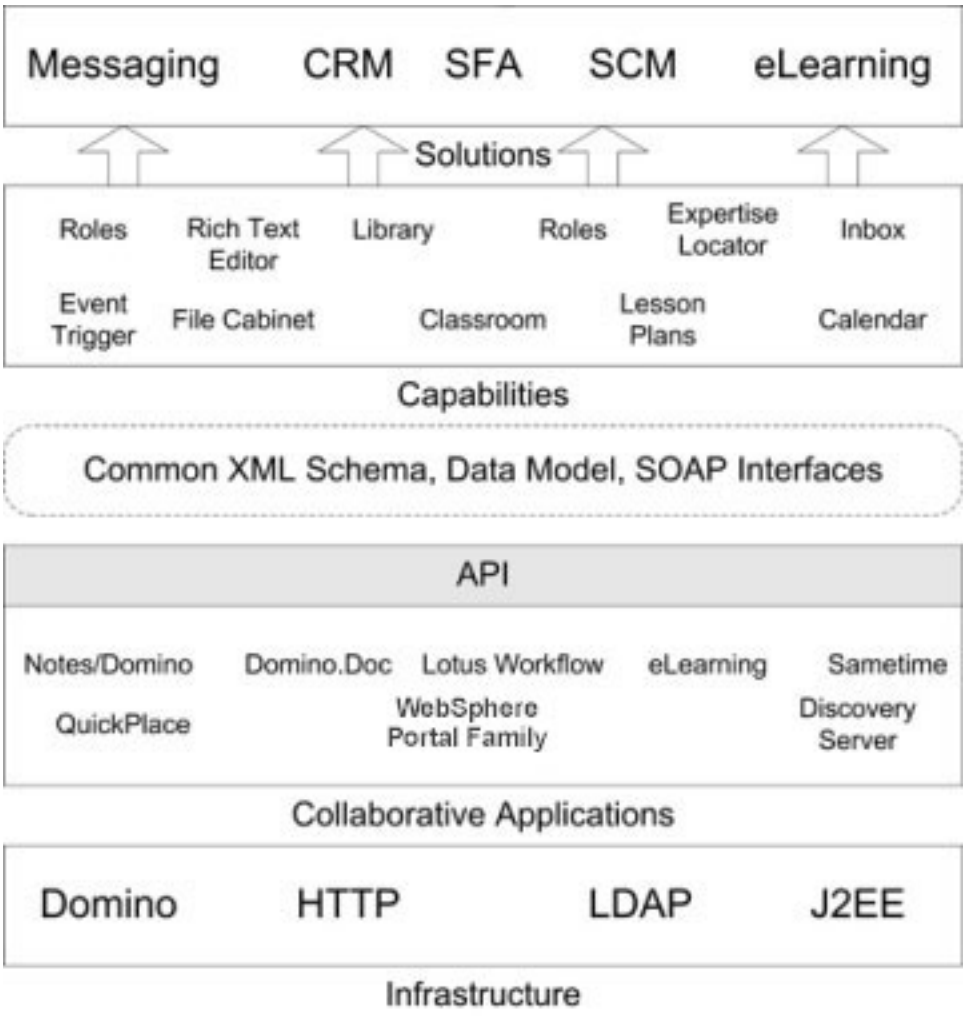
What is a Web service?

Web services, an emerging Web-based technology, allow developers to loosely couple software components across the enterprise. Developers can extend existing business processes as published services allowing other applications the ability to exchange information. Unlike traditional Web-based applications, Web services contain no user interface, but rather provide a unified way to execute business transactions across platforms. A series of technology standards are the basic components of Web services; they include:

- **XML** - The enabling force behind Web services is XML. XML is not a programming language but a platform-independent way to structure data. The syntax of XML makes it easy to programmatically manipulate textual data while still allowing it to be easily understood by humans. Web services use XML as the standard to provide communication between networked devices.
- **SOAP** - Simple Object Access Protocol (SOAP) provides developers a platform-independent mechanism to remotely invoke methods on distributed objects. The communication glue of a SOAP message uses XML for describing the object, method, and arguments to execute. Typically, SOAP messages are delivered to their intended targets using HTTP; however, SMTP can also be used as the communication protocol. Both clients and servers can implement and use SOAP. In this tutorial, a Domino server will act as the SOAP server and a local Java client will play the role of a SOAP client.
- **WSDL** - Web Service Description Language (WSDL) defines the available ports and operations for a particular Web service. You can think of a port as an interface and an operation as a method to be called on a particular object.

The Lotus vision for Web services

Lotus has announced a strategy to embrace Web services for application integration. The figure below illustrates the Lotus view of how Web services will be integrated into existing product lines.



