

Quickly create Domino Web services

New Web services function in Domino 7 speeds development

Skill Level: Intermediate

[Andrei Kouvchinnikov \(andrei@mindspot.se\)](mailto:andrei@mindspot.se)
Consultant in Domino and collaboration solutions
Mindspot AB

15 Mar 2005

Updated 26 Sep 2005

This tutorial shows you how to use the new Web services design element in IBM Lotus Domino 7 for rapid creation of Web services in the LotusScript and Java programming languages. The tutorial walks you through examples of a business scenario involving a fictitious software company. The examples use a Web service that allows multiple clients to search a Domino Directory database. Using IBM Lotus Domino Designer, you learn how to develop a native Domino Web service and test the newly created service from a LotusScript agent and a Java application.

Section 1. Before you start

About this tutorial

This tutorial shows you how to use the IBM® Lotus® Domino® 7 platform to build Web services. You'll learn how to create a Web service that accesses information stored in a Domino Directory database's person documents. You'll also learn how to create a Web service that sends Domino-based e-mail using parameters provided from an external system, then test your Domino-based Web service.

This tutorial is designed for Web developers who want to leverage the Lotus Domino

platform to build Web services. This tutorial assumes that you have access to and understand how to navigate IBM Lotus Domino Designer® 6 or Domino Designer 7. You also need a working knowledge of the LotusScript or Java™ scripting languages. You do not need a background in Web services or an understanding of technologies such as the eXtensible Markup Language (XML), the Simple Object Access Protocol (SOAP), or the Web Services Description Language (WSDL), although a brief introduction to them is provided.

This tutorial might also be useful for developers working with non-Domino systems who are looking for ways to integrate their existing applications with services that Lotus Domino provides, such as messaging, workflow, and information stored in databases.

Prerequisites

While not a prerequisite, this tutorial is easier to follow if you have downloaded, installed, and configured both Domino Designer 7 and the Java 1.4.x Software Development Kit (SDK). IBM Lotus Domino Server 7 is not required for completing this tutorial if you choose to preview your Web services locally using the built-in HTTP server provided with Domino Designer. However, to share your new Lotus Domino 7 Web services with other users and applications, you need a computer running Lotus Domino Server 7. If you don't have access to these tools, see the [Resources](#) section at the end of this tutorial for information about downloading evaluation versions.

The sample code for this tutorial is also available for download. Refer to the [Resources](#) section for the Domino database (WebservicesR7.zip) and the Java client source code and binaries (WSIFclient.zip).

Section 2. Overview of Web services

What are Web services?

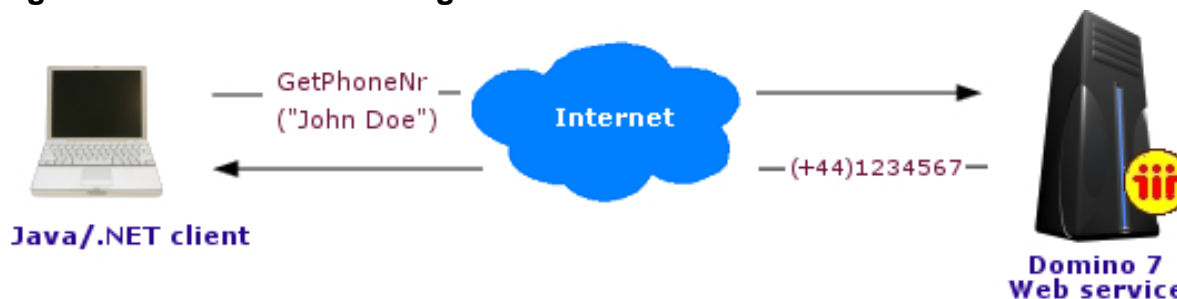
A *Web service* is an application or business logic that's programmatically accessible through standard Internet protocols. Web services technology is based on open standards, so your shared server-side application functions can be available to virtually any client, regardless of language or platform. The goal is to enable one application to contact another application that provides a requested service and to seamlessly exchange data with it.

The basic components of Web services are based on a series of technology standards. You don't need to know the details about these standards for purposes of creating Web services in Lotus Domino 7. But if you want to use services that other developers have created, you need to have a working knowledge of at least the four most important standards:

- **XML:**XML is designed to improve the functionality of the Web by providing more flexible and adaptable information identification. In Lotus Domino 7, the XML structure is generated automatically based on data types you define in your LotusScript and Java classes.
- **SOAP:**SOAP is a simple XML-based communication protocol designed to let applications exchange information over HTTP. It is the basic component of Web services. (Some people prefer to call this standard *Services-Oriented Architecture Protocol*.)
- **WSDL:**WSDL uses XML to describe the structure of Web services. In Lotus Domino 7, the WSDL file is generated automatically or can be imported.
- **Universal Description, Discovery, and Integration (UDDI):**UDDI is a Web-based directory that enables businesses to list themselves and discover each other on the Internet.

Figure 1 provides a schematic of information flow between a client sending a request and a server returning a computed response based on received request parameters.

Figure 1. Information exchange



The WSDL structure

Listing 1 shows the core content of a WSDL document for an e-mail-providing Web service.

Listing 1. WSDL document for an e-mail-providing Web service

```
<wsdl:message name="MAILSENDRequest" >
```

```

(1)
<wsdl:part name="SENDTO" type="xsd:string" /> (2)
<wsdl:part name="COPYTO" type="xsd:string" />
<wsdl:part name="BLINDCOPYTO" type="xsd:string" />
<wsdl:part name="SUBJECT" type="xsd:string" />
<wsdl:part name="BODY" type="xsd:string" />
</wsdl:message>
<wsdl:message name="MAILSENDResponse">
  <wsdl:part name="MAILSENDReturn" type="xsd:string" /> (3)
</wsdl:message>
<wsdl:portType name="SimpleMailer"> (4)
  <wsdl:operation name="MAILSEND"
    parameterOrder "SENDTO COPYTO BLINDCOPYTO SUBJECT BODY"> (5)
    <wsdl:input message="impl:MAILSENDRequest"
      name="MAILSENDRequest" />
    <wsdl:output message="impl:MAILSENDResponse"
      name="MAILSENDResponse" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="DominoSoapBinding"
  type="impl:SimpleMailer"> (6)
  <wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" /> (7)
  <wsdl:operation name="MAILSEND">
    <wsdlsoap:operation soapAction="" /> (8)
    <wsdl:input name="MAILSENDRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:DefaultNamespace" use="encoded" />
    </wsdl:input>
    <wsdl:output name="MAILSENDResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:DefaultNamespace" use="encoded" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="SimpleMailerService"> (9)
  <wsdl:port binding="impl:DominoSoapBinding" name="Domino"> (10)
    <wsdlsoap:address location=
      "http://localhost/ws/WebServicesR7.nsf/Simplemailer?OpenWebService" />
  </wsdl:port>
</wsdl:service>

```

Let's look at the various parts of this document. Each number below refers to the corresponding numbers in the WSDL document.

