Lotus. software

IBM

Redbooks Paper

Victoria Amor
Peter Kovari

Lotus Domino and .NET coexistence

Lotus Domino and Microsoft .NET technologies can be integrated using Web Services. Web Services are self-contained, self-describing, modular applications that can be published to and invoked from the Web. Unlike traditional Web-based applications, Web Services contain no user interface, which means that these applications could be remotely invoked by other applications known as *clients*. Web Services use technology standards such as XML, SOAP, WSDL, and UDDI. In addition to this, Web Services applications communicate with each other by using the HTTP protocol and SOAP messages.

Lotus Domino is ideal software for using Web Services to extend the collaborative features of Notes/Domino across the enterprise. Domino Application Server can either host or use Web Services. As we have also discussed in this redbook, Microsoft .NET platform is a collection of tools from Microsoft that let you both use and host Web Services.

Web Services integration

It is not the purpose of this paper to introduce Web Services concepts and their architecture (you can find detailed information about these topics in the redbook *WebSphere Version 5 Web Services Handbook*, SG24-689). This paper shows how Web Services, developed on different languages and different tools, can interact with each other.

The principal elements involved on the interoperability of Web Services are the following:

- A service provider
- A service requester
- A way to code the data - XML language
- A way to define and describe the service - WSDL document
- A way to format remote calls -SOAP Protocol
- A network protocol - HTTP

Also in a runtime environment, we need:

- A Service Broker known also as a Service Registry.
- A way to publish and find services - UDDI

The Web Services model includes three roles: the service provider, the service broker, and the service requester and the methods and properties are associated with the service. The Web Service is published in an external or internal registry using UDDI. Once the Web Service is publicly or privately available in the appropriate UDDI registry, the service requester uses UDDI to find the Web Service and consume it. SOAP is used to invoke a Web Service, therefore binding the service requester to the service provider.

As mentioned before, Lotus Domino Application Server and Microsoft .NET platform can host or provide Web Services. This means that it is possible to use Lotus Domino Server to host Web Services as a service provider, because Domino Applications can be modified to provide SOAP Interfaces and WSDL Descriptions using XML, and use .NET clients to invoke the Lotus Domino Service as a consumer or the other way around, using Lotus Domino as a consumer (Service Requester), which involves calling or invoking the .NET service and getting the response back.

Therefore, the purpose of this section is to explain the following scenarios:

- Lotus Domino as a provider and .NET as a consumer.
- .NET as a provider and Lotus Domino as a consumer.

Domino provider, .NET consumer

As explained above, Lotus Domino applications can have a Web Service interface that allows it to be accessed as a Web Service by remote users or by Web server clients. The design elements needed to provide a Domino Web Service are the following:

- A *Lotus Script Web Agent*: this agent is written to accept a SOAP request, parse it, call the requested method (function), and return the result as a SOAP response to the requester.
- Any standard LotusScript function stored in a *Lotus Script library*.
- A *page containing a WSDL definition of the service*. This is required only because the database is going to be accessible by the .NET consumer.

The application components are represented in the following diagram.

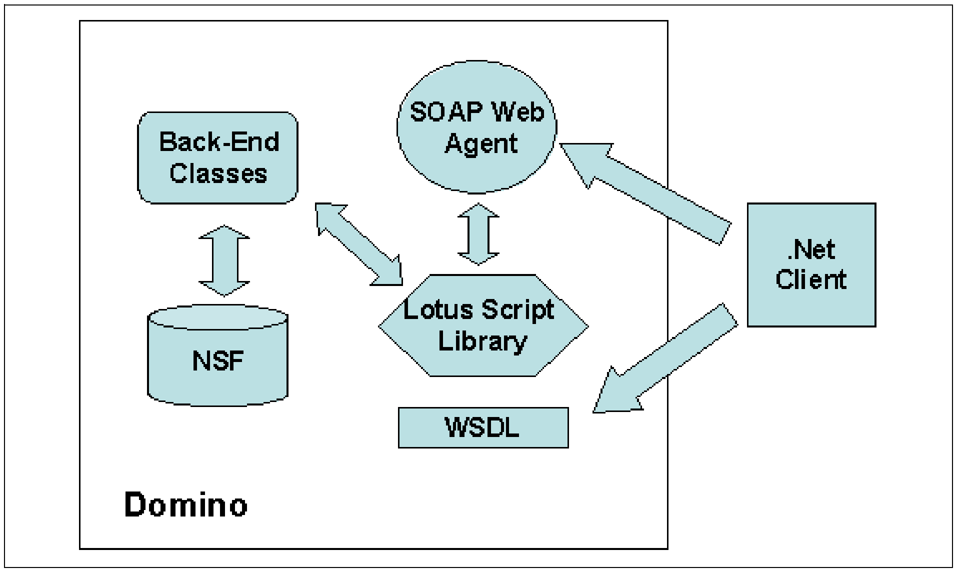


Figure 1 Domino - .NET client interaction

To explain how to use Domino as a Web Service Provider, we have created a Notes Sample Web Services .NET Database Application (WebServiceNet.nsf). This database shows how you can easily write a 100% Lotus Script Web Service that allows .NET clients to access the details of a particular upcoming ITSO Residency by giving a Residency Code.

Note: The example was created based on the Building Web Services using Lotus Domino 6 Tutorial which was developed by IBMdeveloperWorks and can be located in the following URL: `https://www6.software.ibm.com/reg/devworks/dw-lsdom6ws-i?S_TACT=103AMW13&S_CMP=LDD`. More information regarding developing Web Services in Domino can be found in Lotus Domino Designer® 6 Help database.

For the development, we used the following product versions:

- Microsoft .NET Framework Software Development kit V1.1
- Microsoft Windows 2000 Server with Service Pack 4
- Lotus Domino Server V6.0.2 CF2
- Microsoft IE V5.5 or newer
- Lotus Domino Administration Client V6.0.2.
- Lotus Domino Designer Client V6.0.2.

For more details on how to install the software products, refer to the installation manuals. Lotus Domino Administrator client and Lotus Domino Designer was installed in another machine in order to administrate the Domino Server and to design the Sample application.

Before starting, make sure that you have TCP/IP network configured (it is recommended to have a fixed IP address), that it is possible to resolve the machine host name (via Host file or DNS) and also that you have Domino Server configured. For our example, the following Domino nomenclature was selected within the configuration process, but of course it is possible to use another one.

Table 1 Domino nomenclature

Concepts	Selected Name
Domino Domain Name	TEST
Organization Name	TEST
Server Name	Domin6/TEST
Server Title	Test Server
Notes Network Name	TCPIP Network
Notes Administrator Name	Notes Admin/TEST

Note: For more details about configuring a Domino Server, refer to the Lotus Domino Administrator 6 help database.

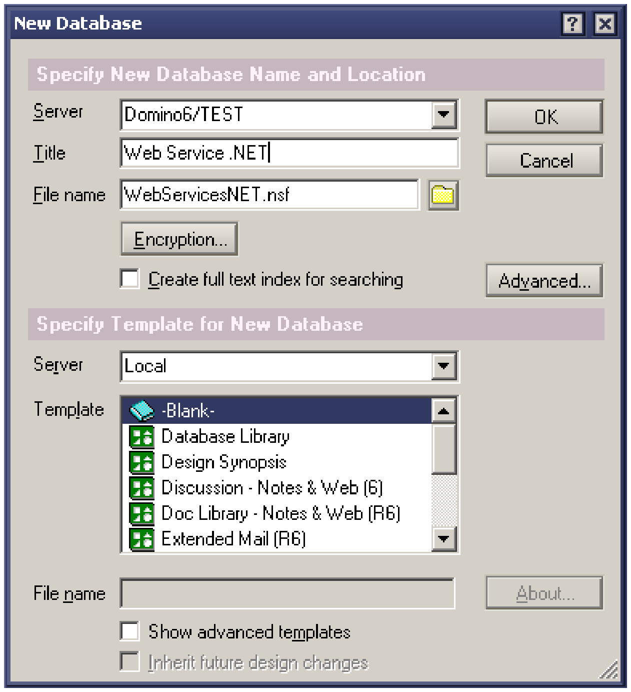
Once you have installed and configured the products to begin with the example, follow these steps:

1. Create a new database.
2. Create the Forms and Views for the database.
3. Create a Lotus Script Web Agent (ResidencyWS).
4. Create a Lotus Script Library (Domino).
5. Create a WSDL page to describe the Domino Web Services.
6. Create a .NET client.
7. Test the application.

Creating a new database

Although included in this redbook is the sample database inside the additional material, we are going to start from scratch by creating a new database in our server.

Open Lotus Domino Designer 6 and select **File -> Database -> New** from the menu. This opens the New Database dialog box as shown below.


New Database
Specify New Database Name and Location
OK
Cancel
Encryption...
Advanced...
Specify Template for New Database
About...

Server Domino6/TEST
Title Web Service .NET
File name WebServicesNET.nsf
Create full text index for searching
Server Local
Template -Blank-
Database Library
Design Synopsis
Discussion - Notes & Web (6)
Doc Library - Notes & Web (R6)
Extended Mail (R6)
File name
Show advanced templates
Inherit future design changes

Figure 2   Create database

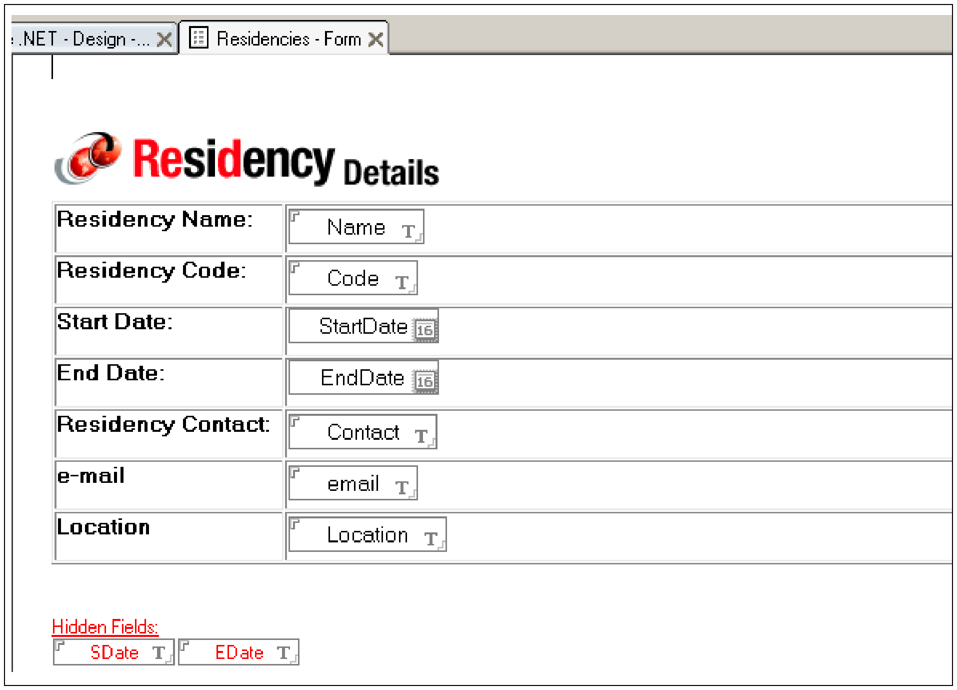
In the Server field, select: **<The name of the Domino Server>** (in our case **Domino6/TEST**), as Database Title type: `Web Service .NET` and for File name: `WebServiceNET.nsf`. Select **-Blank-** as the template. Click the **OK** button.

Creating the Forms and Views for the database

When a new database is created from scratch, it does not contain design elements except one default view, so it is necessary to create all of the elements needed for the application. Because the sample database is going to contain all the details for the upcoming ITSO residencies, the minimum design elements to create are:

1. A new Form to provide the structure for creating and displaying ITSO Residency documents details.

   Open Lotus Domino Designer 6 and select **Forms** in the Design pane, then click **New Form**; an untitled blank form is displayed. Add the necessary fields for display the residency information details such as: *Residency Name, Residency Code, Start date, End Date, Residency Contact, e-mail and Location*. Save the form with the `Residencies` name. The New Form could look like the figure below.



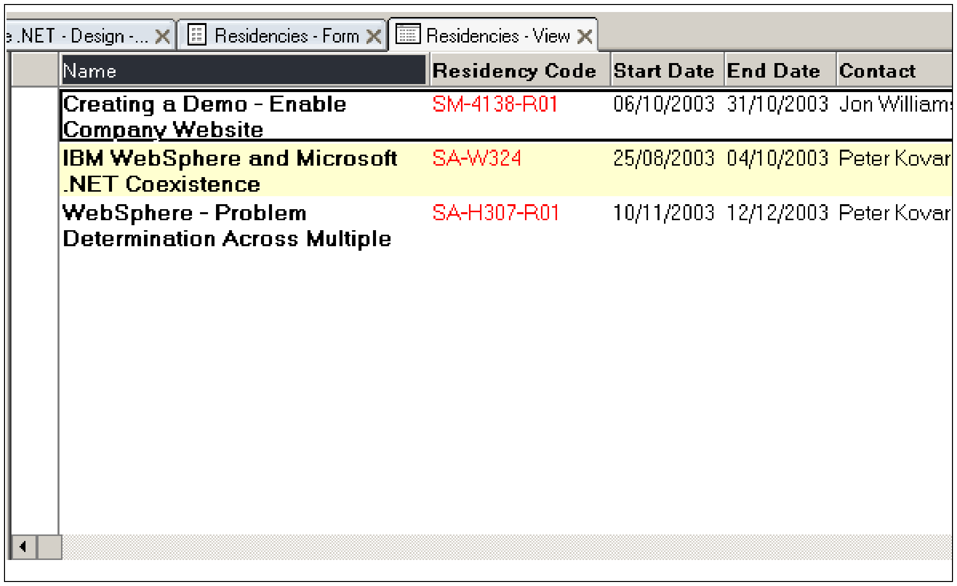
Residency Name:	Name
Residency Code:	Code
Start Date:	StartDate
End Date:	EndDate
Residency Contact:	Contact
e-mail	email
Location	Location

SDate EDate

Figure 3 Create Residencies Form

2. A new View for access to the list of Residency documents is created in the database.

   Open Lotus Domino Designer 6 and select **Views** in the Design pane, and then click **New View** or modify the default view initially created. Add the following columns: Name, Residency Code, Start Date, Contact, email and Location. Save the View with the name `Residencies`. The new view could look like the figure below.



Name	Residency Code	Start Date	End Date	Contact
Creating a Demo - Enable Company Website	SM-4138-R01	06/10/2003	31/10/2003	Jon William
IBM WebSphere and Microsoft .NET Coexistence	SA-W324	25/08/2003	04/10/2003	Peter Kovar
WebSphere - Problem Determination Across Multiple	SA-H307-R01	10/11/2003	12/12/2003	Peter Kovar

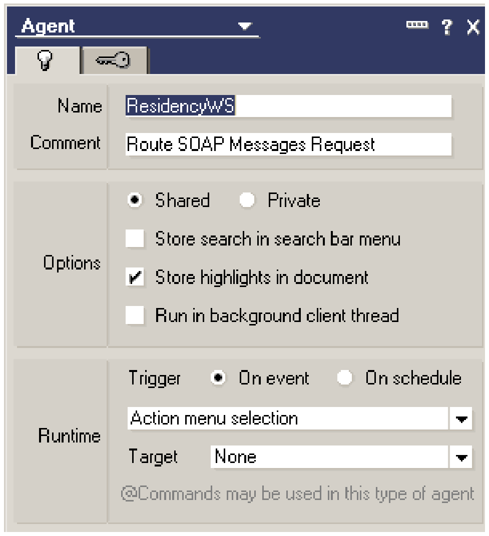
Figure 4 Residencies View

3. Also, create a hidden View called `(By Code)` with the first column ordered by Residency Code. This hidden view is where the Domino Web Service is going to access for locating the residency.

Creating a Lotus Script Web Agent

After the database is created, to route the SOAP Messages Request to the appropriate function inside the Lotus Script Library, a Lotus Script Web Agent is needed. Open the Lotus Domino Designer.

1. In the left pane, select **Shared Code -> Agents**. Click the **New Agent** action button. The Agent properties dialog box is displayed.


Agent

Name: ResidencyWS

Comment: Route SOAP Messages Request

Options: [✓ Shared] Private

- [ ] Store search in search bar menu
- [x] Store highlights in document
- [ ] Run in background client thread

Runtime: Trigger [✓ On event] On schedule

Action menu selection

Target: None

@Commands may be used in this type of agent

Figure 5 Web agent

Give `ResidencyWS` as the Domino Web Agent Name and select **Shared** as the Agent Option. Set the agent trigger to the `On event` and also set `Action menu selection`. The last thing to do is to set the Agent target to `None` which means that the agent is going to work on fields of the current document such as those launched from WebQueryOpen or WebQueryClose, or like a form action or hotspot that also works on fields in the current document.

