XML Web Services for Visual FoxPro Developers

Doug Hennig
Stonefield Systems Group Inc.
1112 Winnipeg Street, Suite 200
Regina, SK Canada S4R 1J6
Voice: 306-586-3341
Fax: 306-586-5080
Email: dhennig@stonefield.com
Web: http://www.stonefield.com
Web: http://www.stonefieldquery.com

Overview

XML Web services are becoming very important for application development. Based on XML, SOAP and open standards, XML Web services are the building blocks of distributed applications. Visual FoxPro 8 makes it easier than ever to register, explore, consume, and publish XML Web services. This document explores the use of XML Web services in VFP applications. You'll see how to use the new XML Web Service Builder to bind controls to XML Web services results with very little code. You'll also discover how to publish XML Web services written in VFP.

Introduction

Microsoft's vision of software development strongly embraces XML Web services. A Web service is nothing more than a component that sits on a Web server somewhere. A client calls a function of the component, using SOAP (Simple Object Access Protocol) as the transport mechanism for the function call, parameters, and return value, and does something with the results, which are usually in the form of XML. The reason Web services are becoming more important for application development is they can form the building blocks of distributed applications.

Although you could use Web services in VFP 6 using Microsoft's SOAP Toolkit, VFP 7 made it much easier by providing some wrapper classes and providing IntelliSense for Web services you register in VFP. VFP 8 greatly extends the use of Web services in VFP, making it easier than ever to work with them.

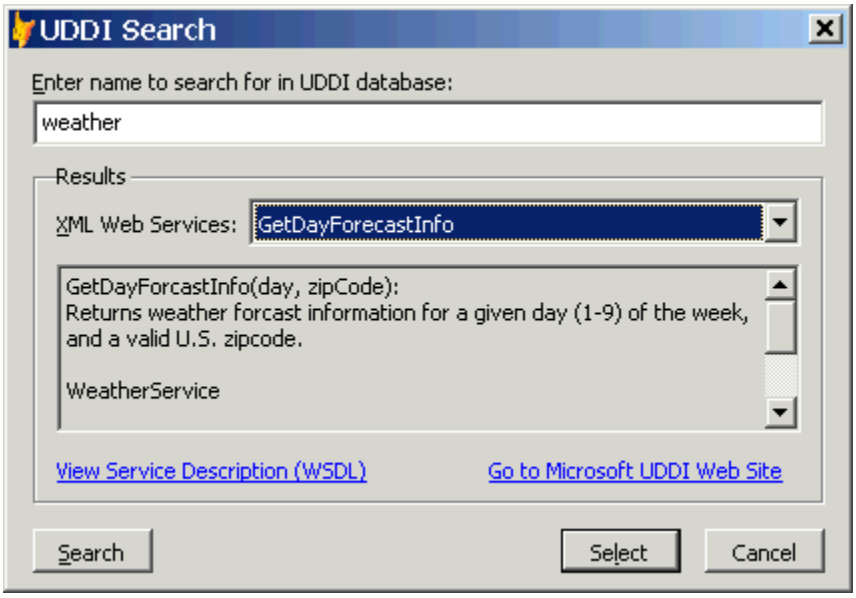
This document examines how to register, explore, consume, and publish Web services in VFP 8. It doesn't discuss the plumbing involved in Web services, such as SOAP, WSDL, and UDDI. There are numerous white papers available, including many on the MSDN Web site (http://msdn.microsoft.com), that go into great detail on that topic.

Registering Web Services

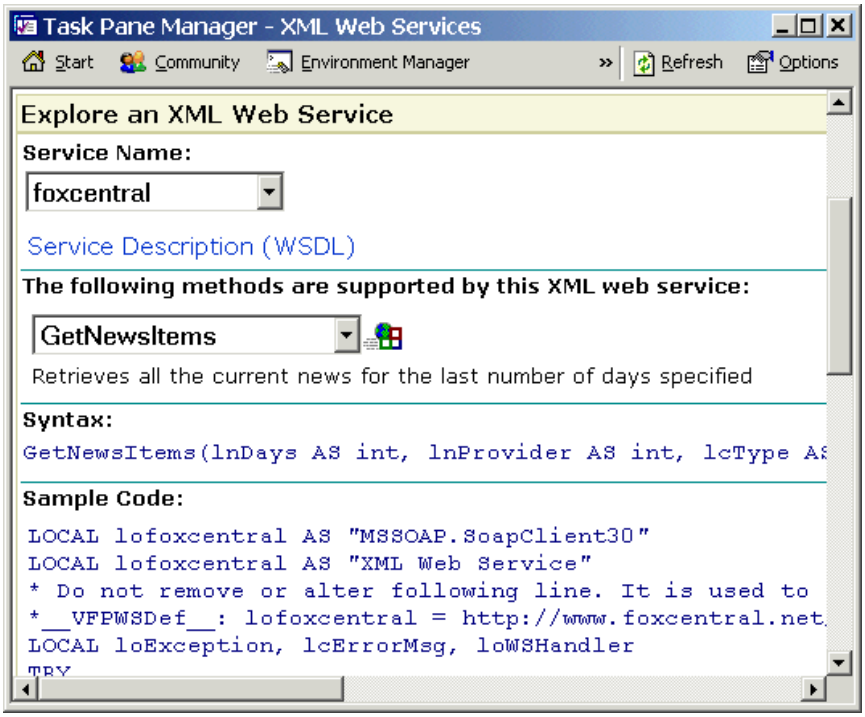
The first step to using a Web service is to register it with VFP. Although this isn't strictly necessary, it's easier to work with a Web services if you do. To register a Web service, launch the XML Web Services Registration dialog by clicking on the Web Services button in the Types page of the IntelliSense Manager (this was the only way to launch this dialog in VFP 7), clicking on the Register link in the My XML Web Services category of the new Toolbox, choosing the Register an XML Web Service link in the XML Web Services pane of the new Task Pane Manager, or selecting the New button in the XML Web Services Manager, which is also available from the XML Web Services pane of the Task Pane Manager.