1. The message consists of several `<part>` tags that refer to parameters or return values. Set the `requestMethod` to accept five parameters.
2. Assign a name and data type to all parts of the message.
3. Set the response to return one value of data type `String`.
4. The `portType` element specifies a set of operations that the Web service supports.
5. This line specifies the `MailSend` operation (function) and the order of parameters for this operation.
6. A binding defines the message format and the protocol details for

operations and messages defined by a particular portType.

7. This binding specifies the style of interaction (Remote Procedure Call, or RPC) and the transport protocol used (HTTP).
8. Use the optional `soapActionHTTP` request header field to indicate the intent of the SOAP HTTP request.
9. Domino Designer 7 generates one service and one port for the WSDL file or, if importing a WSDL file, processes only the first port of the first service element encountered.
10. A port defines an individual endpoint by specifying a single address for a binding.

The SOAP structure

Listing 2 shows the SOAP structure for a response that the e-mail-providing Web service generated.

Listing 2. SOAP response generated by the e-mail-providing Web service

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> (1)
  <soapenv:Body> (2)
    <ns1:MAILSENDResponse soapenv:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="urn:DefaultNamespace"> (3)
      <MAILSENDReturn xsi:type="xsd:string">OK</MAILSENDReturn> (4)
    </ns1:MAILSENDResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

1. The whole message is contained within an `Envelope` element. `Envelope` contains a `Body` element and an optional `Header` element.
2. SOAP `Body` contains the payload of the message, carrying the data that represents the purpose of the message.
3. This line defines the output operation `MAILSENDResponse` and sets its encoding type and namespace. If not otherwise specified, Lotus Domino 7 uses `urn:DefaultNamespace` for its namespaces.
4. The value for the output part of the message is a single string `OK`.

Section 3. Web services in Lotus Domino 7

Web Services: A new design element in Lotus Domino 7

Web Services is a new design element in Lotus Domino 7 that you can program by using either the LotusScript or Java programming languages. You can find Web Services in the Shared Code section of the database design pane. Web Services design element implementation looks much like regular Domino agent design and has the same programming logic.

The same basic security rules apply to Web services as apply to agents:

- A database access control list (ACL) for verifying access
- Run as Web User/as Signer/on Behalf of
- Restricted/Unrestricted access to resources
- Public Access option

Creating a new Web service is as easy as clicking **New Web Service** in the action bar of the Web services designer panel (see Figure 2).

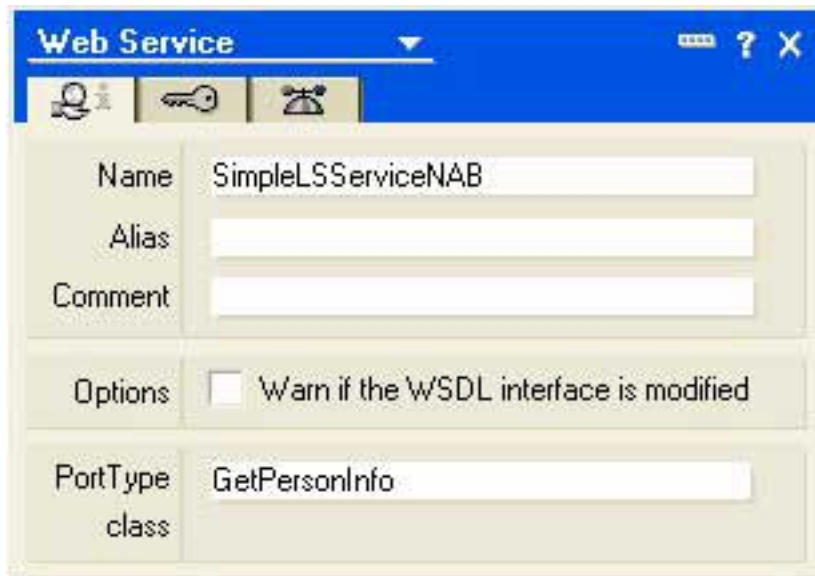
Figure 2. The Action button for creating a new Web service



The Web Services Properties window

When you create a new Web service, you can set its basic behavior and security properties in the service's Properties window. The process is similar to that for Domino agents. Figure 3 shows the Basics tab of the Properties window for the simple LotusScript Web service you create later in the tutorial.

Figure 3. The Basics tab



On the Basics tab, you can set the Web service's name, alias, and PortType class. Notice the **Warn if the WSDL interface is modified** option: Selecting this option causes the service to display a message if the implied WSDL structure changes, such as revised class names, changed names of methods or functions, or added or removed class variables. In a production environment, such changes can cause existing software clients consuming your Web service to stop working because of errors in parsing the changed Web service response structure. Setting this option also prevents you from saving such changes and gives you a chance to review and change your code back. You don't need to set this option during the creation and testing period.

In the **PortType class** field, specify the class that contains the operations (methods and functions) available from this particular service. If you try to save the Web service without setting this field first, the Domino Designer 7 client generates a warning.

The Security tab, shown in Figure 4, looks the same as the Security tab in an agent's properties. Here, you can set security options for your Web service.

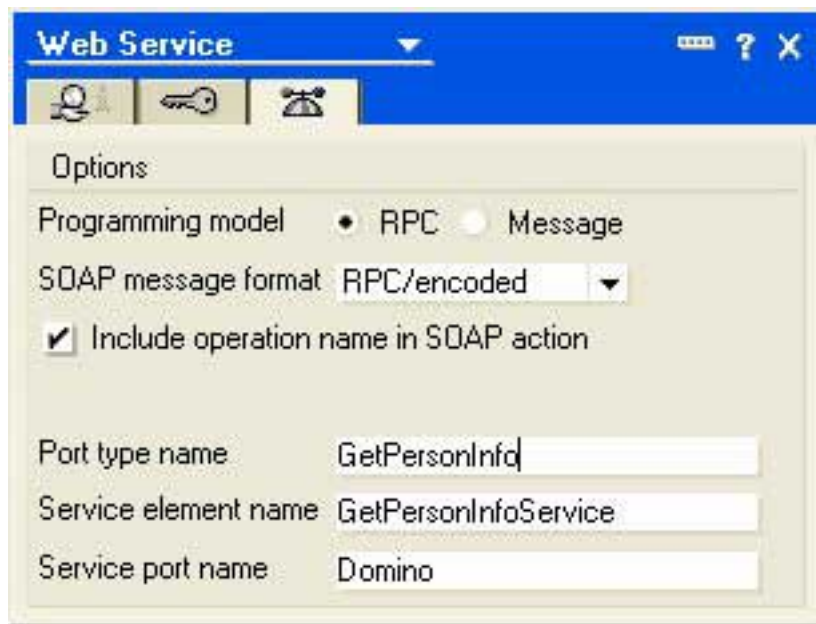
Figure 4. The Security tab



The **Allow remote debugging** option is for LotusScript Web services, and it generates code suitable for remote debugging. For Java Web services, another option called **Compile Java code with debugging information** is available that makes it possible to see in the console at which code line the runtime error occurred. The **Profile this web service** option allows the collection of run time statistics for profiling.