First of all, declare the incoming and response SOAP Messages variables as string.

```
'Declare the response as String
Dim response As String
'Declare the incoming SOAP Message as String
Dim SOAPin As String
```

2. The next step is to create a NotesSession object by declaring the variable session and setting it as `New` to create a new instance for that object. Then, initialize the object variable *doc* using the *DocumentContext* property of the NotesSession class. The agent can use this property to access the in-memory document. The next line of code sets the SOAPin variable equal to the content of the "Request_content" field using the *GetItemValue* method of the NotesDocument class. This is where the SOAP message resides as a result of a "Post," in the DocumentContext object.

```
Dim session As New NotesSession
Set doc = session.DocumentContext
SOAPin = doc.GetItemValue("Request_content")(0)
```

3. For debugging purposes, a Log Message Function was created, with the objective of writing a new document in the database with the incoming SOAP Message. For that purpose, a new Form and a new View were added to the database:

   Messages View: the view that displays all the SOAP incoming Message documents created.

   Message Form: used for creating a document with the SOAP incoming Message every time the Domino Web Services is accessed.

   After the message is logged, a `RemoveWhitespace` function is used to remove all spaces, tabs, and new line characters as shown below:

   ```
   LogMessage(SOAPin)
   SOAPin = RemoveWhitespace(Fulltrim(SOAPin))
   ```

   Note: For more details about the `Log Message` and `RemoveWhitespace` functions created only for debugging purposes, refer to the sample database included in the additional material.

4. A SOAP Message consists of the following parts: a SOAP Envelope which marks the beginning and end of a message and a SOAP Body inside the SOAP Envelope which includes the method signature to be executed and the method arguments. The next piece of the code manually parses the SOAP message using the Lotus Script Language string handle functions (Instr and Mid) to extract the following items.

Table 2 SOAP Message table

Item	Variable Stored In	Definition
namespace	NameSpace	Script Library to load.
method	MethodName	Function to execute in the script library.
argument	argValue	Parameter to pass to function as the aString variable.

   Later, for parsing the SOAP content more efficiently, we will use the new NotesDOMParser object. Note also the first line (`On Error Goto ErrHandle`), which determines how an error will be handled in this case.

   ```
   On Error Goto Errhandle
   bodyPos= Instr(1,SOAPin,|<soap:Body
   soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">|)+74
   'parse out method signature
   methodSigPos= Instr(bodyPos,SOAPin,|<|)+1
   methodSigEnd=Instr(bodyPos,SOAPin,| |)
   methodSignature = Mid(SOAPin,methodSigPos,(methodSigEnd-methodSigPos))
   ```

```
'parse out method name
methodPos= Instr(bodyPos,SOAPin,|:|)+1
methodEnd=Instr(methodPos,SOAPin,| |)
methodName = Mid(SOAPin,methodPos,(methodEnd-methodPos))
'parse out namespace
nameSpacePos= Instr(methodEnd,SOAPin,|uri:|)+4
nameSpaceEnd=Instr(nameSpacePos,SOAPin,|"|)
nameSpace=Mid(SOAPin,nameSpacePos,(nameSpaceEnd-nameSpacePos))
```

5. The next code slice maps the namespace, method, and argument from the SOAP request to the script library, function, and parameter (respectively) called by the Web agent, and captures the return value in the response variable.

   ```
   callString  = |Use | & |"| & nameSpace & |"| & |
   response  = | & methodName & |(SOAPin)|
   ```

   Execute the `callString` variable that makes the specified script library run.

   ```
   Execute callString
   ```

6. The next step is to build the SOAP response which includes the `response` and `MethodName` variables and store it in the `strTmp` variable.

   ```
   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>|
   ```

   By default, Domino translates Web agent content to HTML and, because the response is going to be populated in XML, data is necessary to specify that the content type of the agent is XML. Use the `Print` statement for that:

   ```
   Print "Content-Type: text/xml"
   ```

7. To send the SOAP response to the requester, use the following code:

   ```
   Print strTmp
   ```

   To terminate execution of the current block statement, use:

   ```
   Exit Sub
   ```

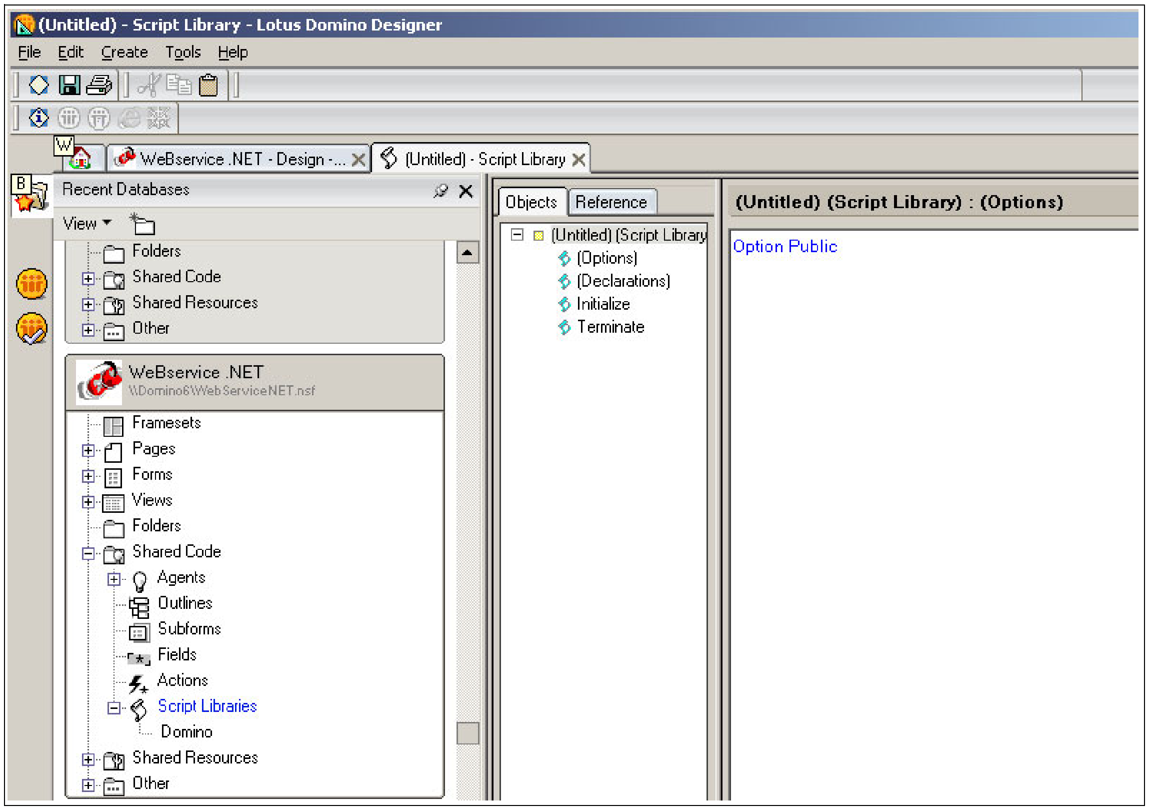
   The last step is to include the error-handling routine that begins at the label Errhandle defined previously:

```
Errhandle:
Exit Sub
```

In our case, we include `Exit Sub` to terminate the execution and do nothing but it is possible to include other Lotus Script code to handle the errors.

Creating a Lotus Script Library

As we defined in the SOAP incoming message skeleton, the method to be executed has to be inside a Lotus Script Library. To create a Lotus Script Library, open the database in Domino Designer 6 and select **Script Libraries**. Then click **New LotusScript Library**. This opens a new Script Library as shown in the figure.



```
Option Public
```

Figure 6 New Script library

1. Before sending the details regarding a new ITSO residency to the client, it is necessary to parse the content of the SOAP incoming Message using the NotesDOMParser class. So first of all, create a NotesDOMParser object by declaring the variable `domParser` in the Script Library (Declaration) section.

This class is new in Lotus Domino 6 and is used to process input XML into a standard DOM (Document Object Model) tree structure.

```
Dim domParser As NotesDOMParser
```

Note: For more information about the new NotesDOM classes in Domino 6, refer to the Lotus Domino Designer 6 Help. Also, more information about DOM Document Object Model Core is found at:

http://www.w3.org/TR/DOM-Level-3-Core/core.html#ID-1590626202

2. Create a new function with the ResCodeSearch name. This function will be the method specified in the SOAP request.

```
Function ResCodeSearch(msg As String) As String
```

3. The next step is to create a NotesSession object by declaring the variable session and setting it as `New` to create a new instance for that object. Initialize the object variables.

```
Dim session As New NotesSession
Dim rootElement As NotesDOMDocumentNode
Dim nodeList As NotesDOMNodeList
Dim node As NotesDOMNode
```

4. Initialize the object variable `domParser` using the `CreateDOMParser` method of the NotesSession class and use the `process` method to generate the DOM tree. Set the object variable `rootElement` using the *Document* property of the NotesDOMParser class to access the document node. Set the object variable `nodeList` using the `GetElementsByTagName` method of the NotesDOMDocumentNode class specifying * as the tag name. This will return a NotesDOMNodeList of all the NotesDOMElementNode objects with this given tag name. The list returned is arranged in the order in which they are encountered. The last step is to locate the value of the Residency Code inside the `<code>` tag of the in the node list and set it to the variable `Code`.

```
Set domParser = session.createDOMParser(msg)
domParser.process
Set rootElement = domParser.Document
Set nodeList = rootElement.GetElementsByTagName("*")
'locate Residency Code
For i=1 To nodeList.NumberOfEntries
Set node = nodeList.GetItem(i)
   If(node.NodeName="code")Then
      Code = node.FirstChild.NodeValue
      Exit For
   End If
Next
```

5. When the `Code` variable is retrieved, it is necessary to locate this code inside the database. For that, set the variable `db` with the property `CurrentDatabase`

of the *NotesSession* class. Then, set the object variable view using the `GetView` method, giving it the name of a view. After that, initialize the object variable `doc` using the `GetDocumentByKey` method, giving it the Residency Code located before and the `true` parameter because we want to find an exact match.

```
Dim db As NotesDatabase
Dim view As NotesView
Dim doc As NotesDocument
Set db = session.CurrentDatabase
Set view = db.getView("(By Code)")
'locate Residency Code in database
Set doc = view.GetDocumentByKey(Code,True)
```

6. The last step is to create a response containing XML data for the SOAP client, and return the result to the "ResidencyWS" agent which calls our function `ResCodeSearch`.

```
tmp =|<Code xsi:type="xsd:string">| & doc.Code(0) &|</Code>|&_
    |<Name xsi:type="xsd:string">| & doc.Name(0) &|</Name>|&_
    |<StartDate xsi:type="xsd:string">| & doc.SDate(0) &|</StartDate>|&_
    |<EndDate xsi:type="xsd:string">| & doc.EDate(0) &|</EndDate>|&_
    |<Contact xsi:type="xsd:string">| & doc.Contact(0) &|</Contact>|&_
    |<email xsi:type="xsd:string">| & doc.email(0) &|</email>|&_
    |<Location xsi:type="xsd:string">| & doc.Location(0) &|</Location>|

ResCodeSearch = tmp
```

7. Save the new Lotus Script Library as `Domino`.

Creating a WSDL page to describe the Domino Web Services

WSDL allows a service provider to specify the following characteristics of a Web Service:

- Name of the Web Service and addressing information.
- Protocol and encoding style to be used when accessing the public operation of the Web Service.
- Type information: operations, parameters, and data types comprising the interface of the Web Service, plus a name for this interface.

The anatomy of a WSDL document is as follows:

- **Types**: A container for data type definitions using some type system, such as XML schema.
- **Message**: An abstract, typed definition of the data being communicated. A message can have one or more typed parts.

- **Port type**: An abstract set of one or more operations supported by one or more ports. Each operation defines an input and an output message as well as an optional fault message.
- **Operation**: An abstract description of an action supported by the service.
- **Binding**: A concrete protocol and data format specification for a particular port type. The binding information contains the protocol name, the invocation style, a service ID, and the encoding for each operation.
- **Port**: A single endpoint, which is defined as an aggregation of a binding and a network address.
- **Service**: A collection of related ports.

WSDL specification uses XML syntax; therefore, there is an XML schema for it.

Note: For more information about WSDL, refer to *WebSphere Version 5 Web Services Handbook*, SG24-6891.

Because the Domino Web Service created before is going to be consumed by a .NET client, a WSDL file is required. This WSDL file will be included in a Lotus Domino Page. Create a new page in Domino; follow the steps below.

1. Open the database in Lotus Domino Designer and select **Pages** on the left pane and click **New Page**; an untitled, blank page is displayed as shown in the following figure.

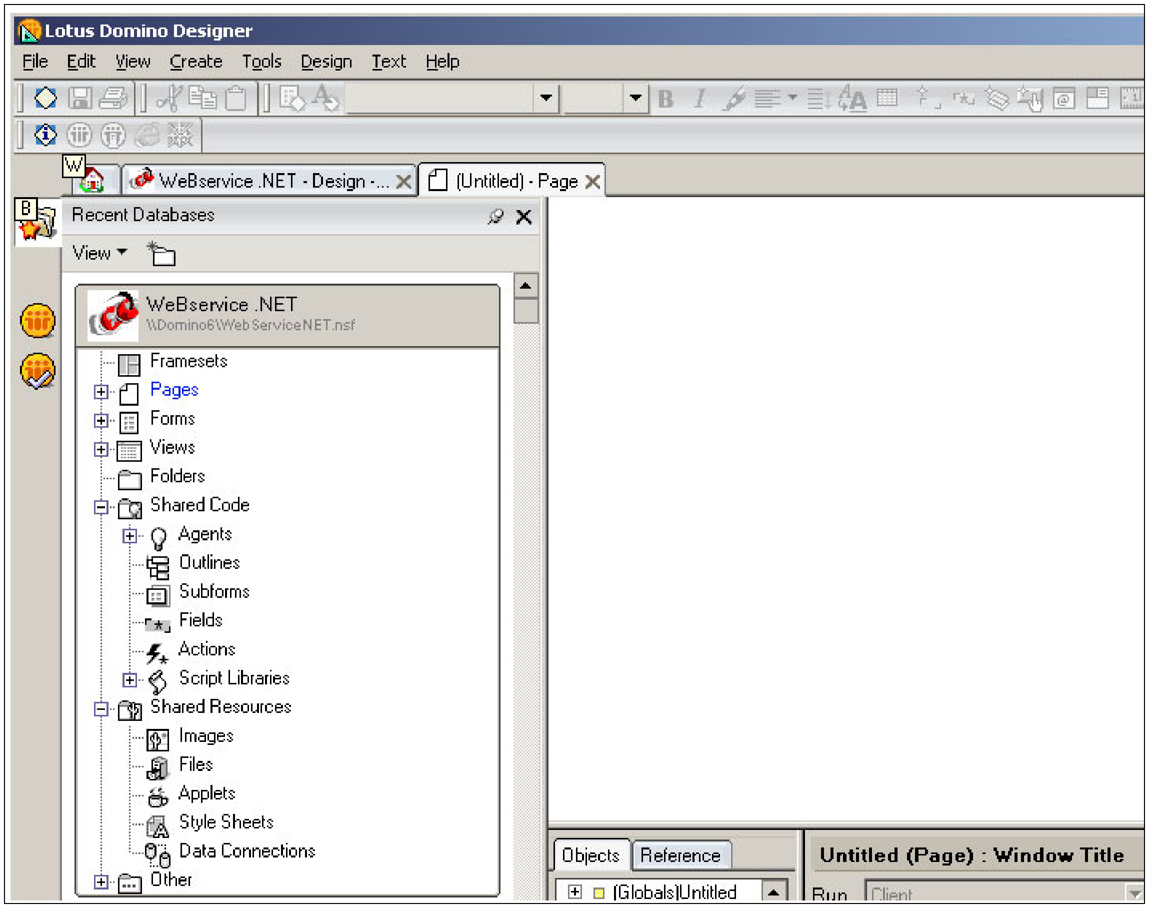
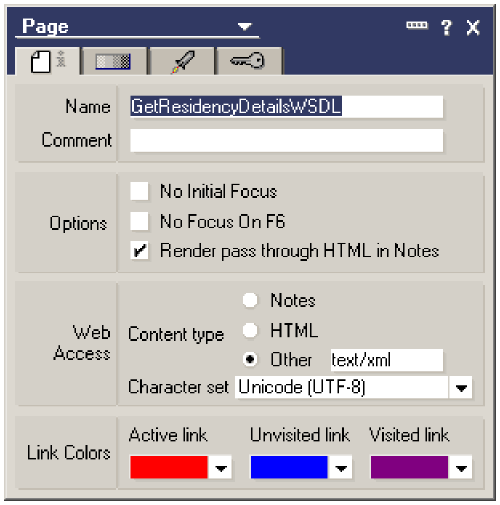


Figure 7 Create a new Page

2. Give `GetResidencyDetailsWSDL` as the Page Name and select **Other** in the Web Access - Content type section of the Page Info tab. Enter `text/xml` in the text box as depicted in the following figure.