Enter the URL, including the http:// prefix, for the WSDL of the Web service you want to register. If you don't know it, click on the UDDI Search button. In the UDDI Search dialog, type

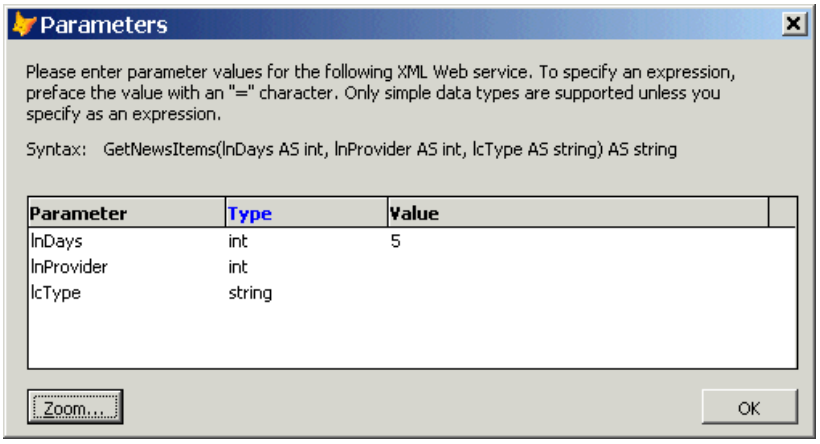
a name and click the Search button. The combo box is populated with a list of matching Web services from the UDDI database and the edit box below it shows information about the selected Web service.



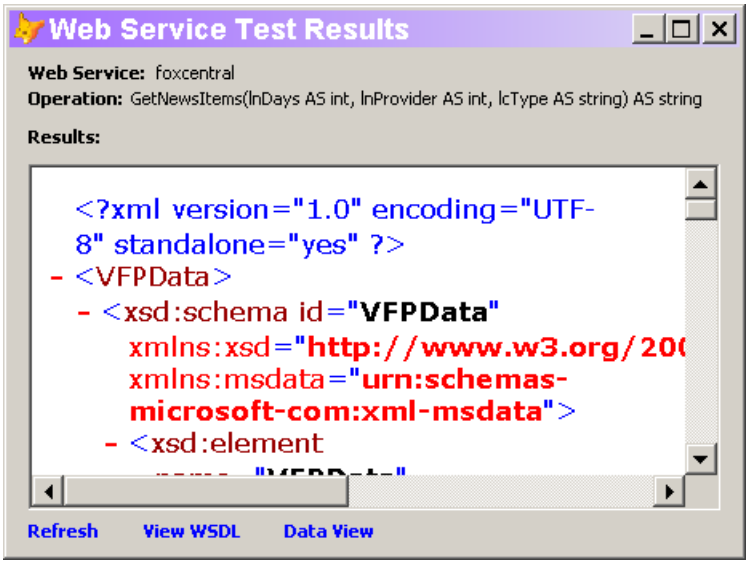
Once you've registered a Web service, you can find out how it works in the XML Web Services pane of the Task Pane Manager. Choose a Web service from the Service Name combo box and a method from the methods combo box, and the pane will show the syntax for the method and sample code to call it.



To test the selected method, click on the button next to the method combo box. If the method requires parameters, a dialog will appear in which you can enter values for the parameters.



The results from the Web service will then be displayed in a dialog as XML and, if the results can be converted into a VFP cursor, optionally in a grid.



Using Web Services

Now that you've registered and explored the Web service, it's time to use it. The way you use registered Web services in VFP 8 is totally different than it was in VFP 7. Rather than selecting a Web service from IntelliSense when you type LOCAL SomeVariable AS, you drag a Web service from the Toolbox to either a code window, which generates code, or a container (such as a form) in the Form or Class Designers, which adds an object to the container. Let's look at each of these.

Using Web Services Programmatically

Here's the code generated when you drag the FoxWiki Web service from the Toolbox to a code window:

```
LOCAL lowikiwebservice AS "MSSOAP.SoapClient30"
LOCAL lowikiwebservice AS "XML Web Service"
* Do not remove or alter following line. It is used to support IntelliSense
for
* your XML Web service.
*__VFPWSDef__: lowikiwebservice = http://fox.wikis.com/wikiwebservice.wsdl ,
*    wikiwebservice , wikiwebserviceSoapPort
LOCAL loException, lcErrorMsg, loWSHandler
TRY
  loWSHandler = NEWOBJECT("WSHandler", IIF(VERSION(2) = 0, "", ;
    HOME() + "FFC\") + "_ws3client.vcx")
  lowikiwebservice = loWSHandler.SetupClient( ;
    "http://fox.wikis.com/wikiwebservice.wsdl", "wikiwebservice", ;
    "wikiwebserviceSoapPort")
* Call your XML Web service here.  ex: leResult =
lowikiwebservice.SomeMethod()
CATCH TO loException
  lcErrorMsg = "Error: " + TRANSFORM(loException.Errorno) + " - " + ;
    loException.Message
  DO CASE
    CASE VARTYPE(lowikiwebservice)#"O"
      * Handle SOAP error connecting to web service
    CASE !EMPTY(lowikiwebservice.FaultCode)
      * Handle SOAP error calling method
      lcErrorMsg = lcErrorMsg + CHR(13) + lowikiwebservice.Detail
    OTHERWISE
      * Handle other error
  ENDCASE
  * Use for debugging purposes
  MESSAGEBOX(lcErrorMsg)
FINALLY
ENDTRY
```

This code uses the new structured error handling in VFP 8 so it handles errors gracefully. Also, if you used Web services in VFP 7, you'll notice this code uses a different class, WSHandler in _WS3Client.VCX rather than WSClient in _WebServices.VCX, to act as the Web service proxy.