The Advanced tab, shown in Figure 5, specifies the programming model and SOAP message format.

Figure 5. The Advanced tab



You can select between two programming model options: RPC and Message. RPC supports conventional method parameters and return types; Message limits the method interface to a single "message" parameter and return type (see the online documentation for more information). For the RPC programming model, the RPC/encoded SOAP message format is the default, but you can change it if you have special requirements to use another format. For the Message programming model, the SOAP message format is fixed at Doc/literal. For more information about the choice of SOAP message formats, see the article *Which style of WSDL should I use?* by Russell Butek listed in the [resources](#) section.

The **Port type name**, **Service element name**, and **Service port name** fields are populated automatically when you save the Web service. In the **Port type name** field, specify the name of the port type for accessing the service. This specification corresponds to the `name` attribute for `<wsdl:portType>` in the WSDL document. Lotus Domino 7 recognizes one port type per service.

In the **Service element name** field, you specify the name of the service. This specification corresponds to the `name` attribute of `<wsdl:service>` in the WSDL document. In the **Service port name** field, you specify the port for accessing the service. This specification corresponds to the `name` attribute of `<wsdl:port>` under `<wsdl:service>` in the WSDL document. Lotus Domino 7 recognizes one port per service.

The next section shows you how to use Web services to integrate your Domino Directory database with other systems. You'll also learn how to return information stored in the Domino Directory database and how to send Domino mail.

Section 4. Create a Lotus Domino 7 Web services

Scenario overview

XimartCo is a large software development company with 400 employees. The company has principle offices in three cities and several small offices around the world. The company headquarters has a Lotus Domino server used for mail purposes and for hosting Lotus Notes clients and Web applications. All the local offices have access to the Internet through broadband connections. Some offices were acquired several months ago and still haven't completely implemented Lotus Domino or established Virtual Private Network (VPN) connections to the main office. These offices still use their old intranet systems based on Microsoft® Internet Information Services (IIS) and Apache for information exchange inside the company.

Many of the offices want to find a way to integrate their Address Books with the corporate Domino Directory database hosted at the company's headquarters to facilitate information searches about their colleagues at other offices. Also, many requests have come from the Research & Development department to integrate dynamic Address Book lookups into their Java applications. Besides that, many developers want to use the Domino mail server as an engine for sending mail and automated notifications from their Microsoft Visual Basic® and Java applications.

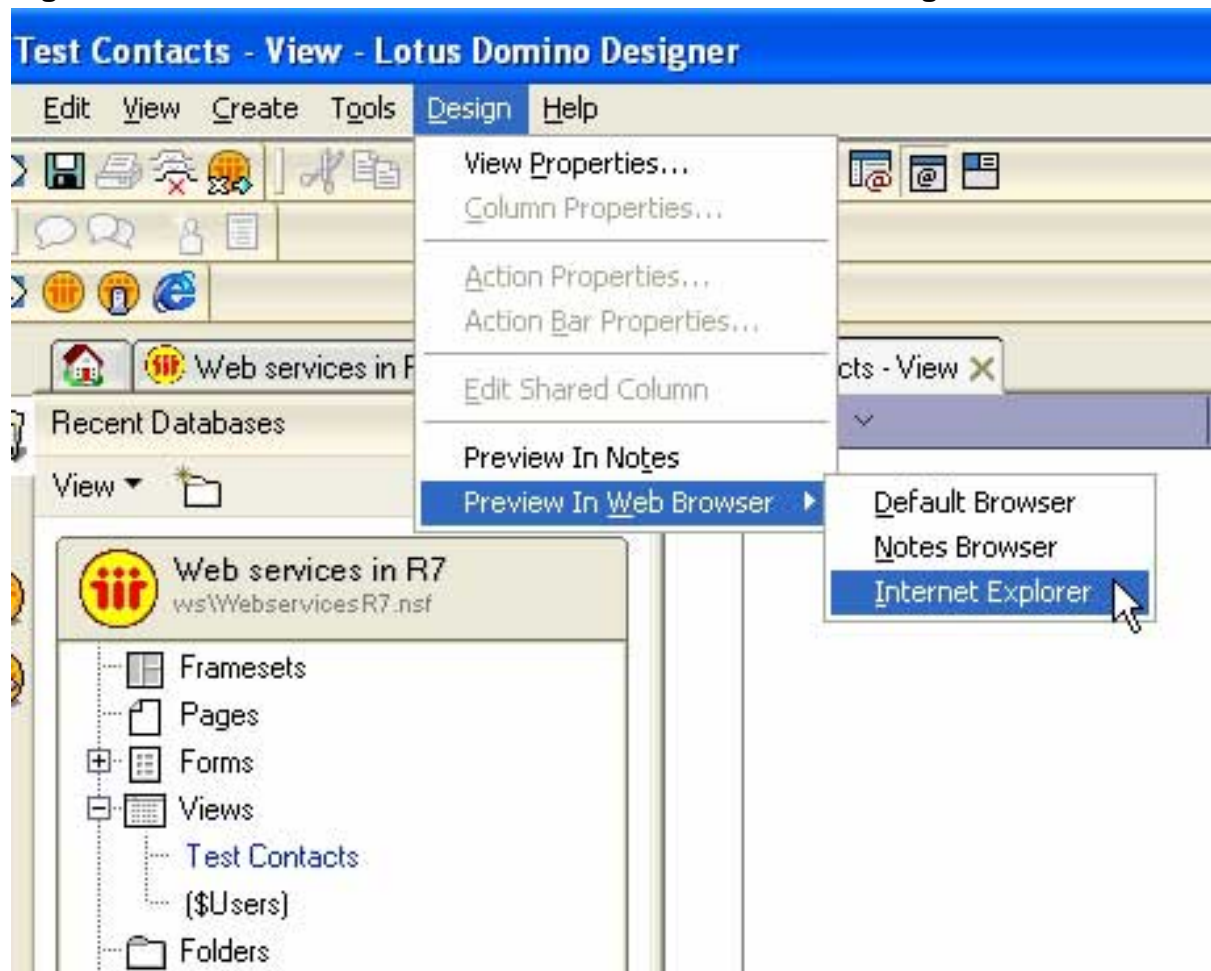
To meet the growing need of real time integration between Lotus Domino and other systems, XimartCo has made the decision to implement access to the Domino Directory database through Web services. To begin with, they decided to allow external systems to make read-only lookups to the Domino Directory database for particular people and to allow the system to send mail programmatically by using Domino mail functionality. For better support of different systems and the possibility of applying the same solution later to interested customers who have similar infrastructures, several Web services need to be created for invocation from Visual Basic-compatible clients and Java clients.