Page

Name GetResidencyDetailsWSDL
Comment

Options
- [ ] No Initial Focus
- [ ] No Focus On F6
- [✓] Render pass through HTML in Notes

Web Access
Content type ○ Notes ○ HTML [● Other] text/xml
Character set Unicode (UTF-8)

Link Colors
Active link 🟥 Unvisited link 🟦 Visited link 🟪

Figure 8 Set the page content to XML

Leave the Character set to `Unicode(UTF-8)` and close the Properties box.

3. Create the WSDL file from scratch taking the following things into account:

   a. The root element of the WSDL file is the <*definitions*> element, which defines the namespaces used in the file. The targetNamespace should be the name of the Domino Lotus Script Library.

   b. The WSDL will contain two <*message*> type of interaction between the service requestor and the service provider: the first type for the request and the other for the response. The message request will contain the Residency Code as the string type and the message response will contain the data retrieved also as the string type.

   c. The <*portType*> will be a request-response operation type which means that there will be an input message (defined in the message part of the WSDL file as the message request) followed by an output message (defined in the message part of the WSDL file as the message Response).

   d. In the <*binding*> part, you will have to specify the following:

      - A name for the binding.
      - The connection should be SOAP HTTP; the style must be RPC.
      - The operation name must be the method name to be executed. In our case, this will be the function (*ResCodeSearch*) inside the Domino Script library.
      - A SOAP Action.

- A reference for the SOAP operation defining an input message and an output message, both to be SOAP encoded because RPC/Literal Web Service calls are not supported by Microsoft .NET. Notice that both input and output messages must contain the name of the Lotus Script Library as the namespace (*namespace="uri:Domino"*).

e. In the <*service*> part, you will define the port that use the SOAP binding specified before and the URL for the Web Service.

Example 1 Web Service WSDL

```
<?xml version='1.0' encoding='UTF-8'?>
<definitions name="ResCodeSearch" targetNamespace="Domino"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="Domino"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <message name="ResCodeSearchRequest">
    <part name="code" type="s:string"/>
  </message>
  <message name="ResCodeSearchResponse">
    <part name="Code" type="s:string"/>
    <part name="Name" type="s:string"/>
    <part name="StartDate" type="s:string"/>
    <part name="EndDate" type="s:string"/>
    <part name="Contact" type="s:string"/>
    <part name="email" type="s:string"/>
    <part name="Location" type="s:string"/>
  </message>
  <portType name="ResCodeSearchPortType">
    <operation name="ResCodeSearch">
      <input message="tns:ResCodeSearchRequest" />
      <output message="tns:ResCodeSearchResponse" />
    </operation>
  </portType>
  <binding name="ResCodeSearchBinding" type="tns:ResCodeSearchPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"
/>
    <operation name="ResCodeSearch">
      <soap:operation
soapAction="capeconnect:ResCodeSearch:ResCodeSearchPortType#ResCodeSearch" />
      <input>
        <soap:body use="encoded" namespace="uri:Domino"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded" namespace="uri:Domino"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  </binding>
  <service name="FindResCode">
    <port name="ResCodeSearchPort" binding="tns:ResCodeSearchBinding">
      <soap:address location="http://127.0.0.1/WebServiceNET.nsf/ResidencyWS"
/>
    </port>
  </service>
</definitions>
```

4. Save the GetResidencyDetailsWSDL page. It is possible to access this WSDL file from another development platform using the following URL:

   `http://<someserver>/WebServiceNet.nsf/GetResidencyDetailsWSDL?OpenPage`

Creating a .NET client

For consuming the Domino Web Service, we are going to create a simple console based Microsoft .NET application using C# as the programming language and Microsoft .NET Framework Software Development Kit V1.1.

Before creating this, it is necessary to understand the way by which the clients communicate with Web Services. Web Services use HTTP and SOAP to make the business data available on the Web. A Web Service Consumer will use SOAP over HTTP to execute Remote Procedure Calls (RPC) to Web Services methods components.

For security reasons, client applications will not execute the Web Services methods on the location where Web Services reside, but will use a 'proxy object' to act on behalf of the original Web Service. The proxy object at the client side communicates with the Web Service using HTTP/SOAP protocols. The WSDL file which describes the Web Service is used to generate the proxy object. The scenario is illustrated in the following figure.

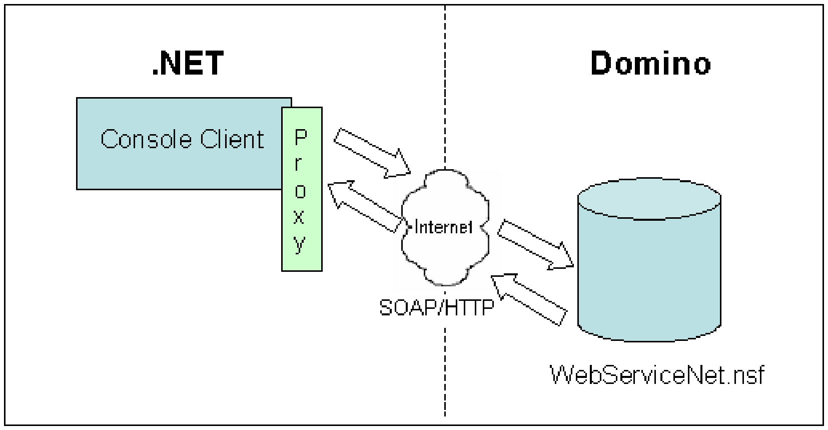


Figure 9 Console .NET application as Web Services consumer

Therefore, the necessary steps for building a .NET client will be:

1. Create a proxy object to allow communication between the console client and the Domino Web Service.
2. Create a simple C# console based application which will invoke methods of the proxy class.

Creating a proxy object to act on behalf the Web Service

For creating a proxy object, use the Microsoft Web Services Description Language Utility (wsdl.exe) included in Microsoft .NET Framework Software Development Kit V1.1. This utility will generate code for Web Service clients from WSDL files.

Note: A full description of the utility is out of the scope of this document but more information is found at the following URL:

`http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptools/html/cpgrfwebservicesdescriptionlanguagetoolwsdlexe.asp`

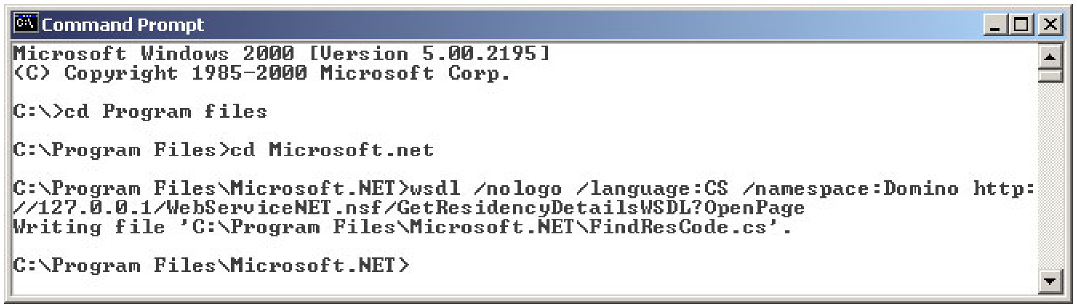
Help information is also available in a command window: **`wsdl /?`**

Before using the utility, make sure that the Domino server is running. After that, type this simple command to generate the proxy class:

```
wsdl /nologo /language:CS /namespace:Domino http://<Domino server
name>/WebServiceNET.nsf/GetResidencyDetailsWSDL?OpenPage
```

Where `<Domino server name>` is the host name for the Domino server or the IP address, `http://<Domino_server_name>/WebServiceNET.nsf/GetResidencyDeta` `ilsWSDL?OpenPage` is the URL where the WSDL is located and '`language:CS`' is the programming language used to generate the proxy class (C#).

The result of the command will be a file called FindResCode.cs as shown in the following figure.



```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>cd Program files

C:\Program Files>cd Microsoft.net

C:\Program Files\Microsoft.NET>wsdl /nologo /language:CS /namespace:Domino http:
//127.0.0.1/WebServiceNET.nsf/GetResidencyDetailsWSDL?OpenPage
Writing file 'C:\Program Files\Microsoft.NET\FindResCode.cs'.

C:\Program Files\Microsoft.NET>
```

Figure 10 Command line wsdl utility

Note that the name of the Proxy class is the same name given to the Service part in the WSDL file.

```
<service name="FindResCode">
   <port name="ResCodeSearchPort" binding="tns:ResCodeSearchBinding">
      <soap:address 
location="http://127.0.0.1/WebServiceNET.nsf/ResidencyWS" />
   </port>
<service>
```

The next step is to compile the file FindResCode.cs with the C# .NET compiler (csc.exe) included in the Microsoft .NET Framework SDK, by issuing the following command from the directory where the proxy class was generated:

```
csc /nologo /out:FindResCode.dll /target:library FindResCode.cs
```

The result of this command will be a Dynamic Link Library called FindResCode.dll (the proxy object).

Creating a simple C# console based application

For creating a simple console-style client application, add the following lines in any text editor, this will generate a new C# source file called NotesConsoleClient.cs.

Example 2 .NET client code

```
using System;
using Domino;
using System.Web.Services;
```

```
amespace NotesConsoleClient
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Client
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            FindResCode WSResCodeSearchService=new FindResCode();
            String 
rName="",rStartDate="",rEndDate="",rContact="",rEmail="",rLocation="";
            String result=WSResCodeSearchService.ResCodeSearch(args[0], out 
rName, out rStartDate, out rEndDate, out rContact, out rEmail, out rLocation);

            System.Console.Out.Write(
                "Name: "+rName+"\r\n"+
                "Start date: "+rStartDate+"\r\n"+
                "End date: "+rEndDate+"\r\n"+
                "Contact: "+rContact+"\r\n"+
                "e-mail: "+rEmail+"\r\n"+
                "Location: "+rLocation+"\r\n"+
                "---------------------------\r\n");
        }
    }
}
```

The purpose of this client is to show the details of a particular ITSO residency located in the Web Services .NET Database Application for a given Residency Code. These details will be: Name, Start date, End Date, Contact, e-mail and Location.

After saving the file, build an executable by compiling this code using the C# library (FindResCode.dll) created before and by issuing the following command:

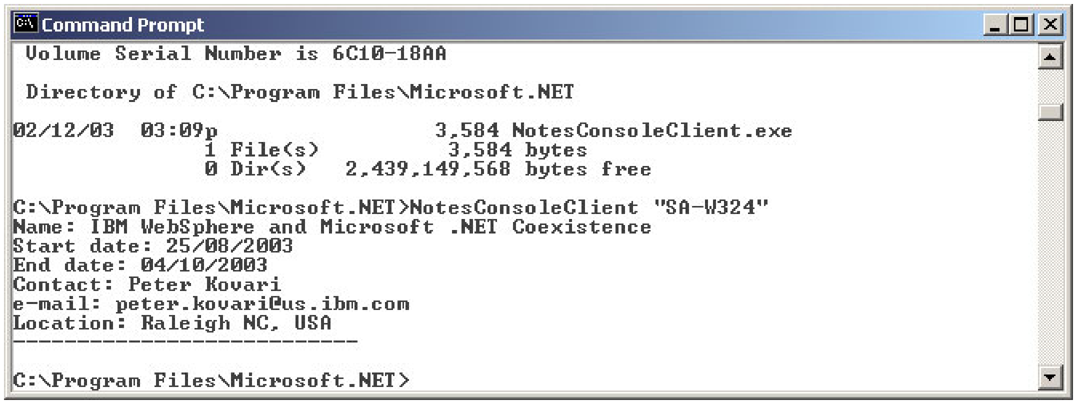
```
csc /r:FindResCode.dll NotesConsoleClient.cs
```

The result of this command will be an executable called NotesConsoleClient.exe.

Test the example

After building the *NotesConsoleClient.exe* executable (using C# library FindResCode.dll), run it from the command line window in the directory where the library is placed and after ensuring that the Domino Server is running.

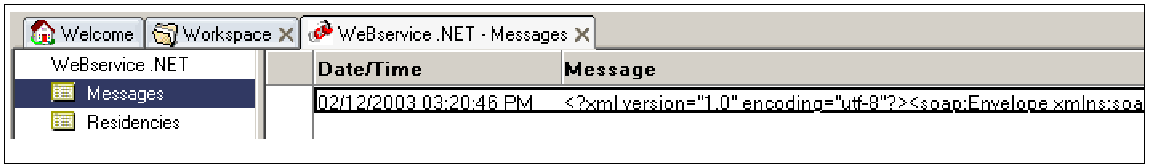
The NotesConsoleClient executable needs to have a Residency Code as an input; as an example, test the Domino Web Service retrieving the information of the Residency Code `SA-W324`; the result of running the client is shown in the following figure.



```
 Volume Serial Number is 6C10-18AA

 Directory of C:\Program Files\Microsoft.NET

02/12/03  03:09p                 3,584 NotesConsoleClient.exe
               1 File(s)          3,584 bytes
               0 Dir(s)   2,439,149,568 bytes free

C:\Program Files\Microsoft.NET>NotesConsoleClient "SA-W324"
Name: IBM WebSphere and Microsoft .NET Coexistence
Start date: 25/08/2003
End date: 04/10/2003
Contact: Peter Kovari
e-mail: peter.kovari@us.ibm.com
Location: Raleigh NC, USA
---------------------------

C:\Program Files\Microsoft.NET>
```

Figure 11 Client application results

When the execution of the client had finished, it is possible to check the SOAP incoming message generated by the .NET client message that was consumed and processed by our Domino agent. Open the WebServiceNet.nsf database in a Domino Client and select the **Message view**. A new Document will appear in the view as shown below.



Date/Time	Message
02/12/2003 03:20:46 PM	<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soa

Figure 12 Message View

Open the document and look at the format of the SOAP incoming message. Included within the SOAP Body is the method signature ResCodeSearch that will be executed on the Domino server. Additionally, the method signature contains the namespace where the method is located (Domino is our LotusScriptLibrary). Notice also that the Residency Code to search for is wrapped within the code argument. An example of the message is shown below with the method name, namespace (Domino Script Library), and method argument highlighted in bold:

```
<?xml version="1.0" encoding="utf-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="Domino" xmlns:types="Domino/encodedTypes"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body
soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<q1:ResCodeSearch xmlns:q1="uri:Domino"><code
xsi:type="xsd:string">SA-W324</code></q1:ResCodeSearch>
</soap:Body></soap:Envelope>
```

Note: Be aware that the format of the incoming SOAP message for Domino is very important and must include all the arguments highlighted in bold.

Because we cannot control the format of the SOAP incoming message form of the .NET client and because the Lotus Script Debugger in Domino is running from the Notes client, when experiencing problems with the format of the SOAP incoming message and Domino, a way to perform problem determination would be:

1. Create a new Lotus Script agent for using from a Lotus Notes Client (use the Lotus Script debugger). This agent will be the same as the ResidencyWS but with the `SOAPin` variable previously initialized in the way that Domino treats the SOAP incoming message as a string. For example, for the previous SOAP message, the `SOAPin` variable will be:

```
SOAPin=|<?xml version="1.0" encoding="utf-8"?>| &_
    |<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="Domino" xmlns:types="Domino/encodedTypes"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">|&_
    |<soap:Body
soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">|&_
    |<q1:ResCodeSearch xmlns:q1="uri:Domino">|&_
    |<code xsi:type="xsd:string">|&"SA-W324"&|</code>|&_
    |</q1:ResCodeSearch>|&_
    |</soap:Body>|&_
    |</soap:Envelope>|
```

2. Save the agent and launch the Lotus Script Debugger by selecting **Menu -> Tools -> Debug Lotus Script**.
3. Run your agent form the Actions Menu and cover all the steps to isolate the problem.