If you'd rather avoid adding an FFC class to your application, you can also manually use the SoapClient COM object (the code is shown here without error handling for simplicity):

```
loWikiWebService = createobject('MSSOAP.SoapClient')
loWikiWebService.MSSoapInit("http://fox.wikis.com/wikiwebservice.wsdl", ;
   "wikiwebservice", "wikiwebserviceSoapPort")
```

If the Web service returns complex data, such as an ADO.NET Dataset, you may have to do some extra coding to make use of the results. XMLAdapterWebService.PRG shows how the new XMLAdapter base class can help. This program adds the following code to that generated by VFP for a Web service called NWWebService that returns an ADO.NET Dataset of records from the SQL Server Customers or Orders table (see the Appendix for a listing of the code for this Web service):

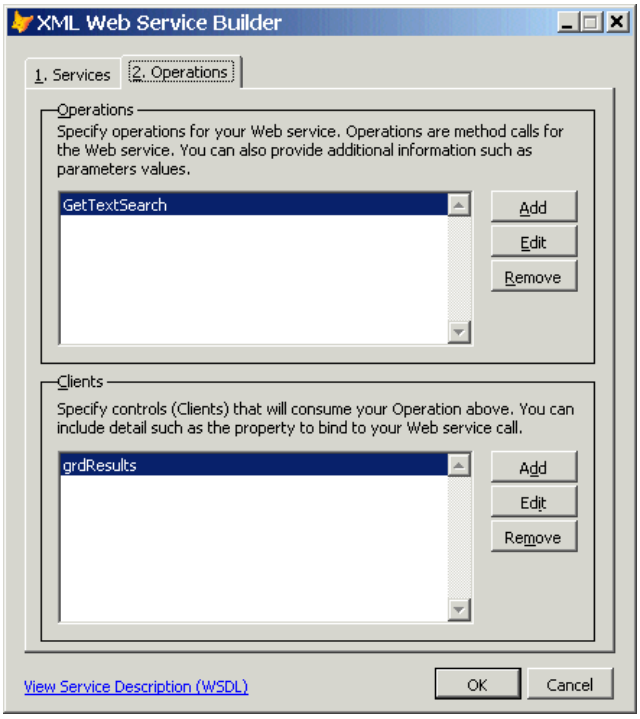
```
local loXMLAdapter as XMLAdapter, ;
   loSchema, ;
   loData, ;
   loTable as XMLTable, ;
   loField as XMLField
loXML = loNWWebService.GetAllCustomers()
*loXML = loNWWebService.GetCustomer('ALFKI')
*loXML = loNWWebService.GetOrdersForCustomer('ALFKI')

* Ensure things are set up the way we want.

close tables
set multilocks on

* Create an XMLAdapter and load the data.

loXMLAdapter = createobject('XMLAdapter')
*** This is one way to do it:
loSchema = loXML.Item(0)
loData   = loXML.Item(1)
loXMLAdapter.Attach(loData, loSchema)
*** Here's another way:
*loXMLAdapter.XMLSchemaLocation = '1'
*loXMLAdapter.LoadXML(loXML.Item(0).parentnode.xml)

* If we succeeded in loading the XML, create and browse a cursor from each
* table object.

if loXMLAdapter.IsLoaded
   for each loTable in loXMLAdapter.Tables
      loTable.ToCursor()
      select (loTable.Alias)
      cursorsetprop('Buffering', 5)
      browse
   next loTable

* Now create a diffgram of changes which we could send back if the Web service
* supported that.

   loXMLAdapter.ReleaseXML(.F.)
   loXMLAdapter.ToXML('test.xml', '', .T., .T., .T.)
   modify file test.xml
   close tables all
endif loXMLAdapter.IsLoaded
```

Using Web Services Visually

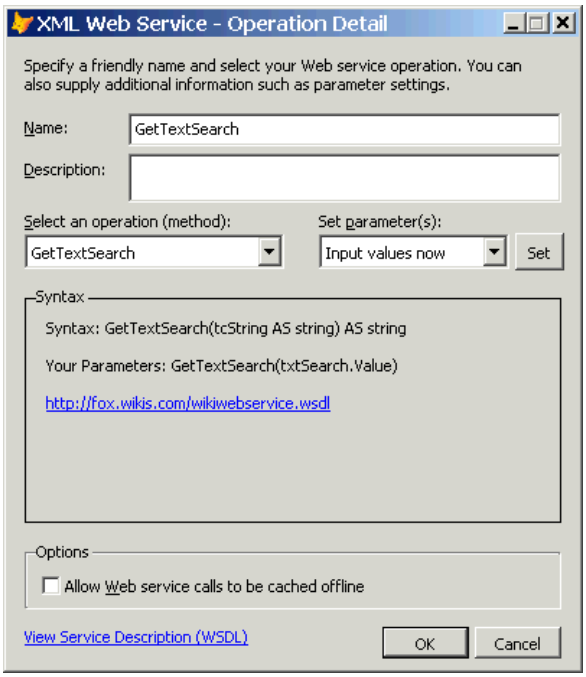
Dragging a Web service from the Toolbox to the Form or Class Designers adds an instance of the same WSHandler class to the container and fills in several properties with information about the selected Web service. The great thing about this class is that VFP provides a builder for it that allows you to configure the object so it not only calls a particular method of the Web service, but also obtains any parameters needed for the method in a variety of ways (such as prompting the user or from controls in the form) and binds the results to other controls.

When you invoke the XML Web Service Builder for the WSHandler object, the Operations page is automatically selected because the Web service has already been defined. (The Services page allows you to define which Web service you're calling.) The Operations page allows you to

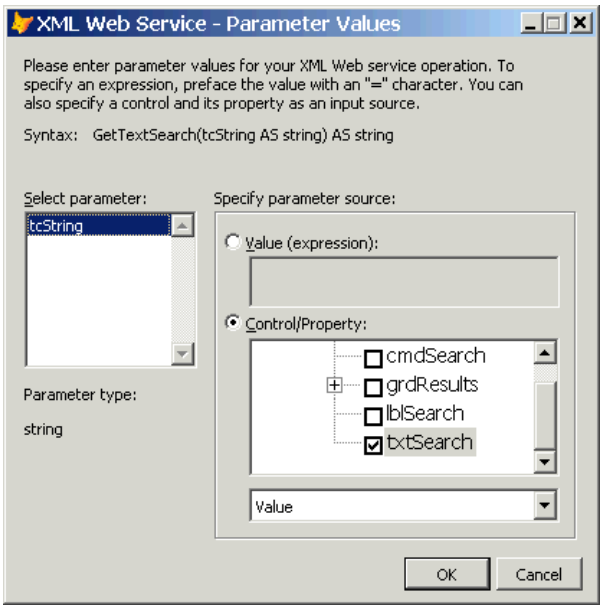
define the operations (which methods of the Web service are called) and which clients (other controls) will consume the results of each operation.