In this tutorial, you create simple and complex LotusScript and Java Domino Web services to fulfill XimartCo's needs. By *complex Web service* I mean a Web service that accepts or outputs an array data type instead of a single text value. In the examples that follow, I begin with a LotusScript implementation, then provide the Java implementation of same Web service.

To test your Web service in a Domino Designer 7 client, preview any of your forms or views in a Web browser (see Figure 6). This procedure starts an internal Web server, which is sufficient for our testing needs. (Note that you don't need to perform

such an operation if your database resides on a Lotus Domino server.)

Figure 6. Start an internal Web server from the Domino Designer client



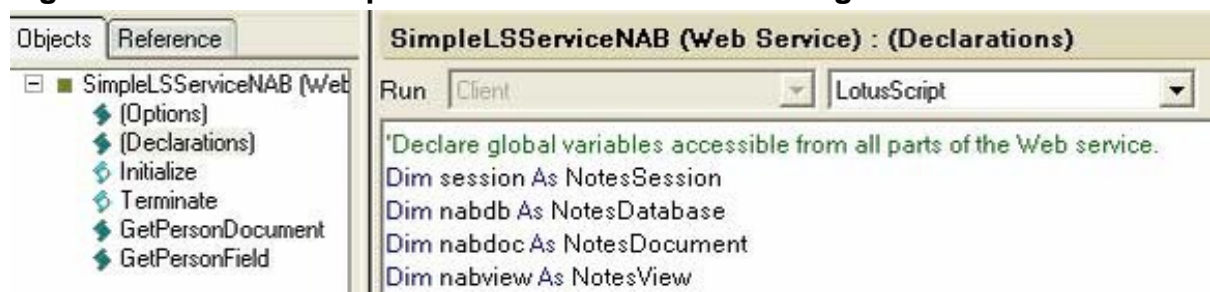
A simple Web service in LotusScript

Follow the steps below to create a LotusScript Web service that returns a single text value. This value can be any field from any database; I used the `MailAddress` field from my Domino Directory database.

1. Create a new Lotus Domino 7 database or open an existing database.
2. Create a new Web service in Domino Designer 7.
3. Select **LotusScript** as the programming language for your Web service. (Note that LotusScript is the default language.)

4. In the service's Properties window, type SimpleLSServiceNAB in the Name field.
5. In the PortType class field, type GetPersonInfo.
6. Paste the LotusScript code shown in Listing 3 into the Declarations section of the Web service (see Figure 7).
7. Paste the GetPersonDocument and GetPersonField functions into the Declarations section of the Web service.
8. Save the Web service.

Figure 7. The LotusScript Web service in Domino Designer



See the comments inside the code for a description of code functionality.

Listing 3. Code for the simple LotusScript Web service

```
'Declare global variables accessible from all parts of the Web service.
'Declare global variables accessible from all parts of the Web service.
Dim session As NotesSession
Dim nabdb As NotesDatabase
Dim nabdoc As NotesDocument
Dim nabview As NotesView
Dim item As NotesItem

'Create a LotusScript class. Sub New can be used to assign values to global
'variables. Here we can set our session object.
Class GetPersonInfo
  Sub New
    Set session = New NotesSession
  End Sub

'Now we can create the main part of our Web service: GetEmailAddress function and
'GetPhoneNumber function. Both functions work in a similar way, so we will look
'only at GetEmailAddress function.
Public Function GetEmailAddress(personname As String) As String
'We declared function and set it to accept 1 parameter.
'We set output type to data type String.
  If nabdoc Is Nothing Then
'Locate a document using our own reusable search function
    result=GetPersonDocument(personname)
  End If
  If result<>"OK" Then
```

```

'If no person document is found, return an error text generated in the
'GetPersonDocument function
  GetEmailAddress=result
  Exit Function
Else
'If person document is found, return content of MailAddress field
'We use our reusable function for getting field's content
  GetEmailAddress=GetPersonField("MailAddress")
End If
End Function

Public Function GetPhoneNumber(personname As String) As String
  If nabdoc Is Nothing Then
    result=GetPersonDocument(personname)
  End If
  If result<>"OK" Then
    GetPhoneNumber="ERROR: "+result
    Exit Function
  Else
    GetPhoneNumber=GetPersonField("OfficePhoneNumber")
  End If
End Function
End Class

'Functions used by Web service to locate documents in Domino Directory:
Private Function GetPersonDocument(personname As String) As String
  Set nabdb=New NotesDatabase("", "names.nsf")
  'Set nabdb=session.CurrentDatabase
' Check that database exists and was opened
  If Not (nabdb.IsOpen) Then
    GetPersonDocument= "Error opening database"
    Exit Function
  End If
'Check that view exists in the database
  Set nabview = nabdb.GetView("($Users)")
  If nabview Is Nothing Then
    GetPersonDocument = "Error in search"
    Exit Function
  End If
'Get a document by provided search key
  Set nabdoc = nabview.GetDocumentByKey(personname, True)
  If nabdoc Is Nothing Then
    GetPersonDocument = "Cannot find person"
    Exit Function
  End If
  GetPersonDocument="OK"
End Function

Private Function GetPersonField(FieldName As String)
'Get item object
  Set item=nabdoc.GetFirstItem(FieldName)
'Sometimes item does not exist in the document, it's not an error
'so we return an empty string.
  If item Is Nothing Then
    GetPersonField=""
    Exit Function
  Else
'Return first value of the item
    GetPersonField=Cstr(item.Values(0))
  End If
End Function

```

As you can see, it's possible to have several functions (`GetEmailAddress` and `GetPhoneNumber`) inside the same Web service. Only those functions and methods declared as `Public` are exposed to clients. Private functions are not exposed for direct invocation.