.NET service provider, Domino service consumer

Lotus Domino can also act as a service requester and consume a .NET service, which means that you can invoke this service programmatically. Programming languages inside Domino (Lotus Script and Java) and external tools such as the Microsoft SOAP Toolkit can be used for accessing to a external .NET Web

Service. The objective of this section is briefly to explain how Domino can use this tool for accessing .NET.

There are three ways to call a Web Service: HTTP GET, HTTP POST and SOAP.

Note: In Section , “Using the COM interface” on page 25 we created a .NET service that can access Domino databases through COM. Running the example in a Web browser will show that the NET Framework SDK will render all the information needed for accessing a .NET Web Service programmatically.

1. HTTP Get
   - Create a Lotus Script Agent for accessing a Web Service using the Microsoft XML parser included in the Internet Explorer 5.0.1 or later by setting the source using `CreateObject` ("Microsoft.XMLDOM"), using the load method to access the URL for the .NET Web Service, selecting the document with the documentElement method and then accessing the fields of the document with the selectSingleNode method.
   - Create a Java Agent using the following classes:
     - URL Class to access the .NET Web Service URL (for example: `http://localhost/webservices/sample.asmx/GetResidencyDetails?Code=SA-W324`) and the `URLConnection openConnection` method that returns a connection to the remote object referred to by the URL:

       ```
       URL url = new url
       (http://localhost/webservices/sample.asmx/GetResidencyDetails?Code=SA
       -W324);
       URLConnection connect = url.openConnection();
       ```

     - BufferedReader and InputStreamReader classes for reading the response from the Web Service.
2. HTTP Post

   Create a Domino Form placing HTML in it by using the `Form method=postaction` tag. HTTP Post will post data to the Web Service.
3. SOAP
   - Create a Java Agent that send a SOAP request to the Web Service and read the response with BufferedReader and InputStreamReader classes.
   - Create a Lotus Script Agent for accessing the Web Services using the Microsoft SOAP Toolkit. Download it from `http://msdn.microsoft.com`. Set the SOAP Client with `CreateObject` ("MSSOAP.SoapClient) and then initialize it with the `mssoapinit` method using the WSDL file.

Using the COM interface

COM (Component Object Model) is an open software component specification developed by Microsoft. It defines a specification for developing reusable binary components across multiple software languages and platforms. COM components can be written and called by any language that can call functions through pointers, including C, C++, Delphi, Basic etc.

The COM specification provides:

- Rules for component-to-component interaction.
- A mechanism for publishing available functions to other components.
- Automatic use tracking to allow components to unload themselves when they are no longer needed.
- Efficient memory usage.
- Transparent versioning.

Since Domino Release 5.0.2b, the back-end Domino objects have a COM interface with the following benefits:

- COM requires the presence of Domino or Notes; the software can be Domino server, Domino Designer, or Notes client.
- COM provides both early-binding (custom) and late-binding (dispatch) interfaces. Early binding makes the Domino classes available as typed variables with compile-time checking. Late binding can be used where the language (for example, VBScript) precludes early binding.
- The COM interface is the same as the Lotus Script interface, with some exceptions.
- Domino can act as a COM server or a COM Client.

Note: For information on COM properties and methods and general exceptions to Lotus Script specifications, refer to the Lotus Domino Designer 6 Help database. For information about the Domino Object Model, refer to *Domino Designer 6: A Developer's HandBook,* SG24-6854.

Microsoft .NET can access Domino Objects through COM. To accomplish this, .NET client or .NET Web Services can call Domino objects via a special wrapper. This wrapper; known as Runtime-Callable Wrapper (RCW), is a piece of software that can accept commands from a component, modify them and forward them to another component. Microsoft .NET Common Language Runtime (CLR) uses the RCW to operate with the unmanaged code by making COM calls to the Domino objects as depicted in the following figure.

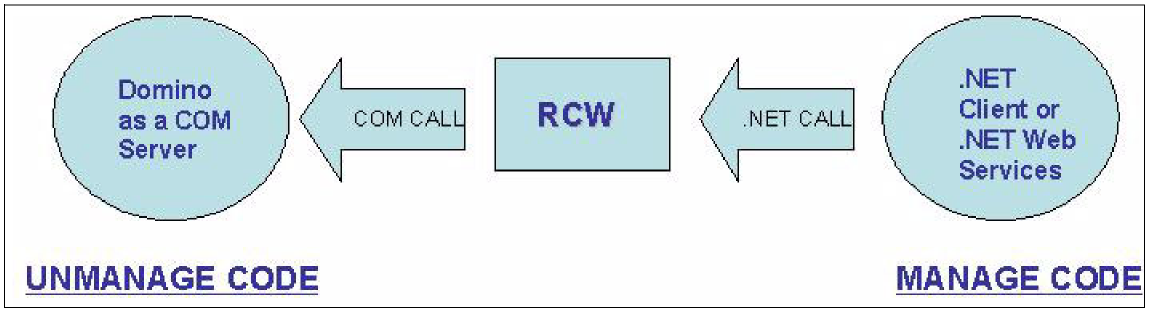


Figure 13 .NET call through RCW

In the same way, Lotus Domino can act as COM client and provide .NET objects via a special wrapper known as a COM Callable Wrapper (CCW). The Domino COM Client uses the CCW to operate with Microsoft .NET Common Language Runtime (CLR) by making .NET calls to .NET servers, as depicted in the following figure.

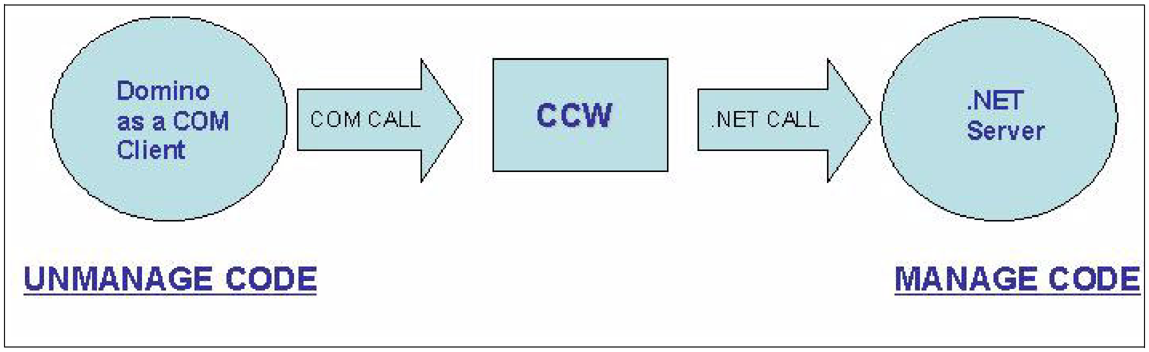


Figure 14 Domino call through CCW

The purpose of this section is to explain how .NET clients can access Domino databases through COM. The example shows how to create a .NET Web Service that returns the details of a particular ITSO Residency located in a Notes database.

For more information on the sample database, refer to , “Domino as a COM server, .NET as a client” on page 26, and , “Creating a Domino sample database” on page 41.

Domino as a COM server, .NET as a client

For running this example, we used the following product versions:

- .NET Framework Software Development kit V1.1
- Microsoft Windows 2000 Server with Service Pack 4
- Lotus Domino Server V6.0.2 CF2
- Microsoft Internet Information Services V5.0
- Microsoft IE V5.5 or newer
- Lotus Domino Administration Client V6.0.2
- Lotus Domino Designer Client V6.0.2

For more details on how to install the software products, refer to the installation manuals. Internet Information Services is included with Microsoft Windows 2000 and it is possible to install it under the Add/Remove Windows Components option inside Add/Remove Programs within the Control Panel.

Lotus Domino Administrator client and Lotus Domino Designer were installed in another machine in order to administrate the Domino Server and to design the sample application.

Before starting, make sure that you have TCP/IP network configured (it is recommended that you have a fixed IP address), that it is possible to resolve the machine host name (via Host file or DNS) and that you also have Domino Server configured. For our example, the following Domino nomenclature was selected within the configuration process, but of course it is possible to use other settings.

Table 3 Domino nomenclature

Concepts	**Selected Name**
Domino Domain Name	TEST
Organization Name	TEST
Server Name	Domin6/TEST
Server Title	Test Server
Notes Network Name	TCPIP Network
Notes Administrator Name	Notes Admin/TEST

Note: For more details on configuring a Domino Server, refer to the Lotus Domino Administrator 6 Help database.

Once you have installed and configured the products to begin with the example, follow these steps:

1. Set up Domino to work with IIS Server.
2. Make the Domino objects accessible to .NET and IIS.
3. Create a Domino sample database.
4. Create a .NET Web Service to access the Domino database.
5. Test the example.

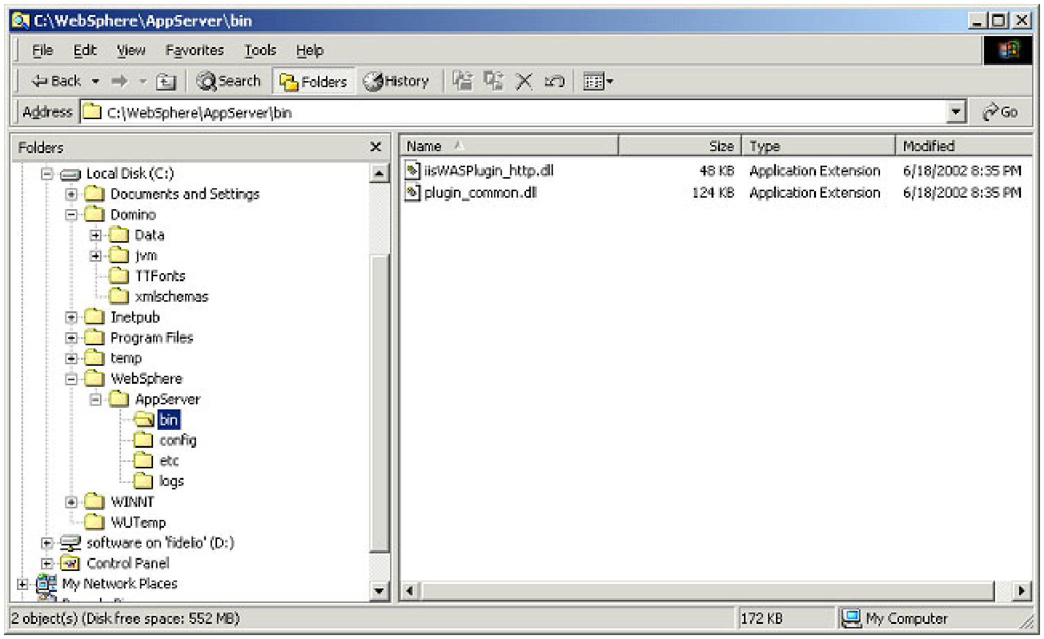
Setting up Domino to work with the Internet Information Services server

For using a Microsoft IIS Server as a front-end machine with Domino, it is necessary to install the WebSphere Application Server 4.0.3 plugin for IIS on the IIS server. The plugin files are packaged with the Domino 6 server and must be copied from the Domino Server to the IIS server. After copying the plugin files, configure the plugin. The last step is configuring the Domino server to work with the plugin IIS. Note that is not necessary to install any other WebSphere components on the IIS machine.

Install ing the WebSphere plugin on an IIS Server

1. First of all, create the following directory structure on the IIS machine (it is possible to use another drive):
   - C:\WebSphere\AppServer\bin
   - C:\WebSphere\AppServer\config
   - C:\WebSphere\AppServer\etc
   - C:\WebSphere\AppServer\logs
2. Then, copy the following files located in the Domino data directory to the IIS server:
   - <Domino data directory>/domino/plug-ins/plugin-cfg.xml to c:\WebSphere\AppServer\config.
   - <Domino data directory>/domino/plug-ins/w32/iisWASPlugin_http.dll to c:\WebSphere\AppServer\bin.
   - <Domino data directory>/domino/plug-ins/w32/plug-in_common.dll to c:\WebSphere\AppServer\bin.

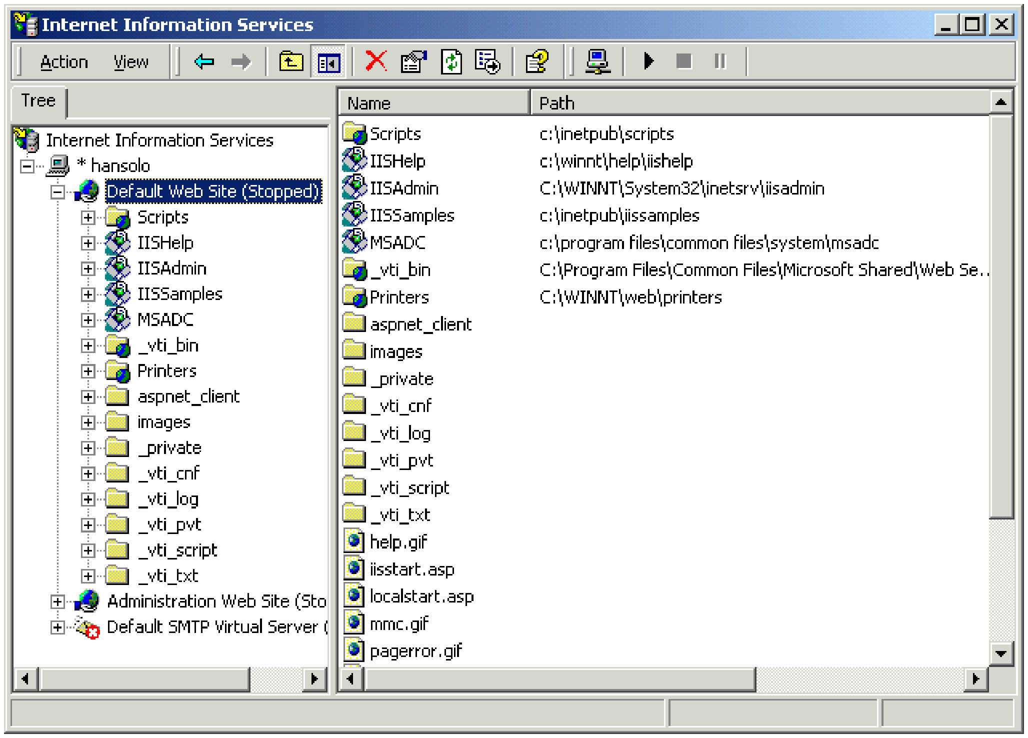
The directory structure is shown in the following figure.



```
C:\WebSphere\AppServer\bin
Address  C:\WebSphere\AppServer\bin

Name                      Size    Type                    Modified
iisWASPlugin_http.dll     48 KB   Application Extension   6/18/2002 8:35 PM
plugin_common.dll        124 KB   Application Extension   6/18/2002 8:35 PM

2 object(s) (Disk free space: 552 MB)                     172 KB   My Computer
```

Figure 15   Directory structure

3. Start the Internet Service Manager application by selecting **Start -> Settings -> Control Panel -> Administrative Tools -> Internet Service Manager**. Expand the Local Machine objects on the left pane to see all the services configured on it, as shown in the following figure.

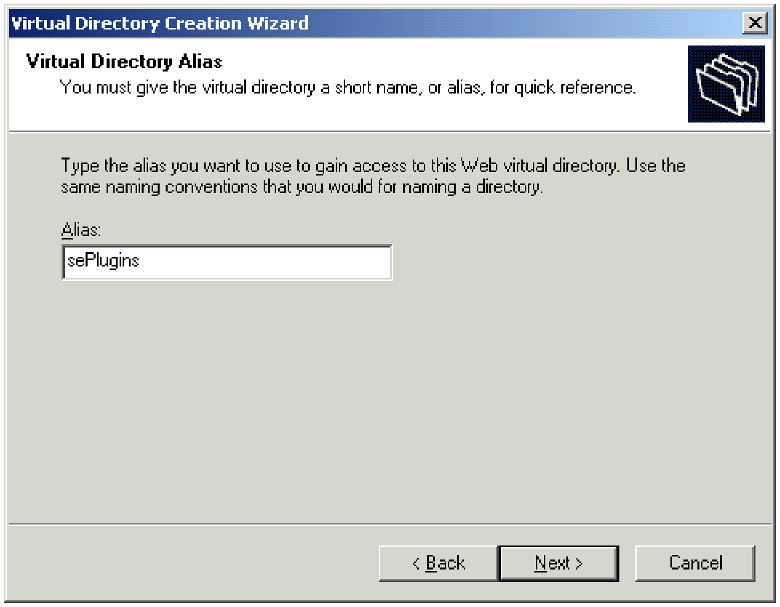


Internet Information Services
* hansolo
Default Web Site (Stopped)
Scripts
IISHelp
IISAdmin
IISSamples
MSADC
_vti_bin
Printers
aspnet_client
images
_private
_vti_cnf
_vti_log
_vti_pvt
_vti_script
_vti_txt
Administration Web Site (Sto
Default SMTP Virtual Server (

Name	Path
Scripts	c:\inetpub\scripts
IISHelp	c:\winnt\help\iishelp
IISAdmin	C:\WINNT\System32\inetsrv\iisadmin
IISSamples	c:\inetpub\iissamples
MSADC	c:\program files\common files\system\msadc
_vti_bin	C:\Program Files\Common Files\Microsoft Shared\Web Se..
Printers	C:\WINNT\web\printers
aspnet_client	
images	
_private	
_vti_cnf	
_vti_log	
_vti_pvt	
_vti_script	
_vti_txt	
help.gif	
iisstart.asp	
localstart.asp	
mmc.gif	
pagerror.gif	

Figure 16 IIS Services manager

4. Create a New Virtual Directory under the Default Web Site. IIS uses the virtual directories to access directories on other machines or directories outside a service's home directory. In this case, IIS uses the Virtual Directory for access to the WebSphere plugin. Right-click the **Default Web Site** and select **New -> Virtual Directory**. When the new Virtual Directory Creation Wizard is displayed, click **Next**.
5. Enter `sePlugins` in the alias field as illustrated below (always use this name) and click **Next**.