On top of its existing core infrastructure and product lines, the Lotus strategy is to develop a common XML schema and data model to allow developers a unified way to interact with its products using Web services. Using these new tools, application developers can integrate Domino's collaborative capabilities into various enterprise applications.

Section 3. Book distributor scenario

Scenario overview

Before we discuss the details of how to implement a LotusScript Web service, let's create a business context for our discussion. BookCo is a fictitious international book distributor for hundreds of independent bookstores around the world. Typically, like most small to medium businesses, an independent bookstore doesn't have the resources to support a large technology infrastructure. Local booksellers are continually competing with large corporate bookstores and the ever-growing number of online retailers. To help their clients, BookCo has made the decision to extend all of their technology assets to the independent booksellers.

While BookCo has chosen Lotus Domino for their collaborative environment, any solution for the local bookstores needs to integrate seamlessly with their back-office systems. Some of the smaller booksellers have very simple needs, like looking up a book's details in BookCo's book catalog. In comparison, the larger independent booksellers are demanding an easy way to directly integrate with BookCo's purchasing and delivery departments. BookCo has decided to standardize on Web services as a simple, yet effective way to extend the enterprise's knowledge, data, and processes.

The ISBNSearch service

After completing an informal survey, BookCo found that many of its customers simply wanted an easy to use mechanism to view the details regarding a particular book and the number of available copies. During a survey of the larger book sellers, they found that many wanted to connect their own back-end systems to BookCo and didn't want to rely on a third-party application. Additionally, some of the smaller customers expressed an interest in an executable application on their desktops that could be used to gather information about the books in BookCo's catalog. Because all of the feature requests are possible using Web services, BookCo's software developers began to sketch out the design of their first Web service.

BookCo uses a variety of server-side technologies to streamline their operations. Particularly, BookCo uses Lotus Domino 6 to store, track, and manage the book catalog and the details regarding each book's availability. Given its developer's skill set with Domino, BookCo wanted to architect a Web services framework that was 100% pure LotusScript. They envisioned a Domino agent that would process the incoming request, parse the message, and send back a response. The outcome of their internal discussions was the ISBNSearch Web service. This Web service allows BookCo's clients to access the details of a book's availability by supplying an ISBN to the Domino server.

The next two panels review the anatomy of a SOAP message and explain how the contents are used to locate a book by ISBN.

Anatomy of a SOAP message

During our discussion, we'll hardly scratch the surface of SOAP. There are many resources online that will provide you more details regarding this technology (see Resources on page 21). In BookCo's ISBNSearch Web service, a SOAP message consists of four logical parts. They include the following:

- **SOAP Envelope** - The SOAP envelope tags are the outer-most boundaries of a SOAP message and mark the beginning and end of a message. Typically, the SOAP envelope includes attributes that define a namespace declaration. In the included implementation, the namespace declarations are ignored and not used.
- **SOAP Body** - Within the SOAP envelope is the SOAP Body. The SOAP Body includes the method signature to be executed and any arguments. A SOAP Body can contain only one child element.
- **Method signature** - Included within the SOAP Body is the method signature that will be executed on the Domino server. Additionally, the method signature contains the namespace where the method is located. For example, the ISBNSearch LotusScript function is located within a script library called `Domino`.
- **Method arguments** - The method arguments are found within the method signature tags. All of the contents of a method argument are passed to the LotusScript function. In the case of the ISBNSearch function, the particular ISBN is parsed from the SOAP message using Domino's new `NotesDOMParser` object.

The next panel includes a sample message to highlight each of these components.

Sample SOAP message for the ISBN search

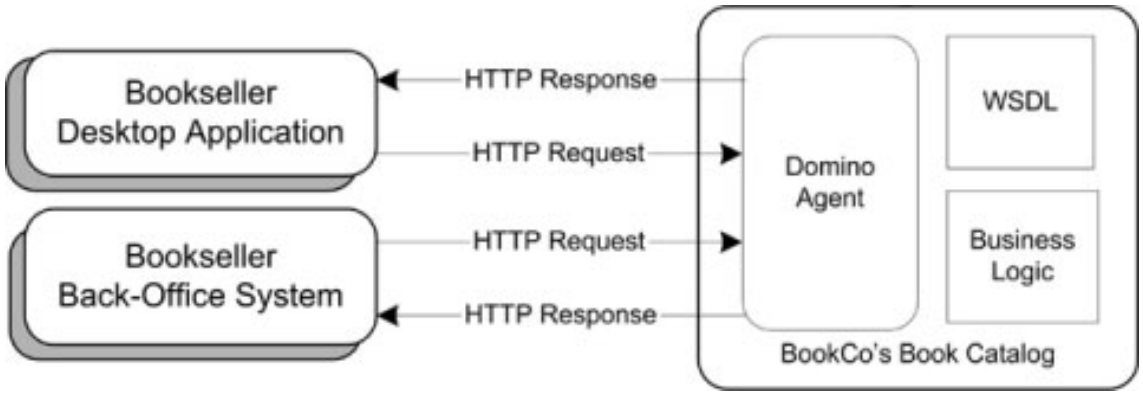
Below is an example of a SOAP message that will be consumed and processed by our Domino agent. Notice that the target book to search for is wrapped within the ISBN argument. The method name, namespace (Domino Script Library), and method argument are highlighted in bold.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
        <SOAP-ENV:Body>
                <m:ISBNSearch xmlns:m="uri:Domino"
                     SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
                        <isbn>0789728486</isbn>
```