Note:The Web services engine that interprets LotusScript changes all exposed functions and variable names to uppercase letters. For example, `GetEmailAddress` changes to `GETEMAILADDRESS` when Lotus Domino 7 auto-generates the WSDL. This change is important to know when you're creating consuming clients for such a Web service.

A simple Web service in Java code

Now that you've created a LotusScript Web service, you might be wondering if it's as easy to create a Web service using Java code. It is: Follow the steps in the previous panel to create a new Web service, but this time choose Java as the programming language rather than LotusScript. The code logic is similar to LotusScript (see Listing 4).

Listing 4. Code for the simple Java Web service

```
import lotus.domino.*;
import lotus.domino.types.*;

//Here comes our main class containing all exposed functions
public class GetPersonInfo {
//Declare global variables which you want to be accessible from all parts
//of this Web service. Set session object.
    private Session session = WebServiceBase.getCurrentSession();
    private Database nabdb = null;
    private View nabview = null;
    private Document nabdoc=null;
    private Item item = null;
//Declare function getEmailAddress and set it to accept 1 parameter.
//Set output type to data type String.
    public String getEmailAddress(String personname){
        String result="";
        String fieldValue="";
        try{
// If person's document is not already found in previous actions, find it now
            if (nabdoc==null){
//Locate a document using our search function
                result=getPersonDocument(personname) ;
            }
            if (!result.equals("OK")){
//If no person document is found, return the error message
//we got from the getPersonDocument function
                fieldValue= "ERROR: "+result;
            }else{
//If person document is found, return content of MailAddress field
//We use our GetPersonField reusable function for getting field's content
                fieldValue= getPersonField("MailAddress");
            }
        } catch(Exception e) {
            e.printStackTrace();
        }
//return result back to client
        return fieldValue;
    }

//These 2 functions are similar to LotusScript functions in the previous example
    private String getPersonDocument(String personname){
        try{
```

```

//AgentContext agentContext = session.getAgentContext();
//nabdb=agentContext.getCurrentDatabase();
nabdb=session.getDatabase("", "names.nsf");
// Check that database exists and was opened
if (!nabdb.isOpen()){
    return "Error opening database";
}
// Check that view exists in the database
nabview = nabdb.getView("($Users)");
if (nabview==null){
    return "Error in search";
}
// Get a document using name supplied as parameter to this function
nabdoc = nabview.getDocumentByKey(personname, true);
if(nabdoc==null){
    return "Cannot find person";
}
} catch(Exception e) {
    e.printStackTrace();
}
}
return "OK";
}
private String getPersonField(String FieldName){
    try{
// Get item object. Global object nabdoc was set earlier in
// getPersonDocument function.
        item=nabdoc.getFirstItem(FieldName);
//Sometimes item does not exist in the document, it's not an error so
// we return an empty string.
        if (item==null){
            return "";
        }else{
//Return the value that the item holds
            return item.getValueString();
        }
    } catch(Exception e) {
        e.printStackTrace();
    }
    return "";
}
}
}

```

A complex Web service in LotusScript

In this example, you create a LotusScript Web service that returns three values: `PersonName`, `EmailAddress`, and `PhoneNumber`. LotusScript functions can't explicitly return an array data type, so you must create a class containing three values, then return this class instead of an array. The functions `GetPersonDocument` and `GetPersonField` are the same as those used in the previous simple LotusScript Web service. Follow the steps described in the [A simple Web service in LotusScript](#) panel for creating a new complex Web service. Listing 5 shows the LotusScript code you need.

Listing 5. Code for the complex LotusScript Web service

```

Dim session As NotesSession
Dim nabdb As NotesDatabase

```

```
Dim nabdoc As NotesDocument
Dim nabview As NotesView
Dim item As NotesItem

'Create a LotusScript class holding 3 fields/values.
'This class will be returned back to the client.
Class MultiFieldArray
    Public PersonName As String
    Public EmailAddress As String
    Public OfficePhoneNumber As String
End Class

'This is our Web service's main class.
'It contains all exposed functions and methods.
Class GetPersonInfo
    Sub New
        Set session = New NotesSession
    End Sub
    'getEmailAndPhone accepts name of a person as argument to function.
    'We set output type to be MultiFieldArray class.
    Function GetEmailAndPhone(personname As String) As MultiFieldArray
    'Initialize a new instance of MultiFieldArray class
        Set GetEmailAndPhone = New MultiFieldArray
        If nabdoc Is Nothing Then
    'Locate a document using our search function
            result=GetPersonDocument(personname)
        End If
        If result<>"OK" Then
            Exit Function
        End If
    'Set MultiFieldArray fields to appropriate field values from document
        GetEmailAndPhone.PersonName=GetPersonField("FirstName")+_
            "+GetPersonField("LastName")
        GetEmailAndPhone.OfficePhoneNumber=GetPersonField("OfficePhoneNumber")
        GetEmailAndPhone.EmailAddress=GetPersonField("MailAddress")
    End Function
End Class
```

Figure 8 shows an example of output generated from this Web service.

Figure 8. Output from the Get Mail and Phone Web service



A complex Web service in Java code

Now, I make things a bit more difficult. In this Java example, you create a Web service that returns multiple fields for multiple people. This example allows you to conduct a "pattern" search -- for example, for all users who have "James" as their first name. You can modify this example to perform any type of database search. Listing 6 shows the Java code you need.

This Web service returns an array of complex type. As I mentioned earlier, returning a native array without code modifications is possible in the Java programming language but not in LotusScript. In the corresponding LotusScript example available in the attached database, I emulate array behavior by encapsulating complex classes inside each other.

Note:The `GetPersonDocument` and `GetPersonField` functions used in this Web service are same as in the previous simple Java Web service.