To define an operation, click on the Add button. There are several properties you can set for an operation in the Operation Detail dialog, including a friendly name and description. To select the Web service method to call, choose it from the methods combo box. If you turn on the Allow Web service calls to be cached offline option, WSHandler will cache the results of a Web service call and use those results when the Web service isn't available for a future call.

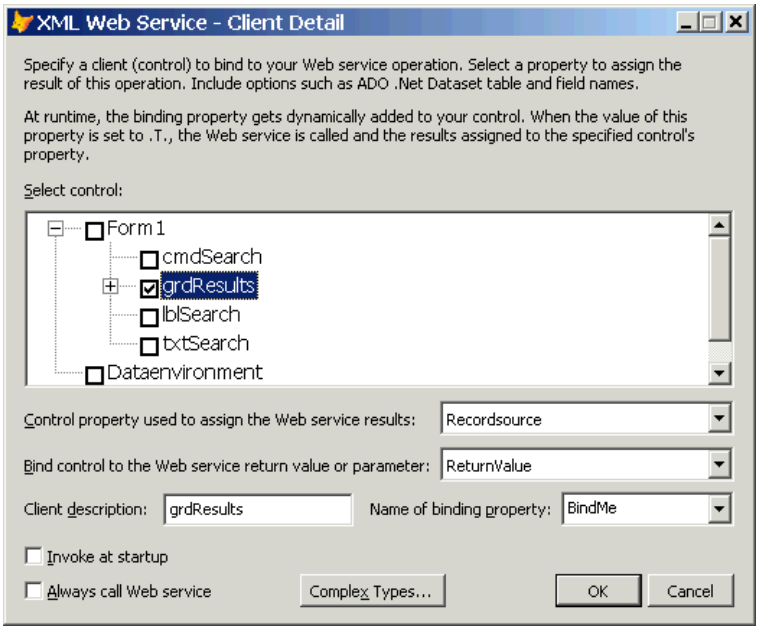


Use the Set parameter(s) combo box to specify how the values for the parameters passed to it are obtained. The choices are Input values now, Programmatically set at run-time, and Prompt at run-time. Programmatically set at run-time means you will set the value in code. Use Prompt at run-time if you want VFP to display a dialog in which the user can enter the values; this probably is more useful for developer tools rather than end-user applications. The term “Input values now” is somewhat misleading, since it doesn’t necessarily mean you enter hard-coded values (although you certainly can do that). When you select that choice, the Set button is enabled. Click on it to display the Parameter Values dialog.



To specify a value for a parameter, select it in the Select parameter list, choose Value (expression) as the parameter type, and enter the value into the edit box below this option. To indicate the value is an expression that should be evaluated at runtime, precede it with "=". To obtain the parameter value from another control, choose Control/Property, put a checkmark in front of the appropriate control, and select the property that holds the value from the combo box below the control list.

To define a client for an operation, click on the Add button beside the Clients list in the Operations page of XML Web Service Builder. Use the Client Detail dialog to specify which control in the container is the client for the operation. You can then indicate which property of the control receives the result, and whether the property is bound to a parameter or the return value of the Web service method.



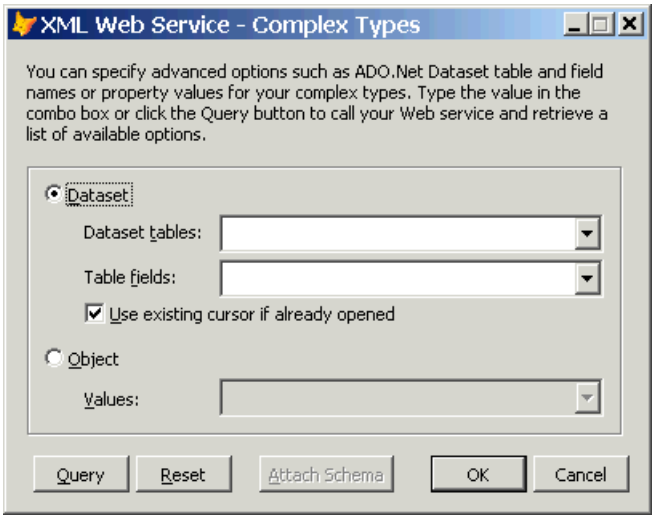
If you turn on the Invoke at startup option, the operation executes when the container instantiates. Otherwise, you'll have to invoke it manually. If the client object is a CursorAdapter and you turn on Invoke at startup, you're warned you should set the form's BindControls property to False. That way there won't be any problem with controls bound to the CursorAdapter. WSHandler automatically sets BindControls to True after the CursorAdapter has been filled with data from the Web service so data binding can occur.

If you turn on the Always call Web service option, the client will always call the Web service method. Otherwise, WSHandler will reuse the results when it detects multiple clients call the same operation at the same time (such as when the container instantiates).