```
            </m:ISBNSearch>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Architecture

To gain a high-level view of the application's components, the diagram below demonstrates the data flow and transport protocol being used.



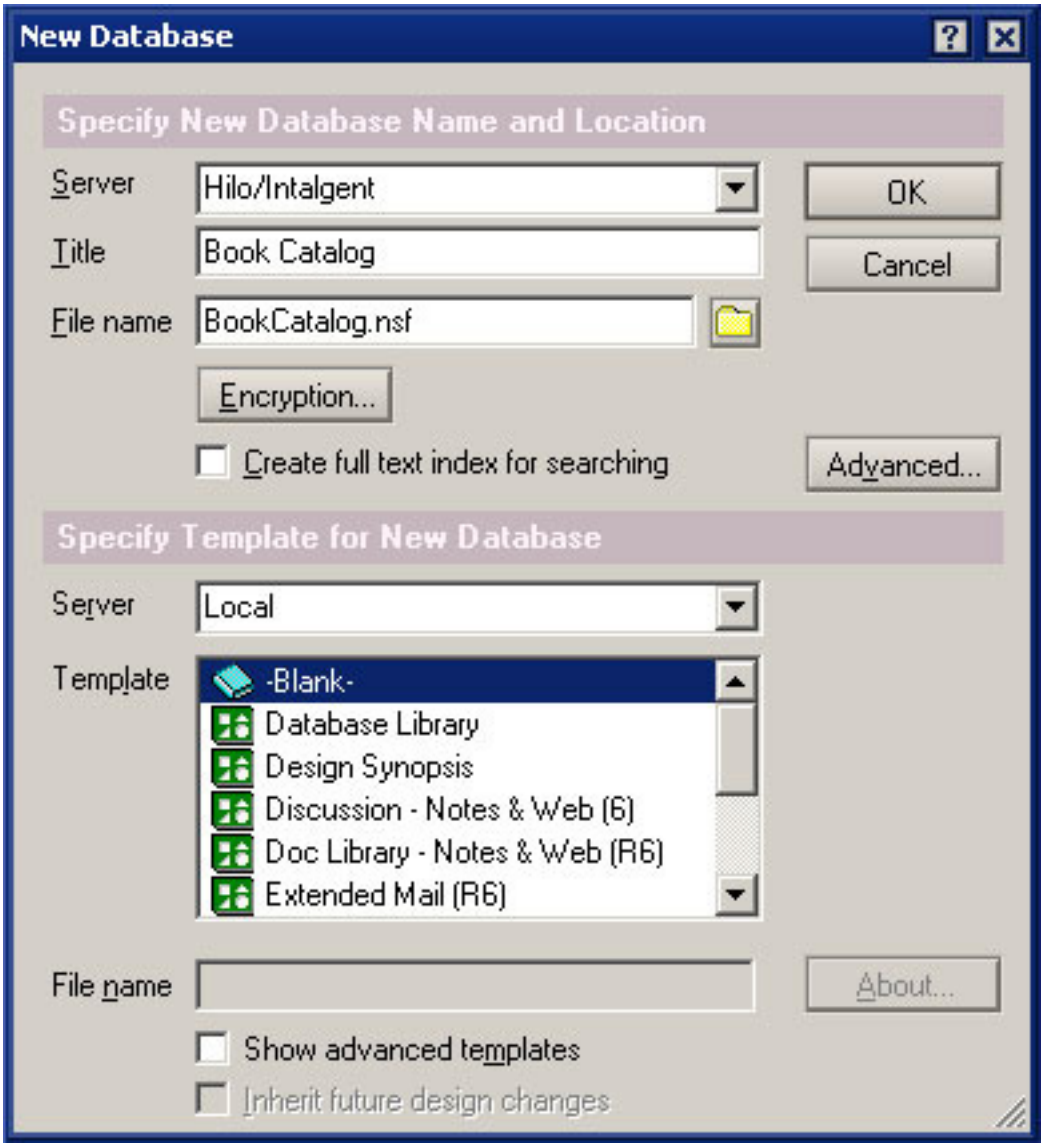
As illustrated in the diagram, whenever a bookseller wants to view a specific book's availability, the request is packaged as an XML message and sent across to BookCo's Domino server. After the request is received and the book is located within the book catalog database, the details regarding the book and its availability are sent back to the client.