Listing 6. Code for the complex Java Web service

```
//Create a class (complex type) which we will return in sequence
// of values (array).
//To create a new class in domino designer use "New Class" button in
//Designer action buttons pane
public class MultiFieldArray{
    public String personName;
    public String emailAddress;
    public String officePhoneNumber;
}

import lotus.domino.*;
import lotus.domino.types.*;
//Our main class
public class GetPersonInfo {
    private Session session = WebServiceBase.getCurrentSession();
    private Database nabdb = null;
    private View nabview = null;
    private Document nabdoc=null;
    private Item item = null;

    //Declare getAllPersonsInfoByName function which returns an array
    //of MultiFieldArray type
    public MultiFieldArray[] getAllPersonsInfoByName(String personname){
        String result="";
        int x =0;
        //Initialize variable mfa for holding an array of field values.
        //Set initial size to 0 elements.
        MultiFieldArray[] mfa = new MultiFieldArray[0];
        DocumentCollection coll;
        try{
            if (nabdoc==null){
                //Use GetPersonDocument function to set database object
                result=getPersonDocument(personname) ;
            }
            if (!result.equals("OK")){
                return mfa;
            }
        }
        //Search the database for documents where First Name field matches received parameter
        coll=nabdb.search("Form=\"Person\" & FirstName=\""+personname+"\"", null, 0);
        //set size of our array to number of documents in search
        mfa = new MultiFieldArray[coll.getCount()];
        //Get first document in collection and continue with processing all other documents
        nabdoc=coll.getFirstDocument();
    }
}
```

```
while (nabdoc != null) {
//Attach a new instance of MultiFieldArray class to our array of complex types
mfa[x]= new MultiFieldArray();
//Set MultiFieldArray fields to appropriate field values from the document
mfa[x].personName=getPersonField("FirstName")+ " "+getPersonField("LastName");
mfa[x].officePhoneNumber=getPersonField("OfficePhoneNumber");
mfa[x].emailAddress=getPersonField("MailAddress");
nabdoc = coll.getNextDocument();
x++;
}
} catch(Exception e) {
e.printStackTrace();
}
//return the whole array back to requesting client
return mfa;
}
```

Figure 9 shows an example of output generated from this Web service.

Figure 9. Output from the Get multiple Mail and Phone Web service



A complex Java Web service for sending e-mail from Lotus Domino

In the previous complex Java example, you saw how to output a complex type to a requesting client. Now, you will see how to accept a complex type as an incoming parameter to a function in a Web service (see Listing 7). The purpose of this Web service is to construct and send an e-mail message based on parameters placed inside the incoming `MailMessage` class. Basically, the service simulates the `@MailSend` function available in Formula Language. If you call this Web service from provided Java and LotusScript client examples, make sure to specify a valid e-mail address as first parameter to the function.

Listing 7. Code to create an e-mail-sending Java Web service

```
//Create a new class which has 5 fields used for sending mail
//As in previous example we create this class using "New Class" action button.
public class MailMessage{
    public String sendTo;
    public String copyTo;
    public String bcc;
    public String subject;
    public String body;
}

import lotus.domino.*;
import lotus.domino.types.*;
//Our main class
public class ComplexMailer {
    //Function which accepts 1 complex argument of MailMessage type
    public String mailSend(MailMessage mm){
        Session session = WebServiceBase.getCurrentSession();
        try{
            AgentContext agentContext = session.getAgentContext();
            Database db=agentContext.getCurrentDatabase();
            //For debugging purposes print out message to console to show incoming arguments
            System.out.println(mm.sendTo+" "+mm.copyTo+" "+mm.bcc+" "+mm.subject+" "+mm.body);
            // Begin creating a new mail message
            Document doc = db.createDocument();
            doc.appendItemValue("Form","Memo");
            //mm.sendTo returns sendTo part of the class supplied as function argument
            doc.appendItemValue("SendTo",mm.sendTo);
            doc.appendItemValue("CopyTo", mm.copyTo);
            doc.appendItemValue("BlindCopyTo", mm.bcc);
            doc.appendItemValue("Subject", mm.subject);
            //Create a richtext field for holding the Body field of the message
            RichTextItem rti = doc.createRichTextItem("Body");
            rti.appendText(mm.body);
            //Send the mail document
            doc.send(false);
        } catch(Exception e) {
            e.printStackTrace();
        }
        //always return "OK",
        //we do not validate if the message has properly formatted field values
        return "OK";
    }
}
```

Section 5. Test the new Web service

Testing options

You can test your Lotus Domino 7 Web services using either a Domino Designer 7 client or a custom application created in your favorite programming language. Most if not all modern programming languages support invocation of Web services. You can use these examples in Visual Basic, VBScript, and Active Server Pages (ASP) with only minor modifications. Java examples were tested in Java 2 Standard Edition (J2SE) 1.3.1 and J2SE 1.4.2. The advantage of testing using LotusScript code in a Domino Designer 7 client is that you don't need to install any other programs on your computer and you can reuse the examples to seamlessly integrate other available Web services as part of your Lotus Domino applications.

For the LotusScript client example, I use the Microsoft SOAP (MSSOAP) component; for the Java example, I use the Web Services Invocation Framework (WSIF). Common between them is the fact that they are generic implementations allowing you to call a wide range of Web services without making modifications to the code. WSDLs of invoked services might differ, but the client code remains much the same. The whole point of WSIF and MSSOAP is to provide access to services exposed via diverse protocols through a uniform WSDL-based API. They automatically parse WSDL descriptions and generate correctly formed requests to the Web service without any instructions from the programmer. This functionality dramatically speeds up the testing process, and you can typically begin using any Web service after just a few minutes of programming. You can even call WSIF from the command line without any programming at all.

Note:The Java source files are included in WSIFclient.zip; the LotusScript examples of MSSOAP are included in the Domino database and can be run from there. To download these files, go to the [Resources](#) section at the end of this tutorial.

There are two URL commands that help you call a Lotus Domino Web service:

- **? OpenWebService:**Your testing client uses this command in the background. In some implementations of Web services clients, you must manually include this command as the service location.
- **? WSDL:**This command shows the contents of the WSDL file. You use this command in WSDL-based clients for specifying the location of the Web service's WSDL definition -- for example,

<http://localhost/ws/WebservicesR7.nsf/SimpleJavaServiceNAB?wsdl>.

As the purpose of this tutorial is to show you how to create a Web service, the details of how the clients work is not included. If you're interested, you can look at the source code of agents and applications. Both Java and LotusScript code are documented and are rather self-explanatory.

To run the WSIF example, you need to have the necessary Java archive files in the `classpathenvironment` variable on your computer. You get those files when you download the WSIF distribution from the Apache Web site. A batch file that helps you start the application is included in the attached .zip file available from the [Resources](#) section.