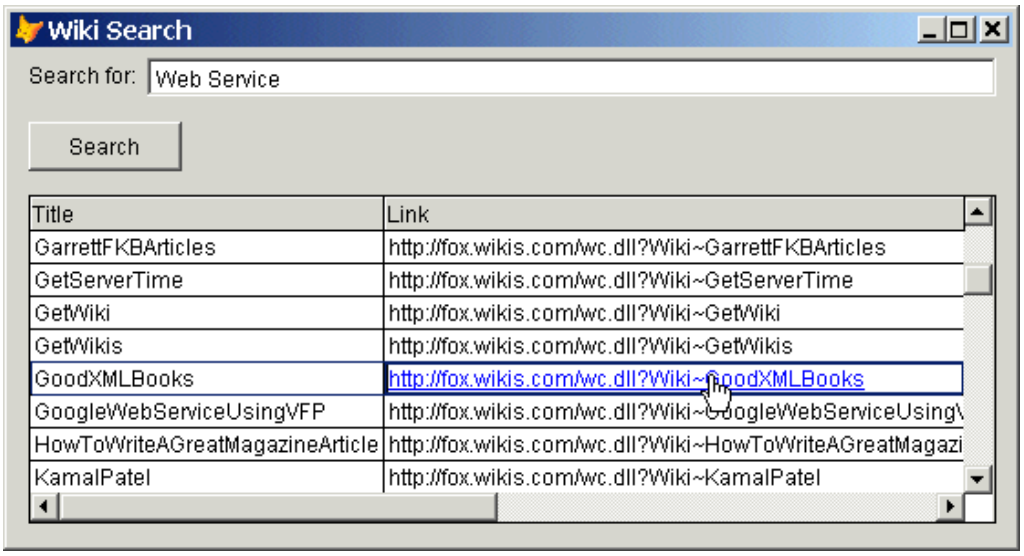
WSHandler adds the property you specify in the Name of binding property option (the default is "BindMe" but you can also choose any custom property or method of the client control) to the client control at runtime if it doesn't already exist. The property acts like a switch: WSHandler uses BINDEVENTS() so it's notified when the property's value changes. Setting its value to .T.

causes the WSHandler object to call the Web service method and put the results into the client control.

If the Web service you want to call returns complex XML, such as objects serialized as XML, ADO.NET DataSets, or VFP cursors converted to XML via CURSORTOXML(), click on the Complex Types button. In the Complex Types dialog, click on the Query button to call the Web service method immediately; the builder will then examine the return value and fill in the rest of the controls. For example, if the Web service returns an ADO.NET DataSet, the Dataset option is selected, the Dataset tables combo box is filled with the names of the tables in the DataSet, and the Table fields combo box shows the fields in the selected table. (In the case of a grid client, Table fields is ignored and the grid is bound to the selected table.) If the Web service returns an XML string that can be converted into a VFP cursor (as is the case with the WikiWebService used in these examples), you'll be informed of that and asked if you want to view the cursor's schema. If you choose Yes, the Dataset tables and Table fields combo boxes are filled in. The Attach button is enabled if the client object is a CursorAdapter and you click on Query to query the Web service. In that case, if you click on the Attach button and choose Yes in the subsequent dialog, the builder will update the CursorSchema property of the CursorAdapter to match the schema of the selected table.



WikiSearch.SCX, included with this document, demonstrates the use of WSHandler. This form uses the FoxWiki Web service to search for a specific string in the FoxWiki. The results are shown in a grid that auto-fits the columns to its contents and provides hyperlinks to the Wiki articles. (Note: instead of running this form directly, run WikiSearch.PRG; it ensures the path to the WSHandler object in WikiSearch.SCX is correct for your system.)



The WSHandler object in the WikiSearch form calls the GetTextSearch method of the FoxWiki Web service, passing it the contents of the Value property of the txtSearch text box, and putting the results into the RecordSource property of the grdResults grid. This operation is not invoked at startup (since we don't know what the search string is), but when you click on the Search button. The Click method of that button has the following code:

```
Thisform.grdResults.BindMe = .T.
Thisform.grdResults.AutoFit()
```

Setting the BindMe property of grdResults to .T. starts the operation because that was the property specified in the builder. When it's finished, the cursor will be created and the grid bound to it (assuming everything worked). The grid's AutoFit method ensures the grid columns are adjusted to fit the received data.

So, we now have a fully functional form that searches the FoxWiki for all articles containing a certain text string and displays the results in a grid, complete with hyperlinks back to the articles. Not bad for just two lines of code!

Publishing Web Services

Publishing a VFP COM DLL as a Web service is easy using the XML Web Services Publisher dialog. Before we do that, let's create a Web service in VFP.

The Web service we'll look at is simplified from a real-world production application. The purpose of the Web service is to allow online activation of an application that must be registered before it can be used (it runs in "demo" mode until it's activated). A registration form in the application will allow the user to enter information about themselves (company name, contact name, etc.) as well as the serial number they were assigned when they purchased a license for the application. Once they have entered the appropriate information, they can click on the Register button to call our Web service to register their license in our database.

The code for our Web service is in Activation.PRG. It consists of a single class called Activation. Activation is based on Session so it hides any properties, events, and methods (PEMs) we don't define, such as the usual VFP PEMs, and it's marked as OLEPUBLIC so it'll be a COM server. Activation has two custom properties: ErrorMessage, which contains the text of any errors that occur, and cDataLocation, a protected property that contains the location of our data files. We likely don't want to store data files in same directory on the Web server as the Web service DLL because that means giving write rights to the browser user. So, we'll use another directory on the server not accessible to the Web. The Init method reads the value for cDataLocation from Activation.INI; using an INI file makes it easy to change the location if necessary.

```
define class Activation as Session olepublic

   ErrorMessage = ''
   protected cDataLocation
   cDataLocation = ''

   protected procedure Init
      local lcDirectory, ;
         lcINIFile
      if version(2) = 2
         lcDirectory = sys(16)
         lcDirectory = addbs(justpath(substr(lcDirectory, ;
            at(' ', lcDirectory, 2) + 1)))
      else
         lcDirectory = addbs(justpath(_vfp.ServerName))
      endif version(2) = 2
      lcINIFile = lcDirectory + 'Activation.ini'
      This.cDataLocation = addbs(ReadINI(lcINIFile, 'Locations', 'Data', ''))
   endproc
```