Now that we have a basic comprehension of Web services, a shared understanding of the business requirements, and a pictorial view of the architecture, let's start working on a Domino-based implementation of the ISBNSearch Web service.

Section 4. Implementing the Domino agent

Creating a new database

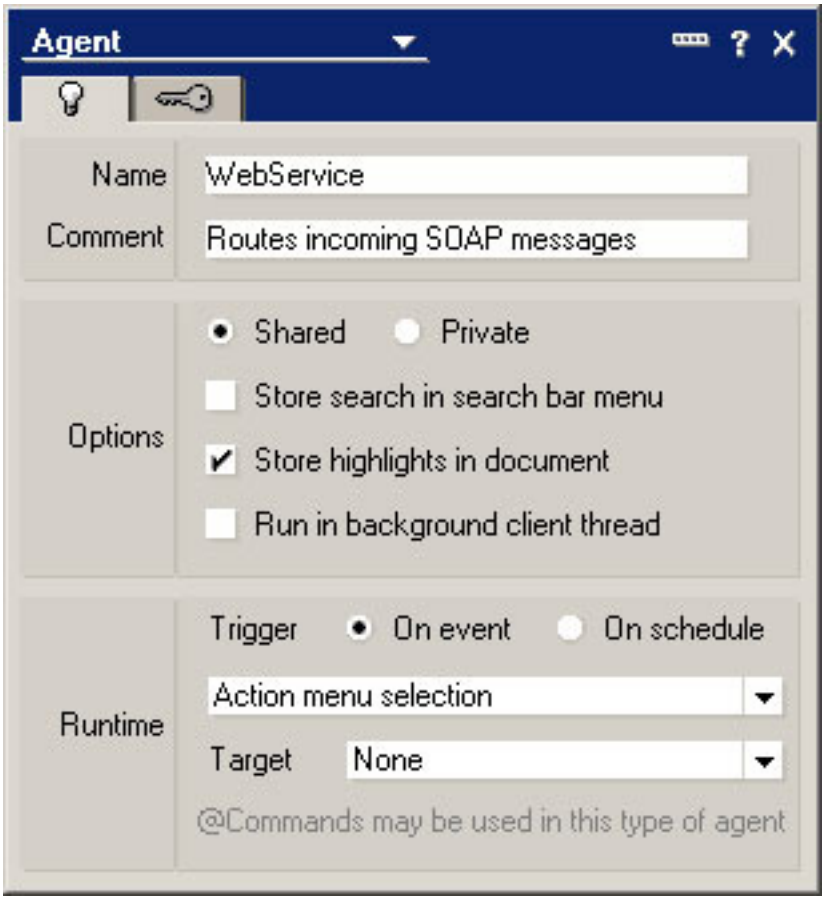
Up to now, we've covered the basics of Web services, and you should have an understanding of how SOAP messaging is handled between the Domino server and third-party clients. Let's get started with creating an application to meet the business requirements of our fictitious book distributor. As with any new Domino application, we need to create a new database. Open Domino Designer 6 and select **File=> Database=> New** from the menu bar. This opens the New Database dialog box:



Create the database on a Domino 6 server with a file name of **BookCatalog.nsf** and click **OK**. To test this Domino agent, it's important that the server and Domino Designer are running Notes/Domino 6. The agent in this tutorial uses the new XML parsing classes in Notes/Domino 6 to parse the incoming SOAP message.

Creating the Web service agent

After the database is created, we need to construct an agent to handle and route all incoming SOAP messages to the appropriate Domino script library and function. Open the database in Domino Designer and navigate to the Agent view under the Shared Code section. Once inside the Agent view, click the New Agent action button. When you first create an agent, you're prompted to complete the Agent properties box:



Name the agent `WebService` with a trigger event of **Action menu selection** and a target of **None**. Two more points are in order before diving into the code:

- A link to all of the code for this agent can be found in Tools on page 2 . If you use the

attached database on your server, make sure to sign it before trying the test client.
- Make sure that Default or Anonymous within the database's Access Control List has at least Editor access.