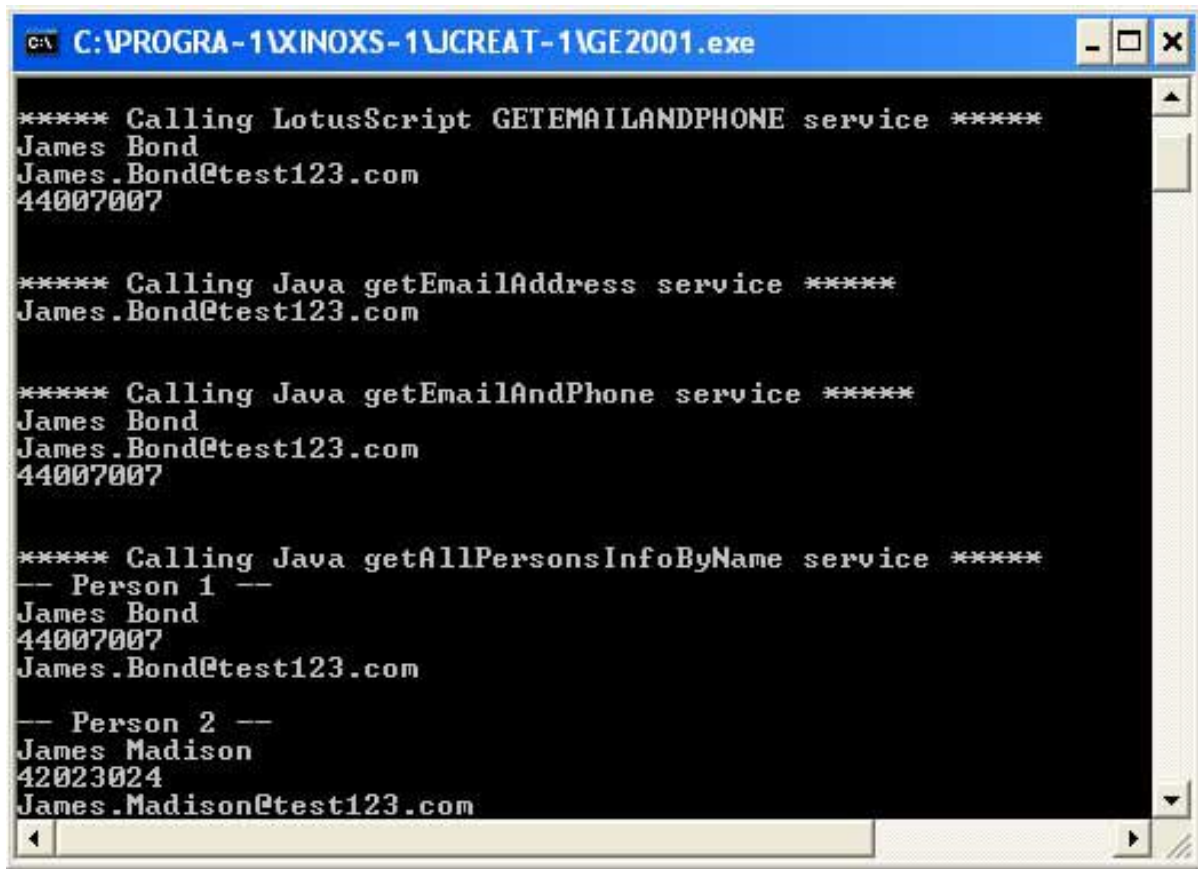
To run the LotusScript client example, you need to install the MSSOAP component on your computer (if you haven't already done so). You can download the component from the Microsoft Web site using the link in the [Resources](#) section. Be aware that the Microsoft SOAP Toolkit is deprecated by the Microsoft .NET Framework. Mainstream SOAP Toolkit support will be retired on March 31, 2005, with extended support lasting until March 31, 2008.

Java WSIF client example

WSIF is a Java implementation of a Web services client. I use WSIF as a Java application, not as a Java Domino agent. With some work, you can adjust the code to be run as a Domino agent. In this case, you can use the `\jvm\lib\ext` folder of the Domino Designer client to store all needed files Java archives or the `JavaUserClasses` notes.ini parameter to point to needed archives.

You can compile and run the code in Eclipse, IBM WebSphere® Studio Application Developer, the Java SDK, or another Java development tool. Include the following four files in your project: `WSIF.java`, `DynamicInvoker.java`, `MultiFieldArray.java`, and `MultiFieldArrayLS.java`. `WSIFclient.java` contains code for testing all the Web services discussed in this tutorial. Run the compiled application; the results appear in the console window. Figure 10 shows the successful output.

Figure 10. Java console window showing successful output



```
C:\PROGRA-1\XINOXS-1\JCREAT-1\GE2001.exe

***** Calling LotusScript GETEMAILANDPHONE service *****
James Bond
James.Bond@test123.com
44007007

***** Calling Java getEmailAddress service *****
James.Bond@test123.com

***** Calling Java getEmailAndPhone service *****
James Bond
James.Bond@test123.com
44007007

***** Calling Java getAllPersonsInfoByName service *****
-- Person 1 --
James Bond
44007007
James.Bond@test123.com

-- Person 2 --
James Madison
42023024
James.Madison@test123.com
```

I created the `MultiFieldArray` class for this Java client by clicking **Import WSDL** in the Domino Designer 7 Web service design element. This functionality works much like the `WSDL2Java` tool included in the Apache Axis toolkit. You point to the WSDL location, and the tool automatically generates a matching Java structure. (Domino Designer 7 offers the same feature for LotusScript Web services.) This functionality simplifies considerably the process of creating matching complex data types and service interfaces that Web services use.

LotusScript client example

The Domino database included in this tutorial contains five agents that you can use for testing your Lotus Domino 7 Web services. Those agents are:

- LS client test simple LS Web service
- LS client test simple Java Web service
- LS client test complex LS Web service
- LS client test complex Java Web service

- LS client test mail LS Web service

The LS client test simple LS Web service and LS client test simple Java Web service agents perform the call to the simple Web services you created earlier. These services return a single value -- for example, an e-mail address or phone number. The LS client test complex LS Web service and LS client test complex Java Web service agents perform two operations:

- They get a complex data type containing three values (PersonName, EmailAddress, and PhoneNumber). Refer back to Figure 8 in the [A complex Web service in LotusScript](#) panel for an example of a message box from this agent.
- They get a complex data type containing an array of complex values. Every element in this array represents a matching person in the Domino Directory database. These people were found by performing a database search. Every person entry has three fields: PersonName, EmailAddress, and PhoneNumber. Refer back to Figure 9 in the [A complex Web service in Java code](#) panel for an example of a message box from this agent. The agent uses the ComplexLSServiceNAB Web service for querying.

The *LS client test mail LS Web service* agent performs a call to the simple mailing Web service. This service accepts five simple text parameters and returns a single value. The first parameter is the recipient's e-mail address, which you must change from this test value to a valid address.

Here's a simple example of MССOAP usage in the LotusScript implementation of a consuming client. If you placed the example database in a location other than data/ws/WebServicesR7.nsf, you must change the URL to the actual location of the WSDL description before running the example code shown in Listing 8.