Virtual Directory Creation Wizard

Virtual Directory Alias

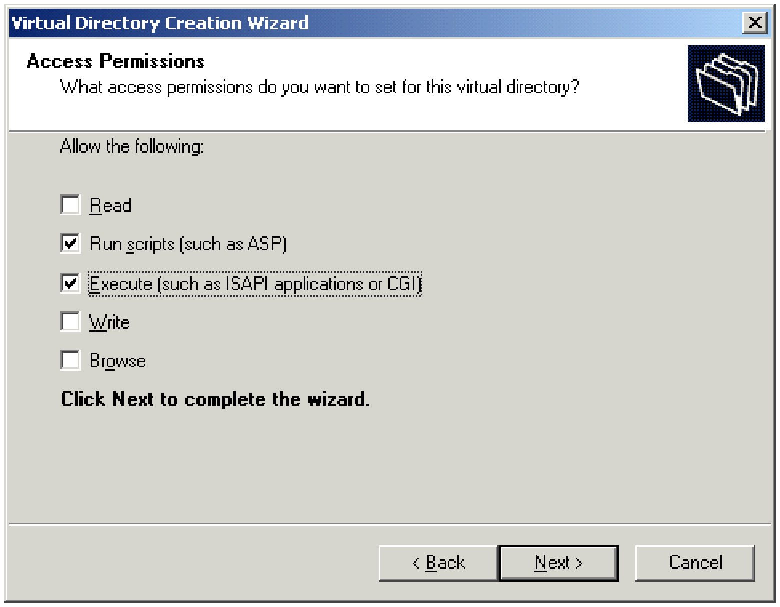
You must give the virtual directory a short name, or alias, for quick reference.

Type the alias you want to use to gain access to this Web virtual directory. Use the same naming conventions that you would for naming a directory.

Alias: sePlugins

Figure 17 Virtual Directory Alias

6. In the Directory field, browse to the WebSphere bin directory (C:\WebSphere\AppServer\bin). Click **Next**.
7. Select **Run Scripts** and **Execute** for the access permission as depicted in the following figure.


Virtual Directory Creation Wizard

Access Permissions

What access permissions do you want to set for this virtual directory?

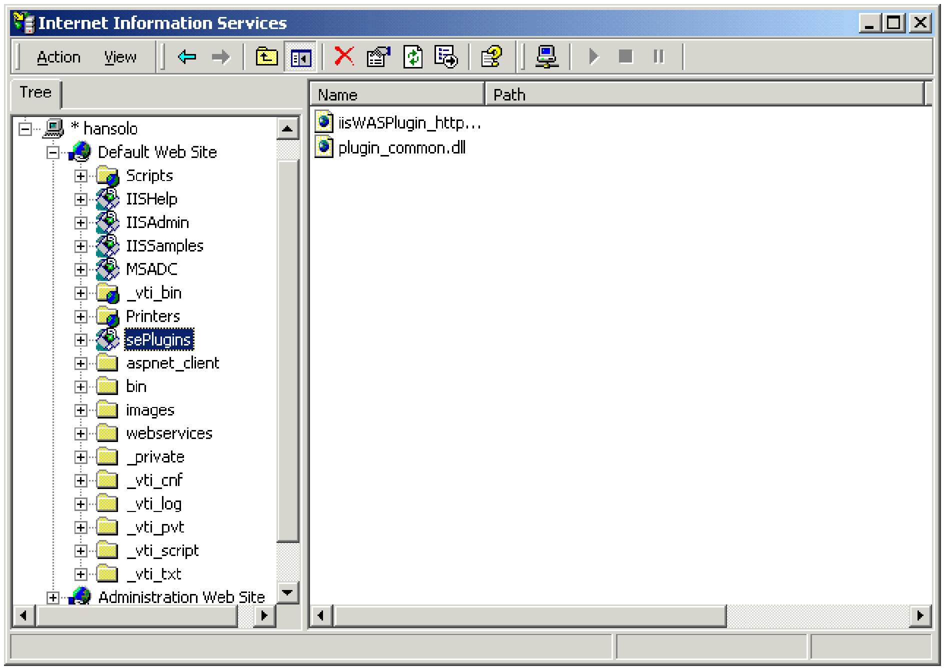
Allow the following:

- [ ] Read
- [✓] Run scripts (such as ASP)
- [✓] Execute (such as ISAPI applications or CGI)
- [ ] Write
- [ ] Browse

Click Next to complete the wizard.

Figure 18 Access Permissions

8. Click **Next** and then click **Finish**; the new Virtual Directory is shown on the default Web site.

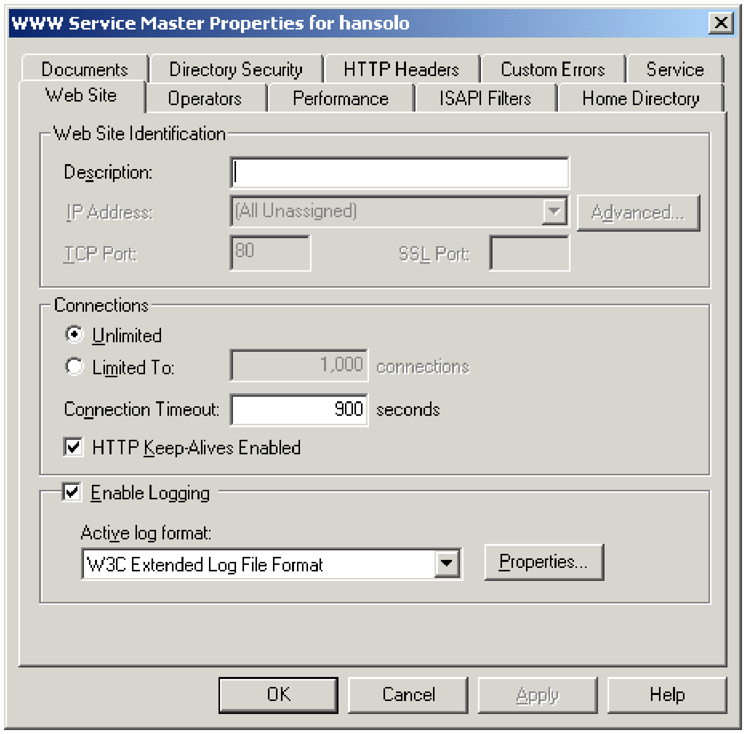


* hansolo
  - Default Web Site
    - Scripts
    - IISHelp
    - IISAdmin
    - IISSamples
    - MSADC
    - _vti_bin
    - Printers
    - sePlugins
    - aspnet_client
    - bin
    - images
    - webservices
    - _private
    - _vti_cnf
    - _vti_log
    - _vti_pvt
    - _vti_script
    - _vti_txt
  - Administration Web Site

Name	Path
iisWASPlugin_http...	
plugin_common.dll	

Figure 19 Virtual Directory in IIS Manager

9. Right-click the machine name and select **Properties**. On the Internet Information Services tab, select **WWW Service** as Master Properties and edit it. The WWW Service properties dialog window will be displayed as shown in the next figure.


WWW Service Master Properties for hansolo

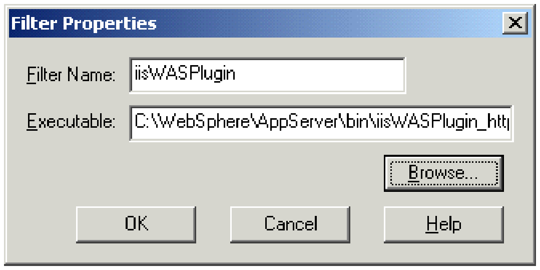
Web Site Identification
Description:
IP Address: (All Unassigned)
TCP Port: 80
SSL Port:

Connections
Unlimited
Limited To: 1,000 connections
Connection Timeout: 900 seconds
HTTP Keep-Alives Enabled

Enable Logging
Active log format: W3C Extended Log File Format

Figure 20 WWW Service Master Properties - Web Site

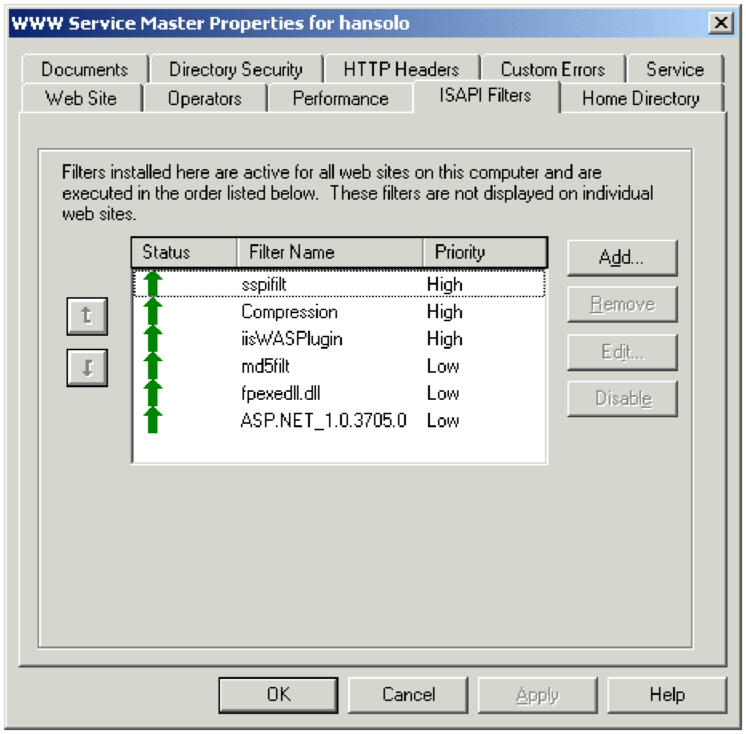
10. Select the **ISAPI Filters** tab and click **Add**; the Filter Properties Dialog window will be displayed. In the Filter Name Field type `iisWASPlugin` and for the Executable field, click **Browse** and open the WebSphere bin directory. Select **iisWASPlugin_http.dll** and click **OK**. The parameters are illustrated in the following figure.


Filter Properties

Filter Name: iisWASPlugin
Executable: C:\WebSphere\AppServer\bin\iisWASPlugin_htt

Figure 21 Filter properties

The new ISAPI filter is displayed as depicted in the following figure.



Filters installed here are active for all web sites on this computer and are executed in the order listed below. These filters are not displayed on individual web sites.

Status	Filter Name	Priority
⬆	sspifilt	High
⬆	Compression	High
⬆	iisWASPlugin	High
⬆	md5filt	Low
⬆	fpexedll.dll	Low
⬆	ASP.NET_1.0.3705.0	Low

Figure 22 WWW Service Master Properties - ISAPI Filters

11. Close all open windows.
12. Open the Windows Registry file and go to **HKEY_LOCAL_MACHINE -> Software -> IBM**. Create the key **`WebSphere Application Server`**. Then select **WebSphere Application Server** and create a new key, `4.0`. Then select **4.0** and create a new string value, `Plugin Config` and set the value for this variable to the location of the plugin-cfg.xml file, for example: `C:\WebSphere\AppServer\config\plugin-cfg.xml`.
13. Restart your system.

Configuring the WebSphere plugin

The WebSphere configuration file plugin-cfg.xml controls the operation of the plugin. In order for the plugin to relay requests to the target Domino server, it is necessary to add directives to the file for defining a transport route to the server, and pattern rules for the URL namespaces that identify requests which are to be relayed to Domino. The plugin will only relay requests that match a namespace rule. All other requests will be handled by the front-end Web server. To configure the plugin-cfg.xml file, follow these steps.

1. Open the file, plugin-cfg.xml, with WordPad.
2. Define a group identifying the Domino Server that handles NSF requests forwarded from IIS. Server groups contain servers, and servers contain transport definitions that give the plugin the information it needs to forward requests to Domino.

```
...
<ServerGroup Name="domino_web_servers">
   <Server Name="Domino Server">
      <!-- The transport defines the hostname and port value that the web
server plugin will use to communicate with the application server. -->
      <Transport Hostname="hansolo" Port="81" Protocol="http"/>
   </Server>
</ServerGroup>
```

Note: The Transport host name and Port number are the specified Host name and Port for our example machine. Substitute the values with the Host Name and Port number needed in every case.

3. Define a URI group which specifies the strings within a URL that indicate to IIS and the plugin that the request should be forwarded to Domino.

```
...
<UriGroup Name="domino_host_URIs">
<Uri Name="/*.nsf*"/>
<Uri Name="*/domjava*"/>
<Uri Name="*/icons*"/>
</UriGroup>
```

4. Define a virtual host group. Specify a Host Name and Port for the incoming requests or specify an asterisk (*) for the Host Name, Port, or both.

```
...
<VirtualHostGroup Name="domino_host">
   <VirtualHost Name="*:80"/>
   <VirtualHost Name="*:81"/>
</VirtualHostGroup>
```

5. Define a route to tie the sections together, so any request that matches the patterns listed in the domino_host_URIs group gets forwarded to the server(s) listed in the domino_web_servers group.

```
...
<Route ServerGroup="domino_web_servers" UriGroup="domino_host_URIs"
VirtualHostGroup="domino_host"/>
```

6. Stop and restart the World Wide Web Publishing Service from the Windows Services Control Panel.

Configuring Domino Server to work with Microsoft IIS

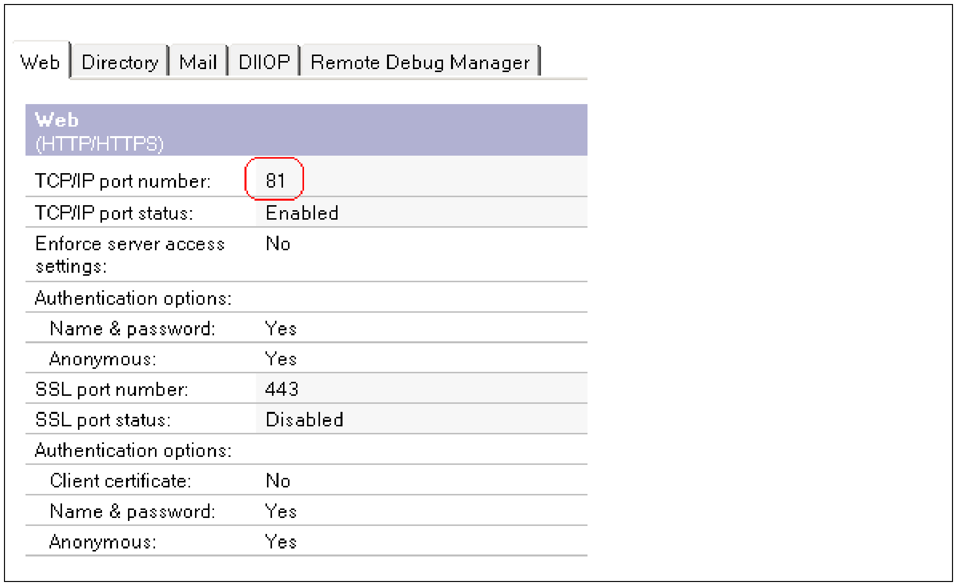
1. In the Domino Server, edit the Notes.ini file located in the Domino directory, in our case c:\Domino, and add the following line:

   ```
   HTTPEnableConnectorHeaders=1
   ```

   The setting enables the Domino HTTP task to process the special headers added by the plugin to requests. These headers include information about the front-end server's configuration and user authentication status. As a security measure, the HTTP task ignores these headers if the setting is not enabled. This prevents an attacker from mimicking a plugin.