Processing the incoming SOAP message

Before we can act on a SOAP message, we need to receive the incoming XML and assign it to a variable of type String. Domino stores the incoming data in the **Request_content** item within the session's document context. Additionally, we need to set the context type for the response being sent back to the SOAP client:

```
Dim response As String
Dim incoming As String

Sub Initialize

Dim session As New NotesSession

Set doc = session.DocumentContext

'set the output content type to XML
Print "Content-Type: text/xml"

'get incoming SOAP message
incoming = doc.GetItemValue("Request_content")(0)
...
```

During development, it's helpful to monitor incoming messages for debugging purposes. The LogMessage helper function creates a copy of the incoming message as a Message document in the database. After the message is logged, a RemoveWhitespace helper function is used to remove all spaces, tabs, and new line characters:

```
...
'log incoming message
LogMessage(incoming)

'remove all whitespace from incoming SOAP message
incoming = RemoveWhitespace(Fulltrim(incoming))
...
```

Parsing the namespace, method, and arguments

Before we can call the appropriate LotusScript function, we need to parse the namespace, method, and arguments from the incoming SOAP message. The

`namespace` variable corresponds to the Domino script library that will be loaded:

```
...
'On Error Resume Next
On Error Goto Errhandle

bodyPos= Instr(1,incoming,|<SOAP-ENV:Body>|)+15

'parse out method signature
methodSigPos=Instr(bodyPos,incoming,|<|)+1
methodSigEnd=Instr(bodyPos,incoming,| |)
methodSignature=Mid(incoming,methodSigPos,(methodSigEnd-methodSigPos))

'parse out method name
methodPos=Instr(bodyPos,incoming,|:|)+1
methodEnd=Instr(methodPos,incoming,| |)
methodName=Mid(incoming,methodPos,(methodEnd-methodPos))

'parse out namespace
nameSpacePos=Instr(methodEnd,incoming,|uri:|)+4
nameSpaceEnd=Instr(nameSpacePos,incoming,|"|)
nameSpace=Mid(incoming,nameSpacePos,(nameSpaceEnd-nameSpacePos))
...
```

Script library and function

Now that we've determined the function to call within the script library, we need to construct a string that will be dynamically compiled and executed by Domino. In our example, the function ISBNSearch within the Domino script library will be executed. Be aware that the *incoming* variable, the flat representation of the SOAP message, is only available to other script libraries if declared Public.

```
...
'Build a string containing the script library, function, and incoming soap message
'The nameSpace variable contains the script library to load
callString  = |Use | & |"| & nameSpace & |"| & |
response  = | & methodName & |(incoming)|

'execute the function
Execute callString
...
```

Sending a response to SOAP the client

In the previous code snippet, the ISBNSearch function was executed and the `response` variable was populated with XML data. The code below wraps the response from the function in a SOAP message for the receiving client:

```
...
'create response
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>| & _
|<m:| & methodName & "Response"  & | xmlns:m="uri:| & nameSpace &
|" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">| & response & _
|</m:| & methodName & |Response>| & _
|</SOAP-ENV:Body>| & _
|</SOAP-ENV:Envelope>|
...
```

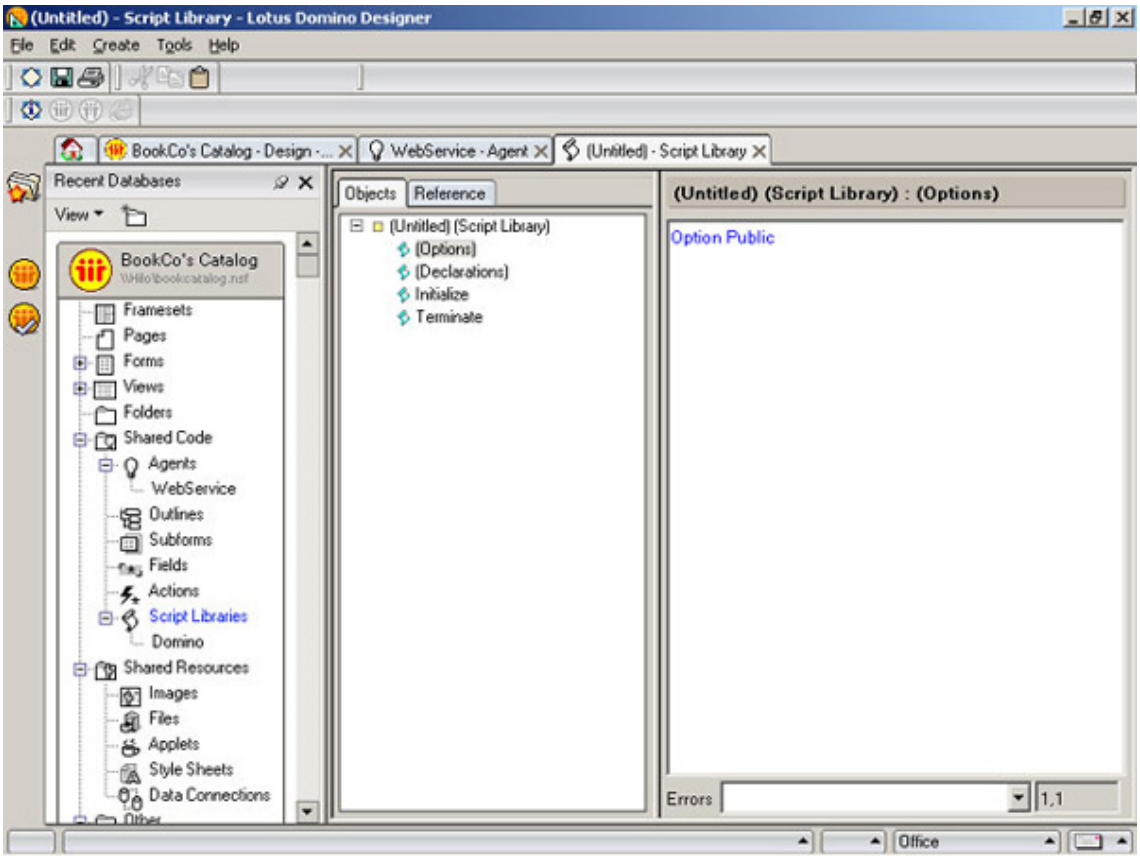
Now it's time to send the SOAP response to the client:

```
...
'send back SOAP response
Print strTmp
...
```