The Activate method is one of only two public methods (the other checks whether the registered user has a valid subscription to the application). This method is called from the registration form as a Web service to perform the activation. It expects to be passed a password (for security reasons), a serial number, and an XML string containing information about the person registering the application. This method first ensures all the parameters are valid, then ensures the serial number is valid (can be found in our Registrations table) and hasn't already been registered, and then registers the application in the table. If anything goes wrong, COMRETURNERROR() returns error information to the caller. Here's the code: we won't look at the methods called from this one, such as GetRegistrationInfo.

```
   procedure Activate(Password as String, SerialNumber as String, ;
      RegistrationInfo as String) as String
      local lcActivationCode, ;
         llOK
      lcActivationCode = ''
      with This

* Blank the error message.

         .ErrorMessage = ''

* Validate the parameters.

         do case
```

```
            case vartype(Password) <> 'C' or Password <> 'SomePassword'
               .ErrorMessage = 'Invalid password specified.'
            case vartype(SerialNumber) <> 'C' or ;
               empty(SerialNumber)
               .ErrorMessage = 'Invalid serial number specified.'
            case vartype(RegistrationInfo) <> 'C' or ;
               empty(RegistrationInfo)
               .ErrorMessage = 'Invalid registration information ' + ;
                  'specified.'
            case not .GetRegistrationInfo(RegistrationInfo, 'TEMP')
               .ErrorMessage = 'Invalid registration information ' + ;
                  'specified.'
            case empty(COMPANY + CONTACT)
               .ErrorMessage = 'The company and contact cannot both ' + ;
                  'be blank.'
            case empty(ADDRESS1)
               .ErrorMessage = 'The address cannot be blank.'
            case empty(CITY)
               .ErrorMessage = 'The city cannot be blank.'
            case empty(COUNTRY)
               .ErrorMessage = 'The country cannot be blank.'
            case empty(EMAIL)
               .ErrorMessage = 'The email address cannot be blank.'
            otherwise
               llOK = .T.
         endcase
         do case
            case not llOK

* Try to open the registration table.

            case not .OpenData()
               .ErrorMessage = 'Unable to activate at this time.'

* See if the serial number is valid.

            case not .IsSerialValid(SerialNumber)
               .ErrorMessage = SerialNumber + ' is not a valid ' + ;
                  'serial number.'

* See if the serial number has already been activated.

            case .IsActivated(SerialNumber)
               .ErrorMessage = 'Serial number ' + SerialNumber + ;
                  ' has already been activated.'

* Everything is OK, so register this license.

            otherwise
               lcActivationCode = .RegisterLicense(SerialNumber, ;
                  RegistrationInfo)
         endcase

* If we have an error message, return it.

         do case
            case empty(.ErrorMessage)
            case version(2) = 2
               lcActivationCode = .ErrorMessage
            otherwise
               comreturnerror('Activation', .ErrorMessage)
```

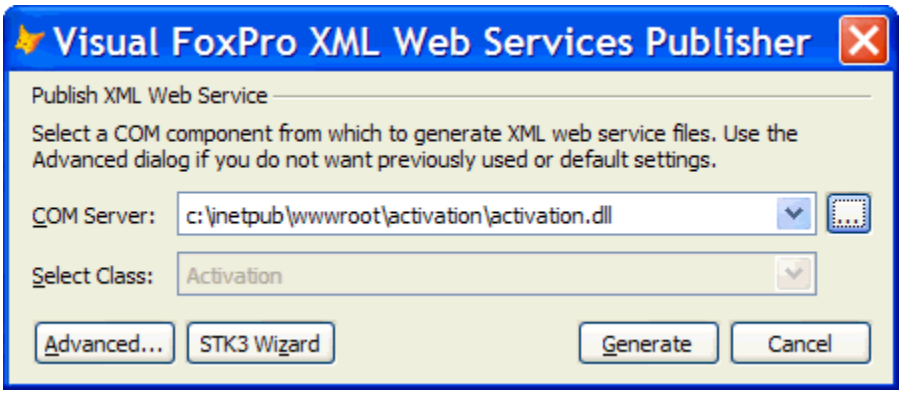
```
            endcase
         endwith

* Clean up, then return.

         close databases all
         return lcActivationCode
      endproc
```

To create a Web service from this class, do the following:

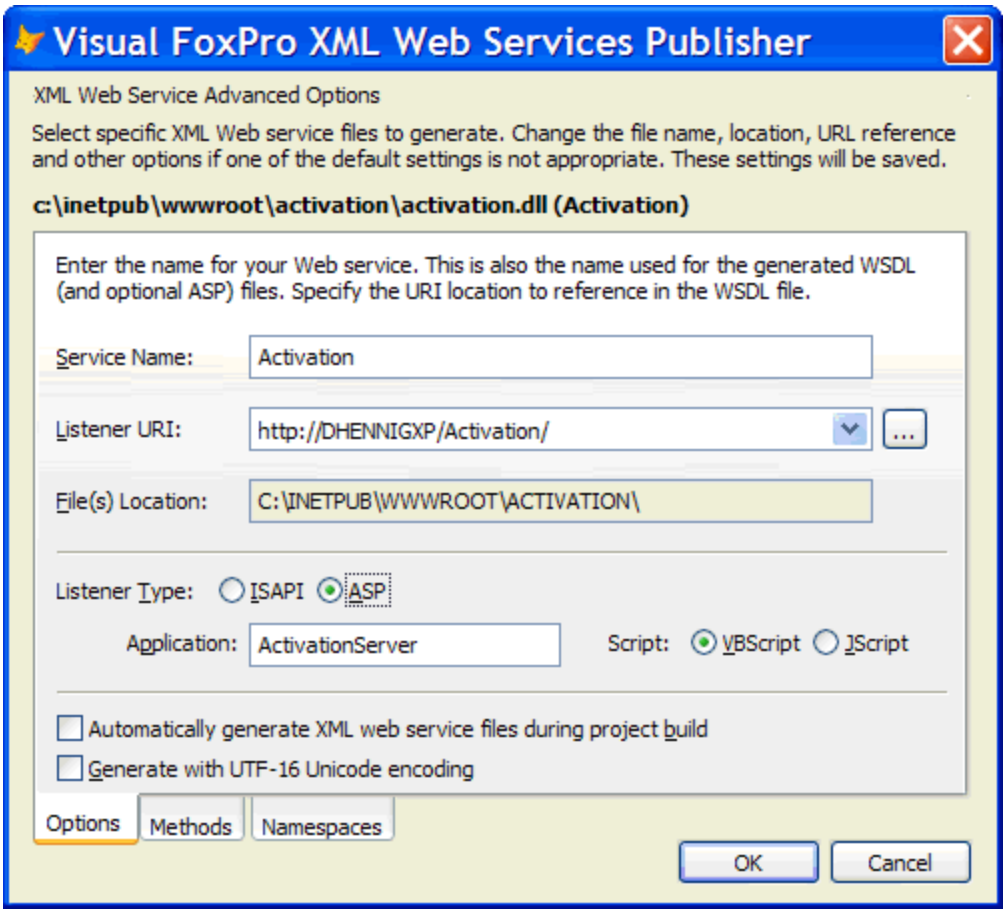
- Create a project called Activation.PJX and add Activation.PRG to it.