2. Because the Domino Server is installed in the same machine as the IIS Server, it is necessary to change the default HTTP port for Domino (80) to an alternative number. We used the 81 port (this is why, in the plugin-cfg.xml file inside <VirtualHostGroup Name="domino_host">, both ports are specified). To change this, open the Domino Server Document from the Domino Administrator by selecting **Configuration Tab -> Server -> Current Server Document** and edit the field.

3. Select **Ports -> Internet ports -> Web** and specify the TCP/IP Port number that the Domino HTTP stack should use, as shown in the following figure.



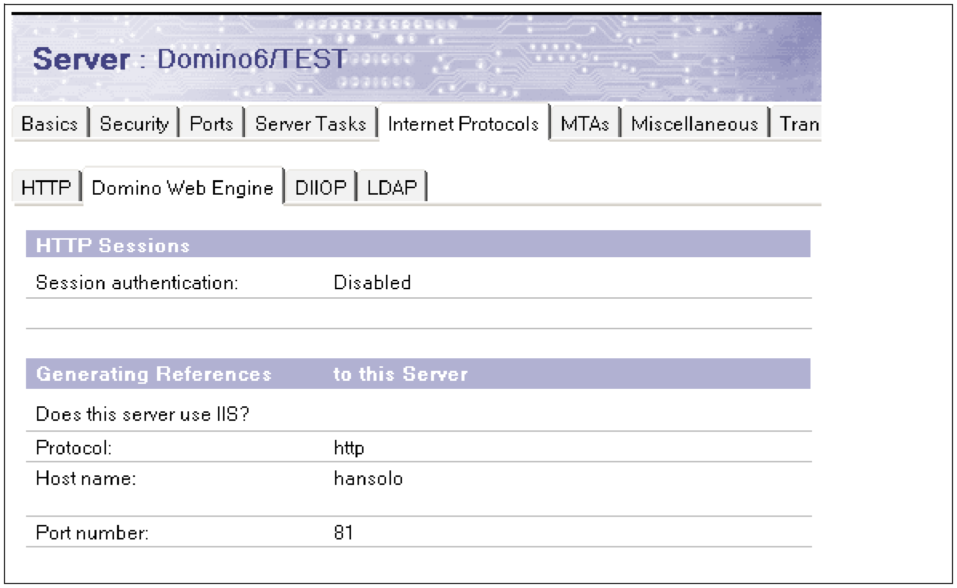
Web (HTTP/HTTPS)

TCP/IP port number:	81
TCP/IP port status:	Enabled
Enforce server access settings:	No
Authentication options:	
Name & password:	Yes
Anonymous:	Yes
SSL port number:	443
SSL port status:	Disabled
Authentication options:	
Client certificate:	No
Name & password:	Yes
Anonymous:	Yes

Figure 23   Domino - Web Internet Ports administration

Note: If Domino and IIS are on separate, dedicated machines, Domino can use port 80 on its own system and no change in the Server document is needed.

4. Select the **Internet Protocols -> Domino Web Engine** tab and configure the Generating References section by selecting the appropriate protocol, Host Name, and Port specified during the configuration of the WebSphere plugin, as depicted below.



HTTP Sessions

Session authentication:	Disabled

Generating References to this Server

Does this server use IIS?	
Protocol:	http
Host name:	hansolo
Port number:	81

Figure 24 Domino Web Engine administration

Note: For Domino 6, the setting "Does this server use IIS?" is not used.

Save the Domino Server Document with all the changes.

5. Make sure you have the HTTP Task running on the Domino Server. If not, add HTTP to the `ServerTasks` line of the Notes.ini file. This guarantees that every time the server starts, the HTTP Task is going to be loaded.

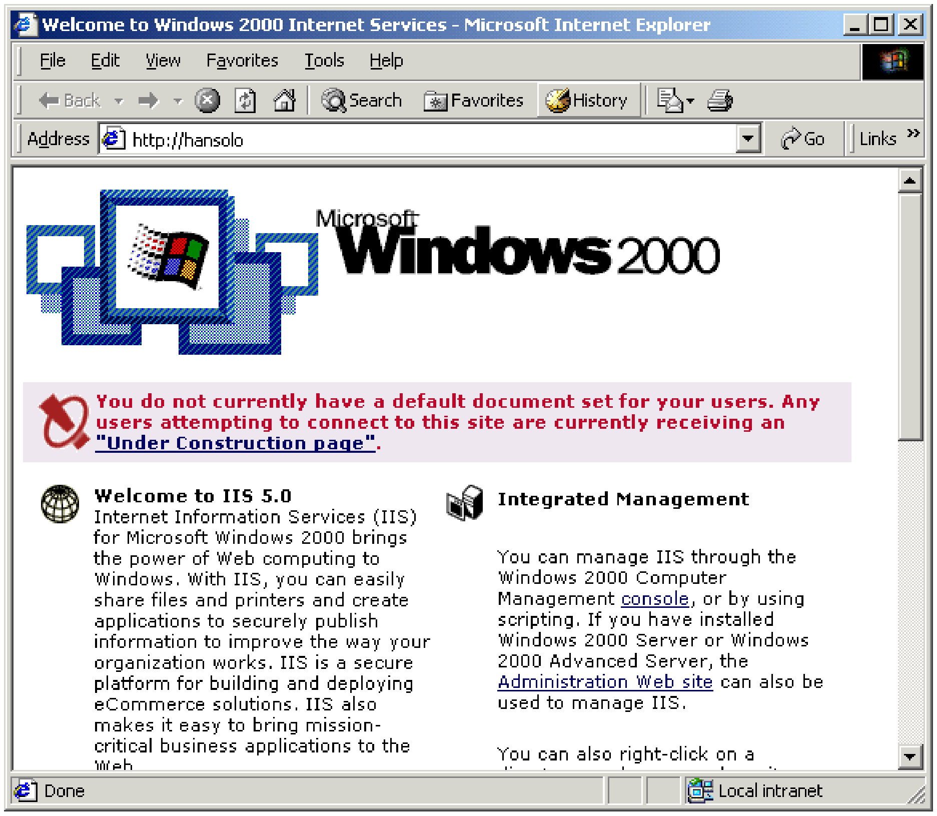
```
ServerTasks=Update,Replica,Router,AMgr,AdminP,CalConn,Sched,http
```

6. Restart the server.

Verifying the configuration

To verify the configuration, do the following:

1. Enter the URL for the Web server in your Web browser.
2. Verify that the IIS server's home page loads as shown below.


Welcome to Windows 2000 Internet Services - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://hansolo
Go Links
Microsoft Windows 2000
Done
Local intranet

You do not currently have a default document set for your users. Any users attempting to connect to this site are currently receiving an "Under Construction page".

Welcome to IIS 5.0
Internet Information Services (IIS) for Microsoft Windows 2000 brings the power of Web computing to Windows. With IIS, you can easily share files and printers and create applications to securely publish information to improve the way your organization works. IIS is a secure platform for building and deploying eCommerce solutions. IIS also makes it easy to bring mission-critical business applications to the Web.

Integrated Management

You can manage IIS through the Windows 2000 Computer Management console, or by using scripting. If you have installed Windows 2000 Server or Windows 2000 Advanced Server, the Administration Web site can also be used to manage IIS.

You can also right-click on a

Figure 25 Verify the plugin configuration

3. Append `homepage.nsf` to the URL in the address bar. If the Domino home page loads, the configuration is successful, as shown in the following figure.

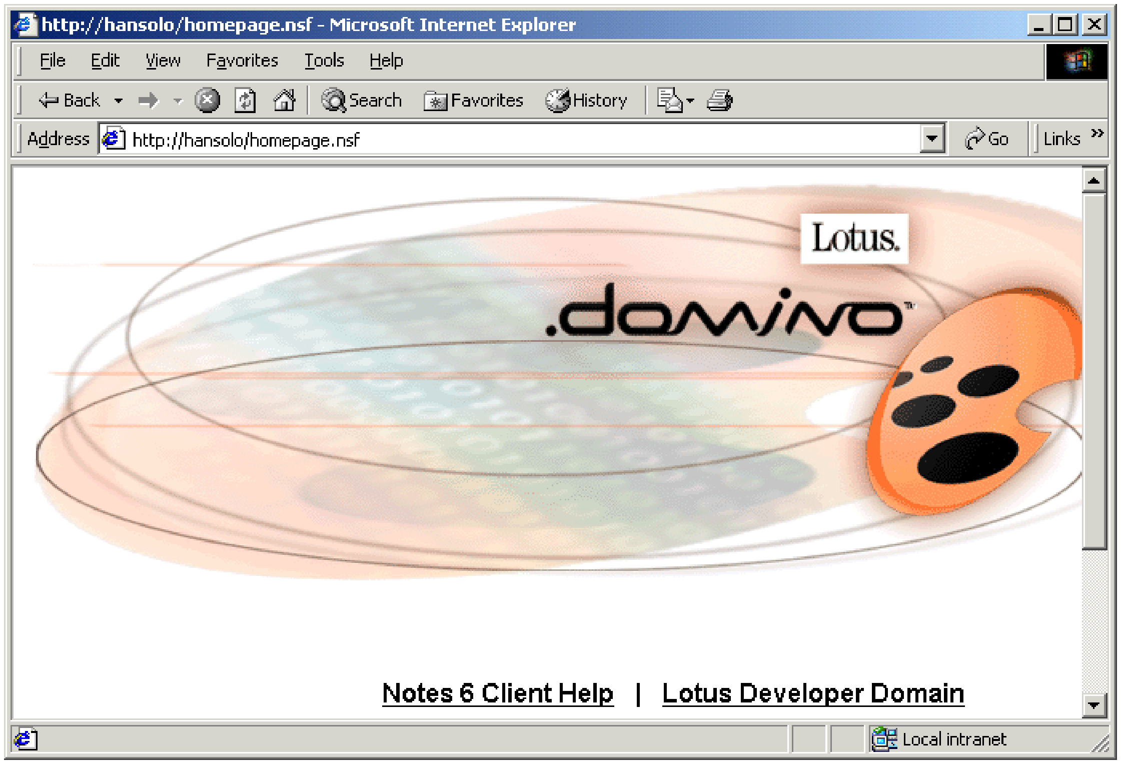

http://hansolo/homepage.nsf - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://hansolo/homepage.nsf
Go Links
Lotus.
.domino
Notes 6 Client Help | Lotus Developer Domain
Local intranet

Figure 26   Domino HTTP Server - homepage.nsf

Making the Domino Objects accessible to .NET and IIS

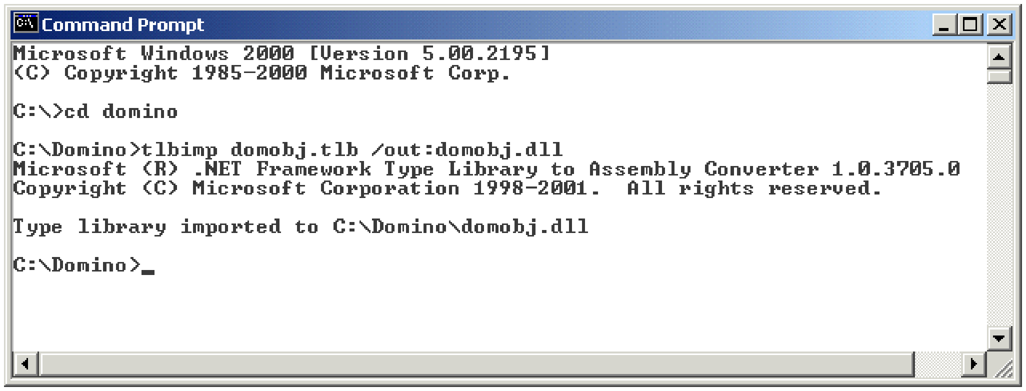
Because the Domino Objects are not included as standard within the .NET Framework Software Developer Kit (SDK), there is a tool included in the software called *Tlbimp* (type library imported) that reads the Domino COM Type Library (domobj.tlb) and creates a matching CLR assembly (domobj.dll) which will be in charge of calling the COM Components.

The Tlbimp tool is a command-line tool which will make the job of RCW easier because it is capable of converting COM metadata to .NET metadata.

Note: More information about Tlbimp tool can be found at the following URL:

`http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptools/html/cpgrfTypeLibraryImporterTlbimpexe.asp`

To create the domobj.dll, from the command prompt, go to the Domino Directory and type `tlbimp domobj.tlb /out:domobj.dll` as shown in the following figure.



Command Prompt

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>cd domino

C:\Domino>tlbimp domobj.tlb /out:domobj.dll
Microsoft (R) .NET Framework Type Library to Assembly Converter 1.0.3705.0
Copyright (C) Microsoft Corporation 1998-2001.  All rights reserved.