That's it! We've covered the lifecycle of the Domino agent. Now let's review the ISBNSearch function that executed our business logic and created the XML data for the response to the SOAP client.

Creating a new script library

As mentioned in Anatomy of a SOAP message on page 7 , the ISBNSearch function is stored in a script library. Open the database in Domino Designer 6 and navigate to the Script Libraries view under the Shared Code section. Once inside the Script Libraries view, click **New LotusScript Library**. This opens a new script library:



Parsing the SOAP message for the ISBN

Let's recap. We've covered how the Domino agent routes incoming messages, executes the appropriate function, and sends a response back to the client. However, we haven't covered the actual ISBNSearch function that was executed. Before the details regarding a book's availability can be sent back to the client, we need to parse the ISBN tag in the SOAP message. The new `NotesDOMParser` object in Notes/Domino 6 is used to complete the XML parsing.

```
Dim domParser As NotesDOMParser

Function ISBNSearch(msg As String) As String

        Dim session As New NotesSession

        Dim rootElement As NotesDOMDocumentNode
        Dim nodeList As NotesDOMNodeList
        Dim node As NotesDOMNode

        'create dom parser to process SOAP message
```

```
        Set domParser = session.createDOMParser(msg)
        domParser.process

        Set rootElement = domParser.Document

        Set nodeList = rootElement.GetElementsByTagName("*")

        'locate ISBN
        For i=1 To nodeList.NumberOfEntries
                Set node = nodeList.GetItem(i)
                If(node.NodeName="isbn")Then
                        ISBN = node.FirstChild.NodeValue
                        Exit For
                End If
        Next
...
```

Locating the book in a Domino view

Because we have the actual value for the ISBN in question, we can look up the book's availability. If you use LotusScript on a daily basis, the code below should look familiar. Given the ISBN, we locate the book's document in a view, create a response containing XML data for the SOAP client, and return the result to the calling agent.

```
...
        Dim db As NotesDatabase
        Dim view As NotesView
        Dim doc As NotesDocument

        Set db = session.CurrentDatabase
        Set view = db.getView("ByISBN")

        'locate ISBN in database
        Set doc = view.GetDocumentByKey(ISBN,True)

        'create response
        tmp =|<isbn xsi:type="xsd:string">| & doc.ISBNTX(0) &|</isbn>|&_
        |<title xsi:type="xsd:string">| & doc.TitleTX(0) &|</title>|&_
        |<price xsi:type="xsd:string">| & doc.ListPriceNU(0) &|</price>|&_
        |<authors xsi:type="xsd:string">| & doc.AuthorsTX(0) &|</authors>|&_
        |<publisher xsi:type="xsd:string">| & doc.PublisherTX(0) &|</publisher>|&_
        |<copies xsi:type="xsd:string">| & doc.CopiesTX(0) &|</copies>|

        ISBNSearch = tmp

End Function
```

Section 5. Generating the WSDL file

Overview of WSDL

Web Services Description Language (WSDL) is one of the core components of Web services. This technology is an abstract view describing the operations, parameters, and details regarding a particular Web service. The discussion of WSDL is outside the scope of this tutorial; however, many Web services development frameworks use WSDL files. For example, an application could interact with a WSDL file at runtime without any understanding of the structure of the Web service being executed. The WSDL file for the ISBNSearch Web service is provided next.