- Create a subdirectory of your Web server directory (typically C:\INetPub\WWWRoot) called Activation and build a multi-threaded DLL from the project to that directory. Also, copy Activation.INI to that directory and edit it to point to the location of Registrations.DBF.

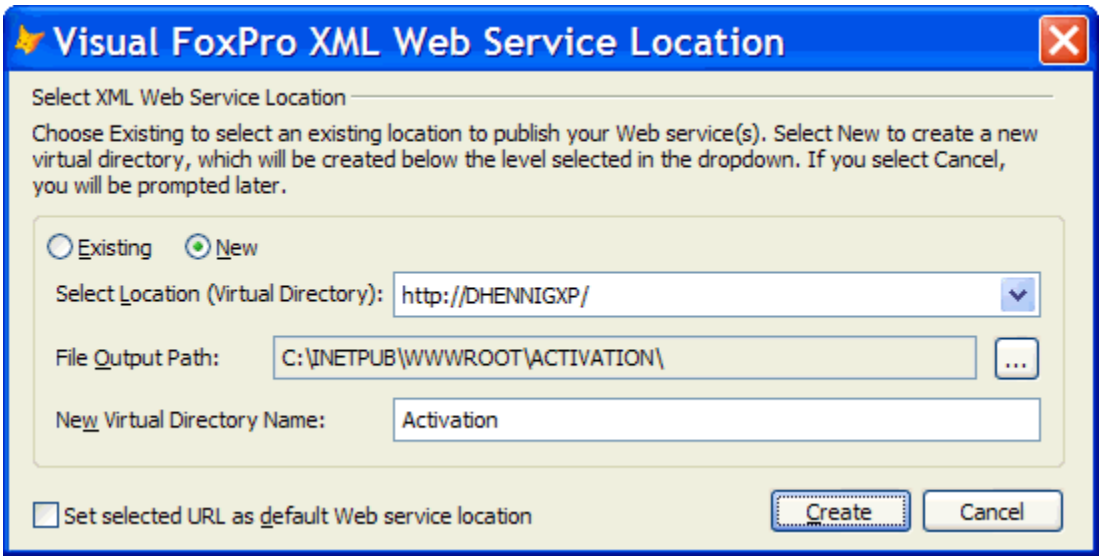
- Launch the VFP XML Web Services Publisher by choosing Web Services from the Tools, Wizards menu or by clicking on the Publish your XML Web Service link in the XML Web Services pane of the Task Pane Manager. (The first time you launch the publisher, you may be presented with a dialog asking you for the default location for your Web service; this dialog is discussed below.) Since you have a project that creates a DLL already open, the COM Server option will automatically be filled in with the name of the DLL, although you'll need to select the one in C:\INetPub\WWWRoot\Activation rather than the default one. If more than one class was defined in the DLL, you'd select it from the Select Class combo box.



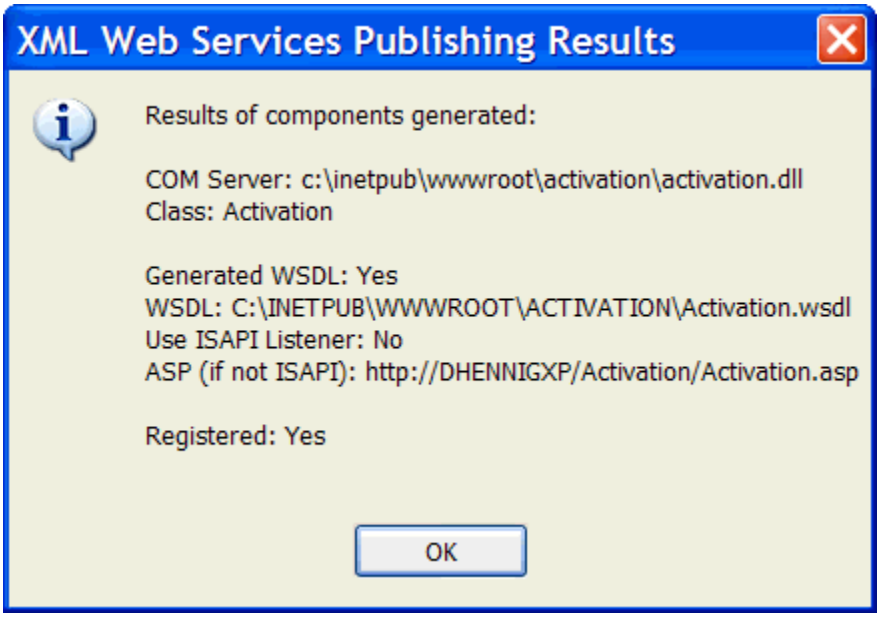
- Click on the Advanced button to display the dialog shown below. This dialog allows you to define things like whether an ISAPI or ASP listener is used (we'll use ASP for this example because it's easier to set up), which script language to use in the case of ASP, which methods are exposed through the Web service, and the namespaces used in the files generated by the Web Services Publisher.