Type library imported to C:\Domino\domobj.dll

C:\Domino>_
```

Figure 27   Creating the Domino Object

Once you have created the domobj.dll, to make it accessible to Microsoft IIS, copy the DLL inside the bin directory included in c:\inetpub\wwwroot\.

Creating a Domino sample database

To show how Microsoft .NET can be integrated with Domino, we have created a Notes sample Web Services .NET database application (WebServiceNet.nsf) using Lotus Domino Designer 6.0.2, to serve as the repository for the upcoming ITSO residencies details.

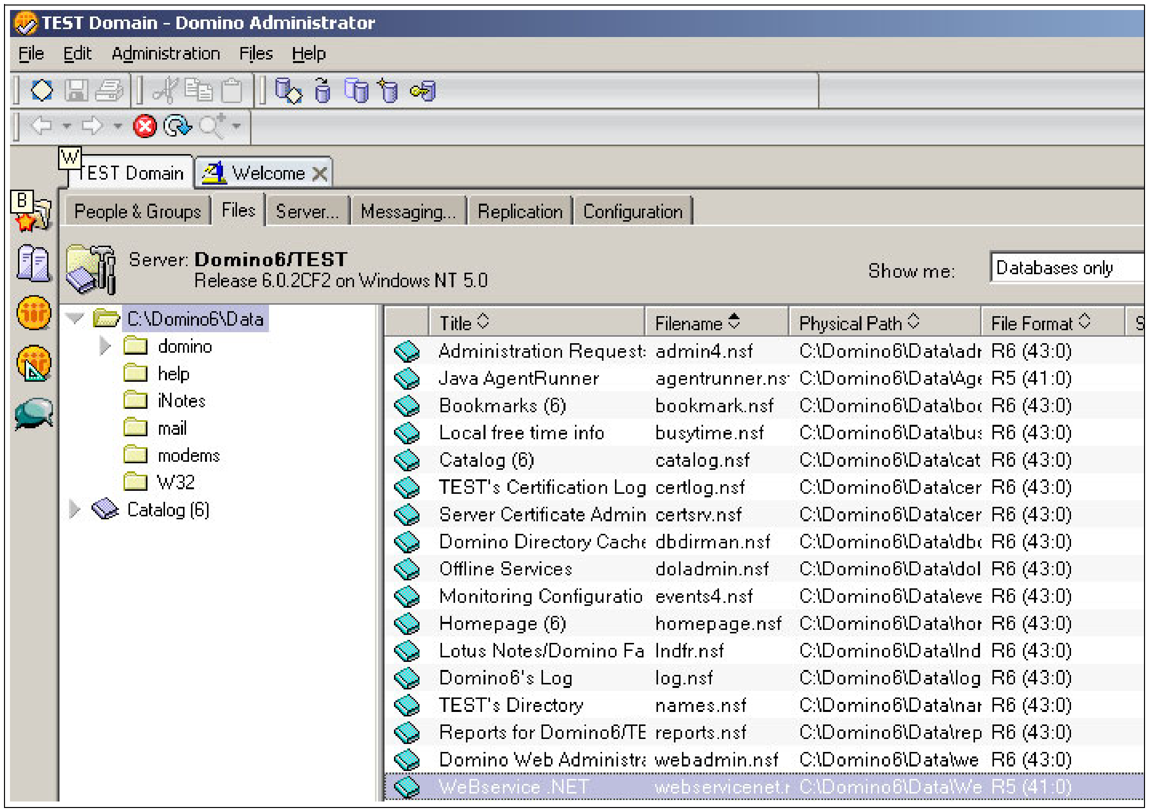
The application includes the following design elements:

- *Residencies Form*: this is the form used to create the information details for a new upcoming residency, such as Residency Name, Residency Code, Start Date, End Date, Residency Contact, e-mail and Location.
- *Residencies View*: the is the view that shows all the residencies to the user.
- *(By Code) View*: this hidden view is ordered by the Residency Code and is where the .NET and the Domino Web Service will find access to locate the residency.
- *Message Form*: this form is used for creating a document with the SOAP incoming message every time Domino Web Services is accessed.
- *Messages View*: this is the view that displays all the SOAP Incoming Messages documents created.
- *ResidencyWS*: this is the Domino Web Agent for our Domino Web Services and is the one in charge of routing the SOAP request, parsing it, calling the requested method (function), and returning the result as a SOAP response to the requester.

- *Domino Script Library*: this contains the method (function) for the Domino Web Service.
- *GetResidencyDetailsWSDL Page*: this contains the WSDL definition of the Domino Services.

To test the integration between both technologies, download the additional material that comes with this redbook, extract the database and then follow the next steps:

1. Copy the database to the Domino server Data Directory and open Lotus Domino Administrator 6.0.2.
2. Log in as a user with administrative privileges, then open the Domino Server from the left server bookmark pane; then click the **Files** tab as shown in the following figure.



Server: **Domino6/TEST**
Release 6.0.2CF2 on Windows NT 5.0

Show me: Databases only

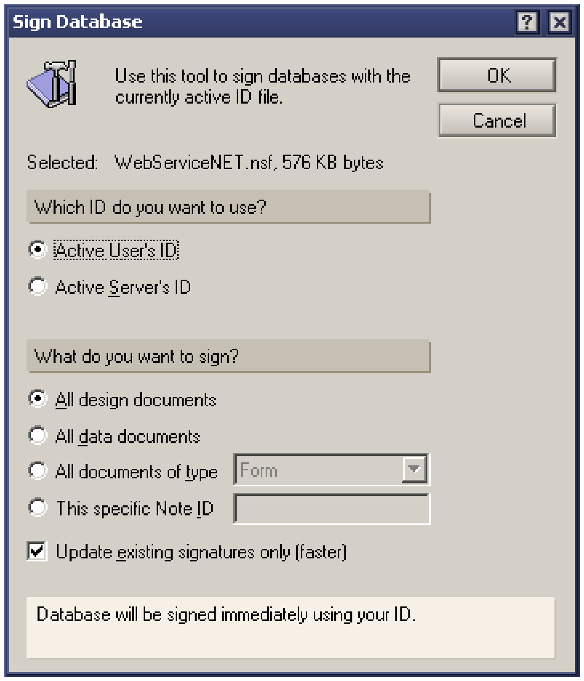
- C:\Domino6\Data
  - domino
  - help
  - iNotes
  - mail
  - modems
  - W32
- Catalog (6)

Title	Filename	Physical Path	File Format
Administration Request:	admin4.nsf	C:\Domino6\Data\adr	R6 (43:0)
Java AgentRunner	agentrunner.ns	C:\Domino6\Data\Age	R5 (41:0)
Bookmarks (6)	bookmark.nsf	C:\Domino6\Data\boo	R6 (43:0)
Local free time info	busytime.nsf	C:\Domino6\Data\bus	R6 (43:0)
Catalog (6)	catalog.nsf	C:\Domino6\Data\cat	R6 (43:0)
TEST's Certification Log	certlog.nsf	C:\Domino6\Data\cer	R6 (43:0)
Server Certificate Admin	certsrv.nsf	C:\Domino6\Data\cer	R6 (43:0)
Domino Directory Cache	dbdirman.nsf	C:\Domino6\Data\dbc	R6 (43:0)
Offline Services	doladmin.nsf	C:\Domino6\Data\dol	R6 (43:0)
Monitoring Configuratio	events4.nsf	C:\Domino6\Data\eve	R6 (43:0)
Homepage (6)	homepage.nsf	C:\Domino6\Data\hor	R6 (43:0)
Lotus Notes/Domino Fa	lndfr.nsf	C:\Domino6\Data\lnd	R6 (43:0)
Domino6's Log	log.nsf	C:\Domino6\Data\log	R6 (43:0)
TEST's Directory	names.nsf	C:\Domino6\Data\nar	R6 (43:0)
Reports for Domino6/TE	reports.nsf	C:\Domino6\Data\rep	R6 (43:0)
Domino Web Administra	webadmin.nsf	C:\Domino6\Data\we	R6 (43:0)
WeBservice .NET	webservicenet.r	C:\Domino6\Data\We	R5 (41:0)

Figure 28 Domino Administrator

3. Select the **WebService** .NET database application and open the Database toolbar located on the right side of the tools pane; select **Sign**.

4. A new dialog box appears. Leave the default parameters selected and click the **OK** button (see below).


Sign Database

Use this tool to sign databases with the currently active ID file.

Selected: WebServiceNET.nsf, 576 KB bytes

Which ID do you want to use?

(●) Active User's ID
( ) Active Server's ID

What do you want to sign?

(●) All design documents
( ) All data documents
( ) All documents of type: Form
( ) This specific Note ID

[✓] Update existing signatures only (faster)

Database will be signed immediately using your ID.

Figure 29 Domino - Sign Database

5. A new dialog box will appear stating `Your Name and Address Book does not contain a cross certificate for this organization` (for example: TEST). Click **Yes** to create a new cross-certificate.
6. All the design elements of the database will be signed with the actual ID. When the process is completed, a dialog box shows the number of databases processed and the number of errors that occurred (if any).

Creating a .NET Web Service to access the Domino database

To show how you achieve can .NET access to Domino using the COM interface, we have created a Web Service file (Sample.asmx) using a standard text editor (Notepad) and C# as the developing language.

The Web Service returns the details of a particular ITSO residency located in the Web Services .NET Database Application. When the Web Service is called, perform the following operations:

1. Open the Local Web Services .NET Database (WebServiceNet.nsf).

2. Open the hidden view (By Code) which contains all the residencies ordered by Residency Code.
3. For a given Residency Code, locate the corresponding document.
4. Access the fields inside the document.
5. Return the values for this particular residency.

Let's analyze the Web Service code:

1. The first line tells the compiler to run the code in Web Service mode and the name of the C# Class:

   ```
   <%@ WebService Language="C#" class="ResidencyDetailsWebService" %>
   ```

2. The next lines make references to the classes that the compiler needs to use during the compilation process. These classes are System and System.WebServices and of course, it will be necessary to use the Domino Objects (domobj.dll) classes for accessing the database.

   ```
   using System;
   using System.Web.Services;
   using domobj;
   ```

3. A C# program file can contain one or more namespaces; a namespace can also contain classes, structs, interfaces, etc. The following line makes a reference to a Web Service namespace which contains our ResidencyDetailsWebService Class, which inherits the functionality of the Web Service class:

   ```
   [WebService(Namespace="http://127.0.0.1/")]
   public class ResidencyDetailsWebService : WebService
   ```

4. The Web Service requires the user to enter a Residency Code and will return the details for this particular residency, such as Residency Name, Residency Code, Start Date, End Date, Residency Contact, e-mail and Location. To handle these data values, we used the C# "structs".

   ```
   // The object to return residency details
   public struct DocumentResult {
           public string ResCode;
           public string ResName;
           public string ResStartDate;
           public string ResEndDate;
           public string ResContact;
           public string ResLocation;
           public string Resemail;
       }
   ```

5. This Web Service is going to be accessed through HTTP. The data that we are going to access is not sensitive and is available to the public, so we used

the [Web method] keyword. The description tag inside the keyword is used to describe the Web Service functionality.

```
[WebMethod(Description="This method will get the Residency details for the
specified code.")]
public DocumentResult GetResidencyDetails(string Code) .
```

6. At this time, we are going to access the Notes database using the Domino classes. First, we need to create a NotesSession object by declaring the variable session and setting it as `New` to create a new instance for that object. Next, we initialize (explicitly) this COM session; there are two ways to achieve this:

   – *Using the Initialize Method*: this method can be used on a computer with a Notes client or Domino server and bases the session on the current user ID. If a password is specified, it must match the user ID's password.

   – *InitializeUsingNotesUserName*: this method can be used only on a computer with a Domino server. If a name is specified, the `InitializeUsingNotesUserName` method looks it up in the local Domino Directory and permits access to the local server depending on the "Server Access" and "COM Restrictions" settings. The password must match the Internet password associated with the name. If no name is specified, access is granted if the server permits Anonymous access.

   In our case, we used the second method because we used a computer with a Domino Server, specifying the user name and password.

```
// Connect to Notes and find the details for the residency.
NotesSession session = new NotesSession();
session.InitializeUsingNotesUserName("Notes Admin/TEST","lotusnotes");
```

   Note: Use the user name and password corresponding to your server.

7. The next step is to declare the variables `db`, `view`, `doc`, `Name`, `StartDate`, `EndDate`, `Contact`, `email` and `Location`. To get the values of the residencies' form fields, we need to follow the hierarchical path from the top to the lower one. In this example, we go from a NotesSession object to a NotesItem object:

   NotesSession -> NotesDatabase -> NotesView -> NotesDocument -> NotesItem

   We initialize the variable `db` with the property `GetDatabase`, indicating the server name (in our case "" because it is a local machine), database name (in our case `WebServiceNET.nsf`) and `false` for the `[createonfail]` parameter, of the higher level object (NotesSession). We set the object variable view using the `GetView` method, giving it the name of a view. We initialize the object variable `doc` using the `GetDocumentByKey` method, giving it the Residency Code and the true parameter because we want to find an exact match. The last step is to set the rest of the variables using the `GetFirstItem` method which for a given a name, returns the first item of the specified name belonging to the document.

```
NotesDatabase db = session.GetDatabase("", "WebServiceNET.nsf",false);
NotesView view = db.GetView("(By code)");
NotesDocument doc = view.GetDocumentByKey(Code,true);
NotesItem Name = doc.GetFirstItem ("Name");
NotesItem StartDate = doc.GetFirstItem ("SDate");
NotesItem EndDate = doc.GetFirstItem ("EDate");
NotesItem Contact = doc.GetFirstItem ("Contact");
NotesItem email = doc.GetFirstItem ("email");
NotesItem Location = doc.GetFirstItem ("Location");
```

Note: For more information about these Lotus Script Classes and Methods, refer to Lotus Domino Designer 6 Help.

8. Create a new DocumentResult object and assign the returning values recovered from the database to the initial parameters, defined in the DocumentResult struct, using the NotesItem Text property class.

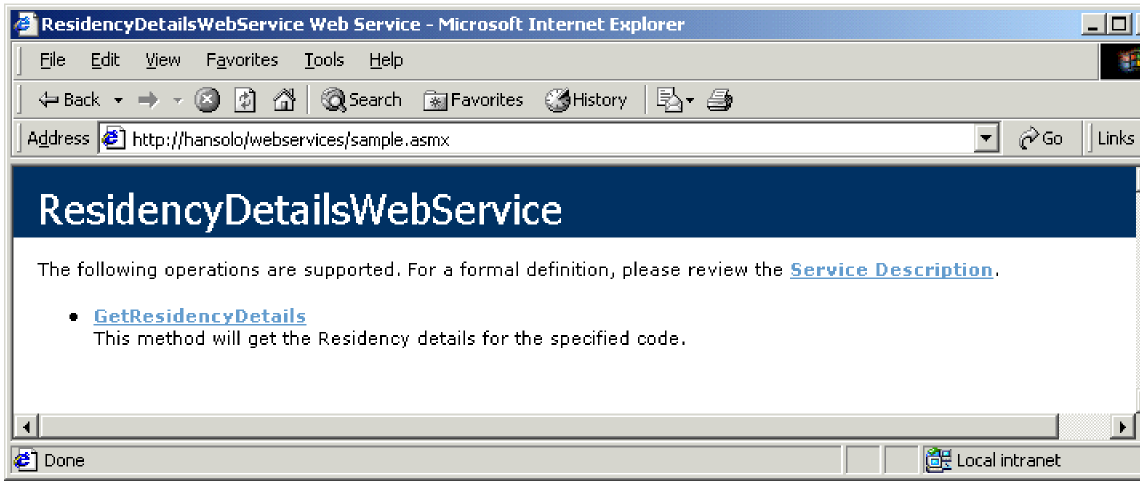
```
// Create a new DocumentResult object and return the values.
DocumentResult dr = new DocumentResult();
dr.ResCode = Code;
dr.ResName = Name.Text;
dr.ResStartDate = StartDate.Text;
dr.ResEndDate = EndDate.Text;
dr.ResContact = Contact.Text;
dr.Resemail = email.Text;
dr.ResLocation = Location.Text;
return dr;
```

9. Save the file with the .asmx extension.

Now, we are ready to test our Web Service; before proceeding, place the file (sample.asmx) inside the IIS Web directory path, for example c:\inetpub\wwwroot\webservices. If you do not have a Web Services directory, create one.

Testing the example

Open Microsoft IE Web Browser and type the URL: `http://<hostmachine>/webservices/sample.asmx`. It will bring up a page that is created automatically by the .NET Framework, as shown in the following figure.


ResidencyDetailsWebService Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://hansolo/webservices/sample.asmx
Go Links
Done
Local intranet

ResidencyDetailsWebService

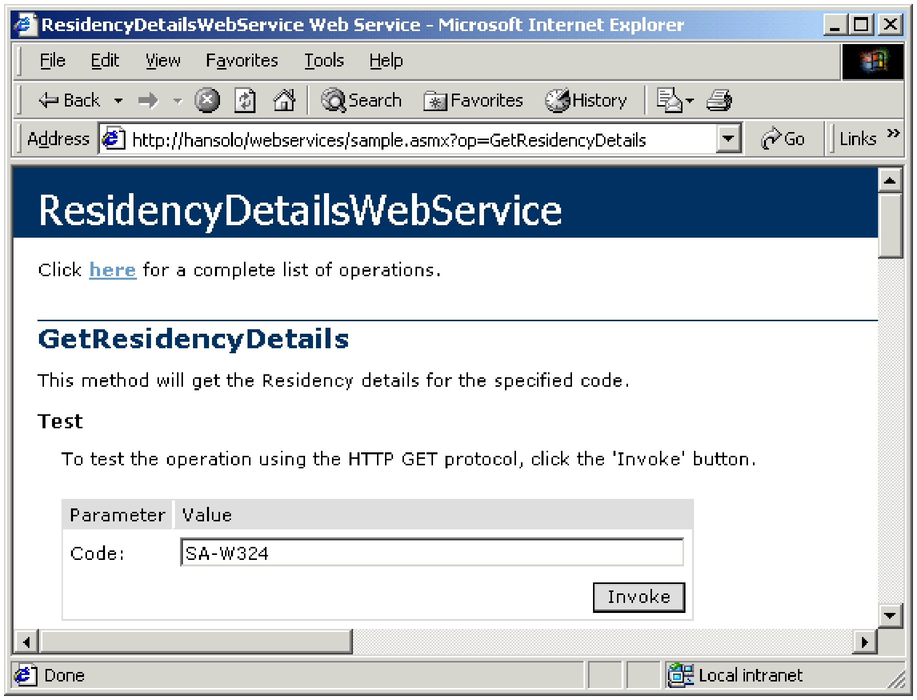
The following operations are supported. For a formal definition, please review the Service Description.

- GetResidencyDetails
  This method will get the Residency details for the specified code.

Figure 30 Web Service test client in IIS 1.

This page has two links: one for the `GetResidencyDetails` method defined in the ResidencyDetailsWebService class and a link for the WSDL file which describes the Web Service also created by .NET Framework SDK.

Click the **GetResidencyDetails** link and you will see the next page, also rendered by .NET Framework.


ResidencyDetailsWebService Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://hansolo/webservices/sample.asmx?op=GetResidencyDetails
Go Links
Done
Local intranet

ResidencyDetailsWebService

Click here for a complete list of operations.

GetResidencyDetails

This method will get the Residency details for the specified code.

Test

To test the operation using the HTTP GET protocol, click the 'Invoke' button.

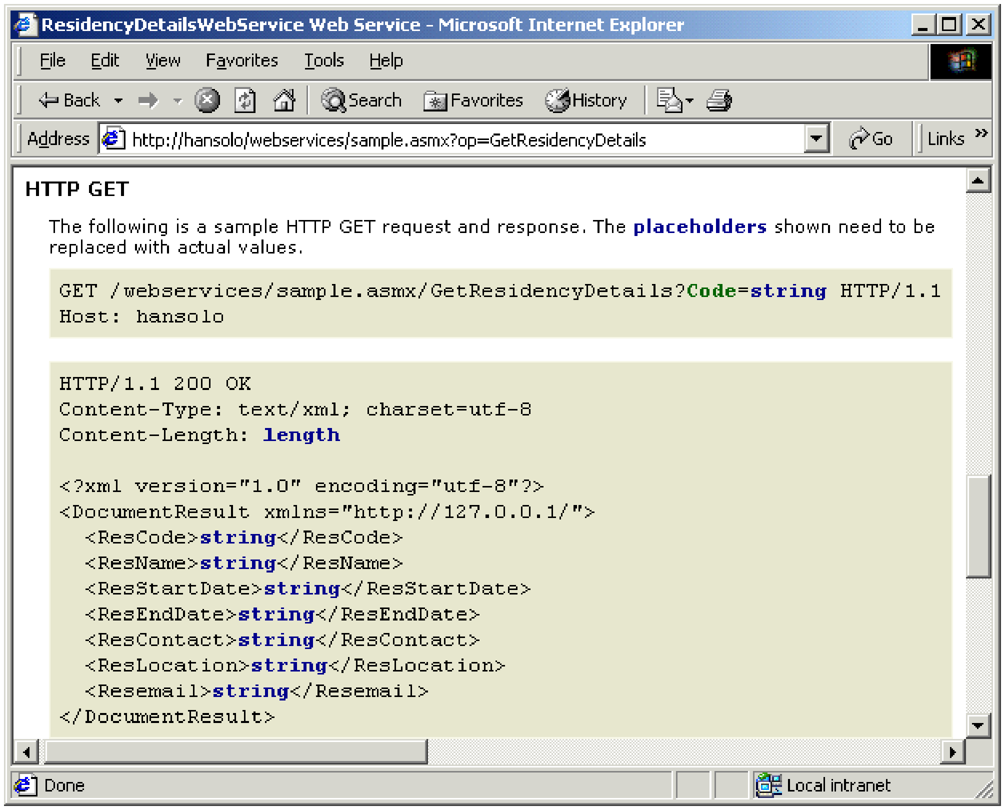
Parameter	Value
Code:	SA-W324

Invoke

Figure 31 Web Service test client in IIS 2.

This second page gives you the opportunity to test the Web Service and presents a good deal of useful information because the output (returned in the form of HTTP GET, HTTP POST and SOAP) provides all the hints you need for calling this Web Service programmatically, as shown in the following figures.

- HTTP GET:


ResidencyDetailsWebService Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://hansolo/webservices/sample.asmx?op=GetResidencyDetails
Go Links
Done
Local intranet

HTTP GET

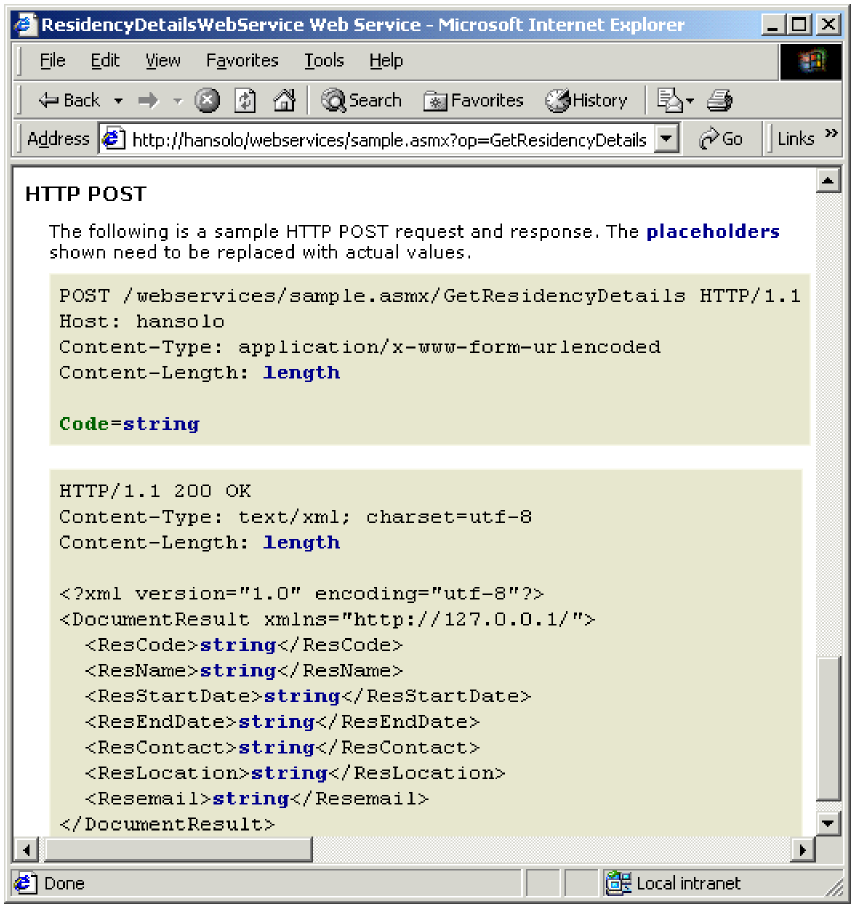
The following is a sample HTTP GET request and response. The **placeholders** shown need to be replaced with actual values.