ISBNSearch.wsdl

Here is the WSDL file for the ISBNSearch Web service.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    name="ISBNSearch"
    targetNamespace="http://www.your-company.com/ISBNSearch.wsdl"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.your-company.com/ISBNSearch.wsdl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd1="http://www.your-company.com/ISBNSearch.xsd1">
    <types>
        <xsd:schema
            targetNamespace="http://www.your-company.com/ISBNSearch.xsd1"
            xmlns="http://schemas.xmlsoap.org/wsdl/"
            xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"> </xsd:schema>
    </types>
    <message name="ISBNSearch">
        <part name="isbn" type="xsd:string"/>
    </message>
    <message name="ISBNSearchResponse">
        <part name="title" type="xsd:string"/>
        <part name="isbn" type="xsd:string"/>
        <part name="price" type="xsd:string"/>
        <part name="authors" type="xsd:string"/>
        <part name="publisher" type="xsd:string"/>
        <part name="copies" type="xsd:string"/>
    </message>
    <portType name="ISBNSearchPortType">
        <operation name="ISBNSearch">
            <input message="tns:ISBNSearch"/>
            <output message="tns:ISBNSearchResponse"/>
        </operation>
```

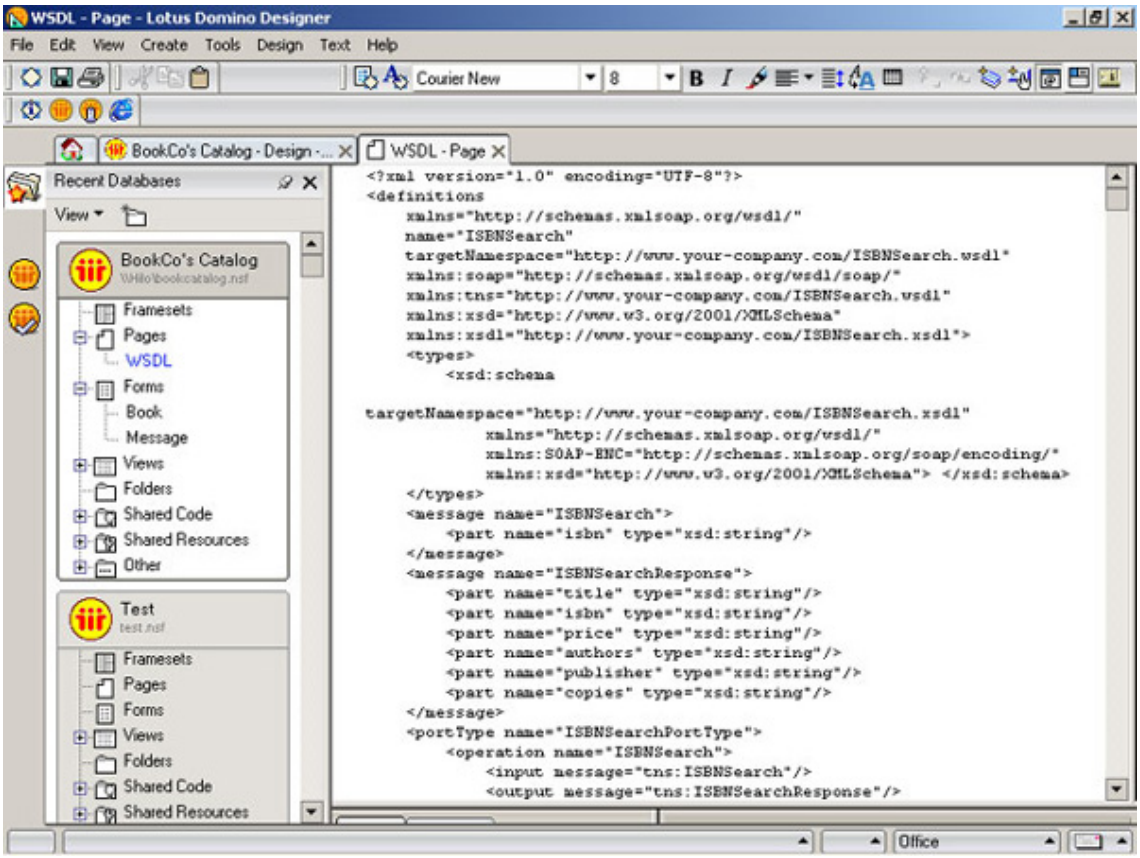
```
    </portType>
    <binding name="ISBNSearchBinding" type="tns:ISBNSearchPortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="ISBNSearch">
            <soap:operation
                soapAction="capeconnect:ISBNSearch:ISBNSearchPortType#ISBNSearch"/>
            <input>
                <soap:body
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="http://www.your-company.com/ISBNSearch/binding"
                    use="encoded"/>
            </input>
            <output>
                <soap:body
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="http://www.your-company.com/ISBNSearch/binding"
                    use="encoded"/>
            </output>
        </operation>
    </binding>
    <service name="ISBNSearch">
        <port binding="tns:ISBNSearchBinding" name="ISBNSearchPort">
            <soap:address location="http://192.168.1.2/bookcatalog.nsf/WebService"/>
        </port>
    </service>
</definitions>
```

Details of the ISBNSearch WSDL file

Before continuing on with the test client, let's briefly highlight a few items of this WSDL file. The *message* elements itemize the types of messages that are used during the lifecycle of a Web service. In our example Web service, the ISBNSearch and ISBNSearchResponse messages are used between the client and the server. Within a particular message, you'll find one or more *part* elements that describe the individual data components used by each message type. The *binding* element describes the transport protocol that is used by this Web service. In this example, HTTP is the protocol being used. Notice that the actual URL for our Web service is not listed in the binding element, but rather found in the *service* element.

Storing the WSDL

Depending on the platform you're using, you need access to a Web service's WSDL file before you can start using it in production. Given the nature of the data, you can simply add the contents of the WSDL file to a Domino page in your database:



To access this WSDL file from another development platform, simply use the following URL:

`http://<someserver>/bookcatalog.nsf/WSDL?OpenPage`

Section 6. Using the test client

Creating the test client

The test client for our Domino Web service was developed in Java 1.4 using the Eclipse Open Source Integrated Development Environment. You can find more information about Eclipse and the source code for the client in Resources on page 21 . The discussion of using Eclipse for developing Web services in Java is outside the scope of this tutorial; however, it's helpful to see how the SOAP library from the XML Apache project can be used.

Coding the ISBNSearchProxy Java class

After a Java URL object has been instantiated based on the provided URL for the Domino Web service agent, it only takes nine lines of code to call the Web service and receive a response back. The code below uses the SOAP library from the XML Apache project.