- Click on the button beside the Listener URI combo box to display the XML Web Service Location dialog (this dialog may have appeared earlier if this is the first time you've build a Web service). Choose New and click on the button beside the File Output Path text box to select C:\INetPub\WWWRoot\Activation. Enter "Activation" as the new virtual directory name and click on Create.



- Click on OK in the Advanced dialog, then click on Generate in the XML Web Services Publisher dialog. If everything worked, you should see the following results dialog:



To test the Web service, run Activation.PRG. Although it contains the class definition for our Web service, it also has some test code at the start. The following code instantiates the Web service class. With #IF .T., it'll instantiate the Activation class as a VFP object (not even a COM object), which works better for debugging. Change this statement to #IF .F. to test it as a Web service.

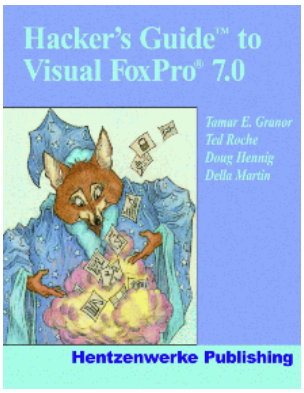
```
#if .T.
   loActivation = createobject('Activation')
#else
   lcWSDLLocation = 'http://localhost/Activation/Activation.wsdl'
   local loActivation as MSSOAP.SoapClient
   loActivation = createobject('MSSOAP.SoapClient')
   loActivation.MSSoapInit(lcWSDLLocation, 'Activation', 'ActivationSoapPort')
#endif
```

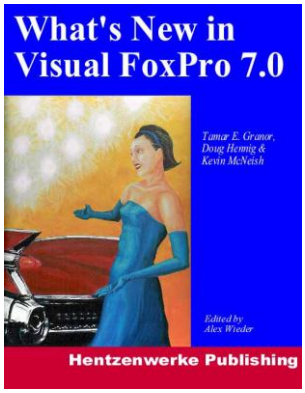
Summary

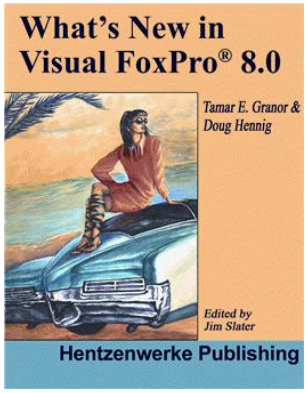
XML Web services are easier to use than ever in VFP 8. The XML Web Services pane of the Task Pane manager makes it easy to register and explore XML Web services. The WSHandler class allows you to consume XML Web services both programmatically and visually, including handling the wiring of form controls to a Web service. Publishing an XML Web service from a VFP DLL is also easily done using the XML Web Services Publisher. Start considering what role XML Web services will take in your distributed applications.

Biography

Doug Hennig is a partner with Stonefield Systems Group Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT), the award-winning Stonefield Query, and the CursorAdapter and DataEnvironment builders that come with Microsoft Visual FoxPro 8. Doug is co-author of "What's New in Visual FoxPro 8.0", "The Hacker's Guide to Visual FoxPro 7.0", and "What's New in Visual FoxPro 7.0". He was the technical editor of "The Hacker's Guide to Visual FoxPro 6.0" and "The Fundamentals". All of these books are from Hentzenwerke Publishing (http://www.hentzenwerke.com). Doug writes the monthly "Reusable Tools" column in FoxTalk. He has spoken at every Microsoft FoxPro Developers Conference (DevCon) since 1997 and at user groups and developer conferences all over North America. He is a Microsoft Most Valuable Professional (MVP) and Certified Professional (MCP).







Doug Hennig
Partner
Stonefield Systems Group Inc.
1112 Winnipeg Street, Suite 200
Regina, SK Canada S4R 1J6
Phone: (306) 586-3341 Fax: (306) 586-5080
Email: dhennig@stonefield.com
Web: www.stonefield.com
Web: www.stonefieldquery.com

Appendix. Listing of ASP.NET NWWebService

```
Imports System.Web.Services
Imports System.Data
Imports System.Data.SqlClient

<WebService(Namespace:="http://tempuri.org/")> _
Public Class NWWebService
  Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

  <WebMethod()> Public Function GetAllCustomers() As DataSet
    Dim loConn As SqlConnection = GetConnection()
    Dim loDataAdapter As New SqlDataAdapter("select * from Customers", loConn)
    loDataAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey
    Dim loDataSet As New DataSet()
    loDataAdapter.Fill(loDataSet)
    Return loDataSet
  End Function

  <WebMethod()> Public Function GetCustomer(ByVal CustomerID As String) _
    As DataSet
    Dim loConn As SqlConnection = GetConnection()
    Dim loCommand As New SqlCommand("select * from Customers " & _
      "where CustomerID=@CustomerID", loConn)
    loCommand.Parameters.Add("@CustomerID", CustomerID)
    Dim loDataAdapter As New SqlDataAdapter(loCommand)
    loDataAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey
    Dim loDataSet As New DataSet()
    loDataAdapter.Fill(loDataSet)
    Return loDataSet
  End Function

  <WebMethod()> Public Function GetOrdersForCustomer(ByVal CustomerID _
    As String) As DataSet
    Dim loConn As SqlConnection = GetConnection()
    Dim loCommand As New SqlCommand("select * from Orders " & _
      "where CustomerID=@CustomerID", loConn)
    loCommand.Parameters.Add("@CustomerID", CustomerID)
    Dim loDataAdapter As New SqlDataAdapter(loCommand)
    loDataAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey
    Dim loDataSet As New DataSet()
    loDataAdapter.Fill(loDataSet)
    Return loDataSet
  End Function

  Protected Function GetConnection() As SqlConnection
    Dim lcConnString As String = "Data Source=localhost;" & _
      "Initial Catalog=Northwind;User ID=sa;Password="
    Dim loConn As New SqlConnection(lcConnString)
    loConn.Open()
    Return loConn
  End Function
End Class
```