```
GET /webservices/sample.asmx/GetResidencyDetails?Code=string HTTP/1.1
Host: hansolo
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<DocumentResult xmlns="http://127.0.0.1/">
  <ResCode>string</ResCode>
  <ResName>string</ResName>
  <ResStartDate>string</ResStartDate>
  <ResEndDate>string</ResEndDate>
  <ResContact>string</ResContact>
  <ResLocation>string</ResLocation>
  <Resemail>string</Resemail>
</DocumentResult>
```

Figure 32 Web Service request/response format - HTTP GET

- HTTP POST:


ResidencyDetailsWebService Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://hansolo/webservices/sample.asmx?op=GetResidencyDetails
Go Links
Done
Local intranet

HTTP POST

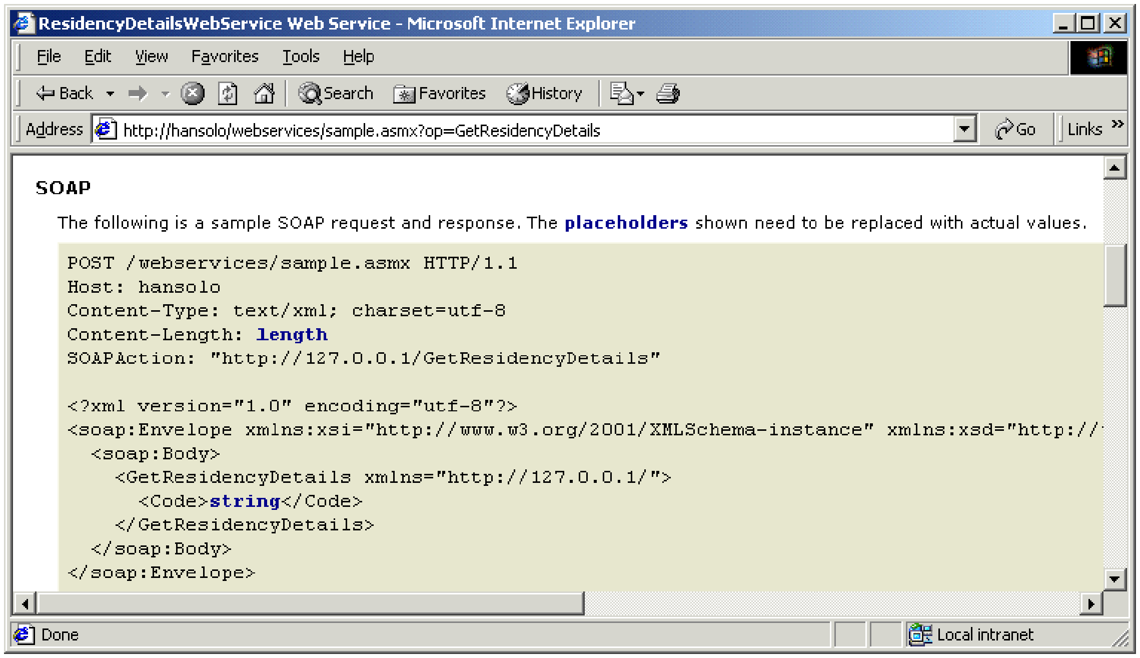
The following is a sample HTTP POST request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /webservices/sample.asmx/GetResidencyDetails HTTP/1.1
Host: hansolo
Content-Type: application/x-www-form-urlencoded
Content-Length: length

Code=string
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<DocumentResult xmlns="http://127.0.0.1/">
  <ResCode>string</ResCode>
  <ResName>string</ResName>
  <ResStartDate>string</ResStartDate>
  <ResEndDate>string</ResEndDate>
  <ResContact>string</ResContact>
  <ResLocation>string</ResLocation>
  <Resemail>string</Resemail>
</DocumentResult>
```

Figure 33 Web Service request/response format - HTTP POST

- SOAP Request:


ResidencyDetailsWebService Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://hansolo/webservices/sample.asmx?op=GetResidencyDetails
Go Links
Done
Local intranet

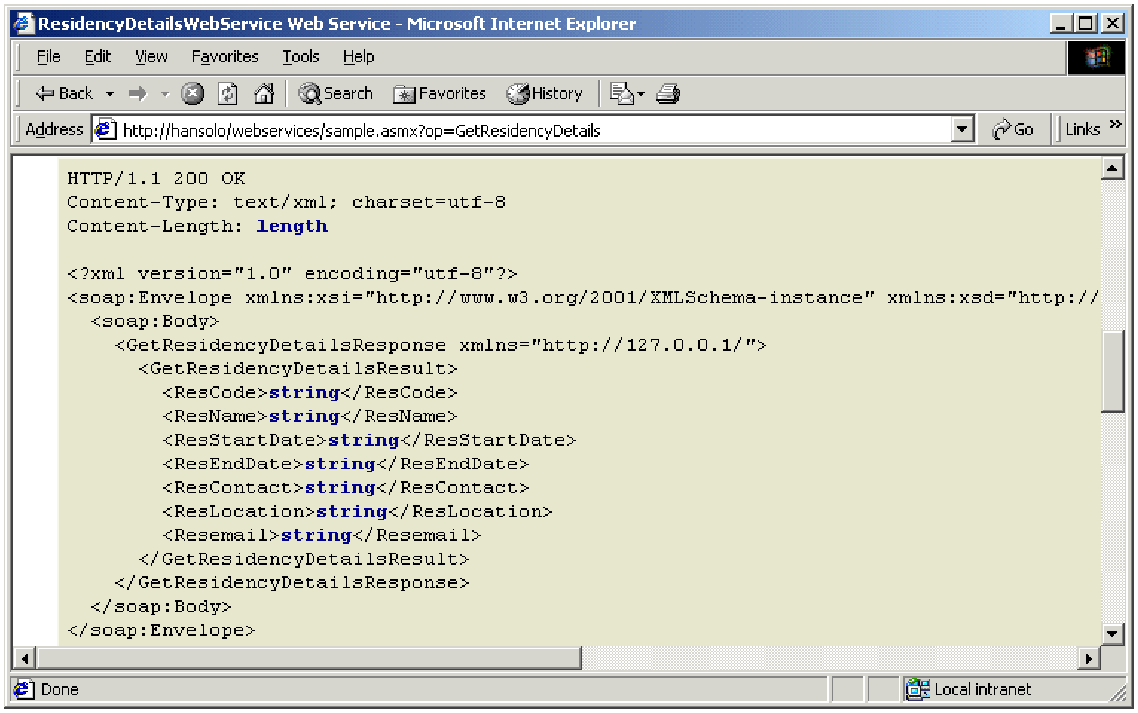
SOAP

The following is a sample SOAP request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /webservices/sample.asmx HTTP/1.1
Host: hansolo
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://127.0.0.1/GetResidencyDetails"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
  <soap:Body>
    <GetResidencyDetails xmlns="http://127.0.0.1/">
      <Code>string</Code>
    </GetResidencyDetails>
  </soap:Body>
</soap:Envelope>
```

Figure 34 Web Service request format - SOAP

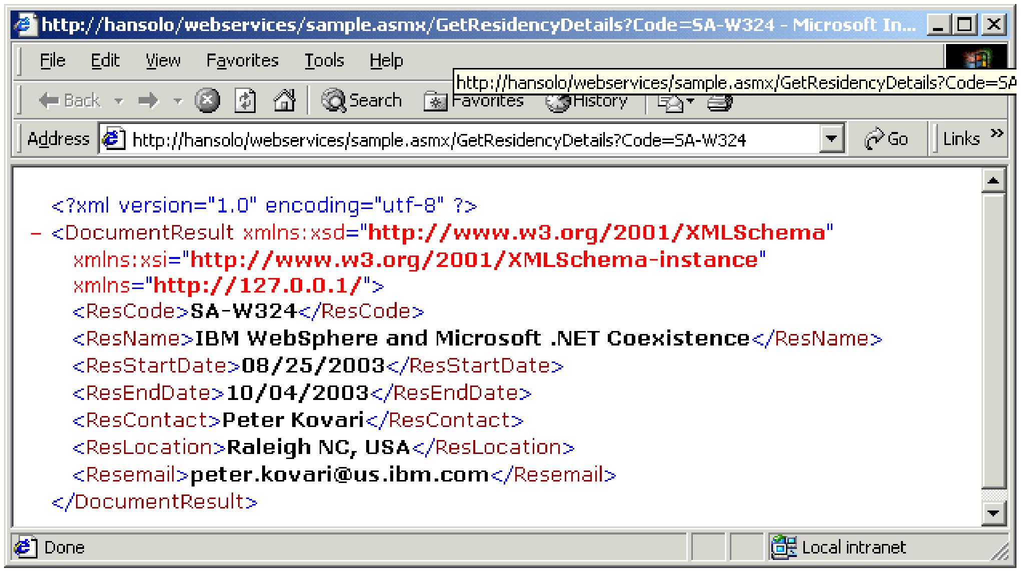
- SOAP Response:


ResidencyDetailsWebService Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://hansolo/webservices/sample.asmx?op=GetResidencyDetails
Go Links
Done
Local intranet

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
  <soap:Body>
    <GetResidencyDetailsResponse xmlns="http://127.0.0.1/">
      <GetResidencyDetailsResult>
        <ResCode>string</ResCode>
        <ResName>string</ResName>
        <ResStartDate>string</ResStartDate>
        <ResEndDate>string</ResEndDate>
        <ResContact>string</ResContact>
        <ResLocation>string</ResLocation>
        <Resemail>string</Resemail>
      </GetResidencyDetailsResult>
    </GetResidencyDetailsResponse>
  </soap:Body>
</soap:Envelope>
```

Figure 35 Web Service response format - SOAP

Introduce a Residency Code and click the **Invoke** button. The XML result page is depicted in the next figure.


http://hansolo/webservices/sample.asmx/GetResidencyDetails?Code=SA-W324 - Microsoft In...
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://hansolo/webservices/sample.asmx/GetResidencyDetails?Code=SA-W324
Go Links
Done
Local intranet

```
  <?xml version="1.0" encoding="utf-8" ?>
- <DocumentResult xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://127.0.0.1/">
    <ResCode>SA-W324</ResCode>
    <ResName>IBM WebSphere and Microsoft .NET Coexistence</ResName>
    <ResStartDate>08/25/2003</ResStartDate>
    <ResEndDate>10/04/2003</ResEndDate>
    <ResContact>Peter Kovari</ResContact>
    <ResLocation>Raleigh NC, USA</ResLocation>
    <Resemail>peter.kovari@us.ibm.com</Resemail>
  </DocumentResult>
```

Figure 36 Sample XML response

The team that wrote this Redpaper

This Redpaper was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Peter Kovari is a WebSphere Specialist at the International Technical Support Organization, Raleigh Center. He writes extensively about all areas of WebSphere. His areas of expertise include e-business, e-commerce, security, Internet technologies and mobile computing. Before joining the ITSO, he worked as an IT Specialist for IBM in Hungary.

Victoria Amor is an IT Specialist in IBM WebSphere and Lotus Domino within IBM Spain. Her areas of expertise include WebSphere support, particularly the areas of security and administration, and design and consultancy in Lotus Domino. She has worked within IBM for six years, participating in remarkable e-business projects such as the Sydney Olympic Games as the responsible of all Lotus Domino Servers in the Games System Center. Currently, she is working for ITS department in Services Delivery. She has previously co-authored the IBM WebSphere V4.0 Advanced Edition: Security Redbook.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

This document created or updated on April 5, 2004.

Send us your comments in one of the following ways:
- Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks
- Send your comments in an Internet note to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HZ8 Building 662, P.O. Box 12195
  Research Triangle Park, NC 27709-2195 U.S.A.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

eserver®
Redbooks (logo) ™
developerWorks®
ibm.com®
Domino Designer®
Domino®
IBM®
Lotus Notes®
Lotus®
Notes®
Redbooks™
WebSphere®

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.