Listing 8. Example for MССOAP usage in LotusScript

```
Sub GetEmail(personname As String)
'Provide location of WSDL file.
'localhost address is used if service is started in Domino Designer
'change this URL to the location of the database containing Web service
  sWSDL = "http://localhost/ws/WebServicesR7.nsf/SimpleLSServiceNAB?wsdl"
'Create a new MССOAP object
  Set Client =CreateObject("MССOAP.SoapClient")
'Initialize connection to WSDL file and get WSDL structure
  Call Client.mssoapinit (sWSDL)
'Call a method or function provided by Web service
'You get an error if you call non-existing function or supply wrong parameter type
  result = Client.GetEmailAddress(personname)
'output result to message box
  Messagebox result, MB_OK, "Get Email"
End Sub
```

Debugging your client and Web service

To help find errors in XML-based requests and responses, the Apache Group distributes a nice application as part of its open source Apache SOAP project. It's a TCP Tunnel/Monitor that you can use to monitor and debug your SOAP interfaces.

Because XML doesn't use a binary format, it is human readable and you can see the information exchange between the client and the service in plain text. You can observe possible errors in XML structure or in your coding and make appropriate changes. You can also call the application from the command line:

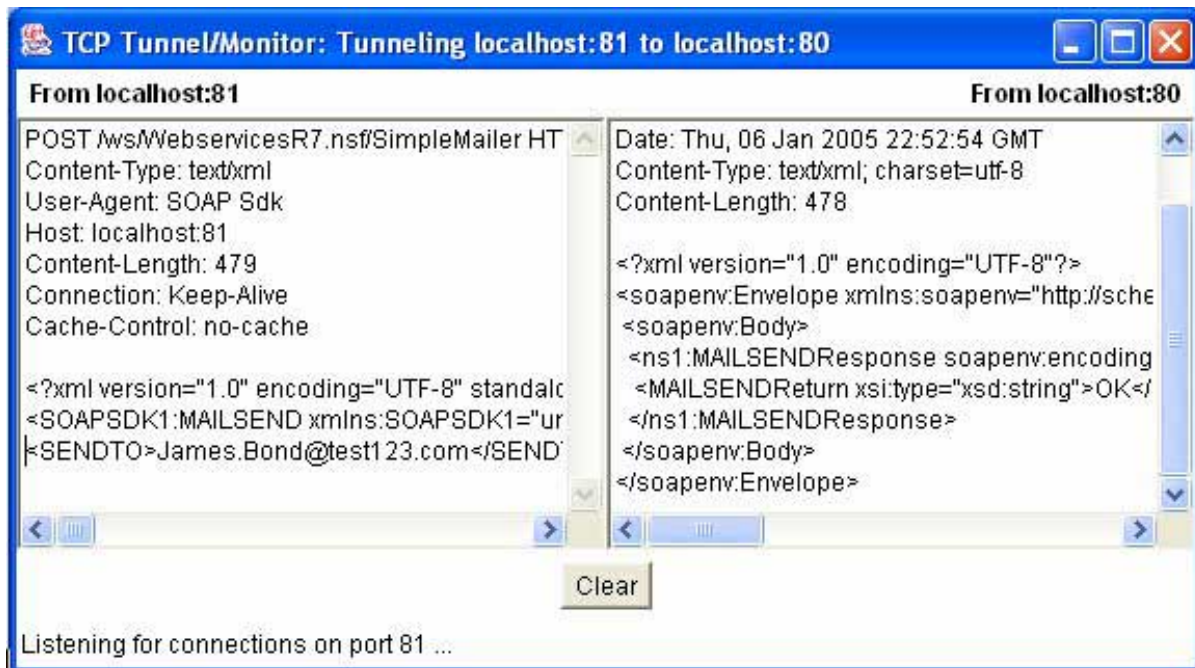
```
java -cp soap.jar org.apache.soap.util.net.TcpTunnelGui 81 www.server.com 80
```

This command redirects any requests at localhost port 81 to server port 80. Note that `TcpTunnelGui` accepts three start parameters:

- The TCP port on which the tool will listen (Redirect your client to this port -- in this case, port 81 -- rather than the normal port by specifying **http://localhost:81** instead of **http://www.server.com:80** as the Web service location.)
- The target address (IP-number or DNS name), which can also be localhost
- The target port (in this case, port 80)

The TCP Tunnel/Monitor shows communication between the client and the remote Web service (see Figure 11). On the left side of the TCP Tunnel/Monitor, you can see the outgoing SOAP request. On the right side, you can see the incoming SOAP response.

Figure 11. The TCP Tunnel/Monitor



Section 6. Wrap up

Summary

Domino Designer 7 is at this date probably the most developer-friendly software for rapid creation and deployment of Web services. As soon as you save your new Web service, it's ready for others to use. You can create a simple Web service in just a couple of minutes. The examples in this tutorial show how easy it is.

As IBM continues to successfully implement the Lotus Domino Web services strategy, it's exciting to explore new features integrated into Lotus Domino 7. The sample database provides a scalable framework that you can use with your own databases to extend Lotus Domino's collaborative features to other applications.

Resources

Learn

- [developerWorks Domino page](#) contains a wealth of information for technical developers, such as; technical content, downloads, betas, and support.
- Check out the [Web Services Invocation Framework](#) for dynamic invocation of Web services from Java applications.
- [Java 2 Platform, Standard Edition, v 1.4.2](#) provides a complete environment for application development on desktops and servers.
- Check out the [MSSOAP toolkit](#) for calling Web services from LotusScript agents and Visual Basic applications.
- Refer to the IBM developerWorks tutorial [Building Web services using Lotus Domino 6.5.1](#) for information about building pre-Lotus Domino 7 Web services.
- Refer to the IBM developerWorks article [Which style of WSDL should I use?](#) for information about usage of different WSDL binding styles.
- The IBM developerWorks article [Lotus Notes/Domino 7 Web services](#) provides more information about Web services in Domino 7.
- [WebSphere Studio Application Developer](#) is a complete development framework for creating Java applications.
- Learn more about Web services from the developerWorks [New to Web services page](#).

Get products and technologies

- Download the [Domino 7 database](#), which contains all the source code and sample data for the Web services and agents described in this tutorial.
- Download the [Java WSIF client](#) and the TCP Tunnel/Monitor described in this tutorial for testing the Web services provided in the sample database.
- Download [Domino and Notes 7](#).
- Download [Apache AXIS](#).
- Download [Apache SOAP](#).

Discuss

- Join a live [IBM Software Development Platform Webcast](#) and participate in the Q&A session or replay the recorded Webcasts at your convenience.

About the author

Andrei Kouvchinnikov

Andrei Kouvchinnikov is a certified Principal Domino Developer and Administrator. Andrei works on Mindspot where he is a consultant and developer in Domino and Collaboration areas. His experience includes full life cycle development of Lotus Domino applications running on multiple platforms and development of applications for Quickplace and Sametime. He has been working with the Lotus Domino platform since version 4.5 for OS2®.