```
...
Call call = new Call();
call.setMethodName("ISBNSearch");
call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
call.setTargetObjectURI("uri:Domino");

Vector v = new Vector();
Parameter parameter = new Parameter("isbn", String.class, bookISBN,
Constants.NS_URI_SOAP_ENC);
v.addElement(parameter);
call.setParams(v);


//call web service
Response response = call.invoke(getURL(), "ISBNSearch");

if (response.generatedFault())
{
Fault fault = response.getFault();
        throw new SOAPException(fault.getFaultCode(), fault.getFaultString());
}
else
{
        return (response.getParams());
}
...
```

Launching the test client

We've covered a lot of ground, but maybe you're still skeptical about how all this works together. Before you run the test client, verify that the following steps have been completed:

- If you used the database provided rather than developing it along the way, make sure you sign the database using the Domino Administrator client.
- Make sure that Default or Anonymous in the database's Access Control List has at least Editor access.
- Make sure the Java 1.4 Runtime Environment is installed on your local workstation.

Unpackage the test client, access a command line, and type the following command from within the `Client` directory:

```
Tester http://<someserver>/bookcatalog.nsf/WebService 0789728486
```

If everything is configured properly, you should receive the details about my new book entitled *Special Edition: Using Lotus Notes and Domino 6*.

Section 7. Summary

Summary

As Lotus begins to implement its Web services strategy, it's exciting to explore how Domino 6 can provide LotusScript developers an integrated environment to develop and deploy Web services today. The sample database provides a scalable framework that can be used with your own databases to extend Domino's collaborative features to other applications. The following topics were presented during our discussion:

- An introduction to the core technologies that make up Web services including: XML, SOAP, and WSDL
- An exploration of how to develop a LotusScript Domino agent that can be used to route and process SOAP messages
- The new Domino 6 XML parsing classes
- A sample Java application that can both send and receive SOAP messages

---

Resources

The tools and code used throughout this tutorial include:

- *Lotus Domino Designer 6* trial version
- *Lotus Domino 6 server for Windows* trial version
- *Java 2 Runtime Environment 1.4.1_01*
- Domino database for the book distributor scenario
- Java client source code and binaries for the book distributor scenario

Resources for the technologies utilized or referenced in this tutorial:

- The *Apache XML project* provides a variety of XML tools including a SOAP implementation that was used within the client.
- The *Eclipse project* is aiming to provide an Open Source Integrated Development Environment for many programming languages including Java.
- For an excellent introduction to Web services and the impact they have on the Web, see Doug Tidwell's tutorial *Web services - The Web's next revolution* .
- For a detailed look at the parts of a SOAP message and the way in which it's transmitted over HTTP, see the developerWorks tutorial *XML Messaging with SOAP*.
- For an article introducing Web services and WSDL, read *Deploying Web services with WSDL, part 1*.

Feedback

Please send us your feedback on this tutorial. We look forward to hearing from you!

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT style sheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial Building tutorials with the Toot-O-Matic demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.