CHAPTER 16

HTTP Endpoints

SQL Server 2000 offered SOAP (Simple Object Access Protocol) web service support via SQLXML. The SQL Server 2000 model relied on a loose coupling of SQL Server and Internet Information Services (IIS), making setup and configuration a bit of a hassle. The end result was that a lot of developers found that creating .NET web services in the middle tier was a far more flexible (and easier to use) solution than the SQL Server 2000/SQLXML/IIS model.

SQL Server 2005 provides native HTTP endpoints to expose your stored procedures as web methods directly from SQL Server via the SOAP protocol. HTTP endpoints are easy to create and administer, and the SQL Server HTTP endpoints model provides tight security integration with SQL Server. HTTP endpoints raise the level of SQL Server functionality from *just* a database management system to a full-fledged application server.

This chapter discusses HTTP endpoint setup and configuration and provides samples to demonstrate how to use HTTP endpoints from your .NET applications.

What Are HTTP Endpoints?

The W3C defines an endpoint as the following:

> *An association between a binding and a network address, specified by a URI, that may be used to communicate with an instance of a service. An endpoint indicates a specific location for accessing a service using a specific protocol and data format.* (http://www.w3.org/TR/ws-gloss/#defs)

SQL Server provides built-in support for exposing endpoints over the HTTP protocol, allowing you to access SQL Server stored procedures as SOAP-based web service methods.

Note SQL Server HTTP endpoints are supported on Windows Server 2003 and Windows XP Service Pack 2. HTTP endpoints are not supported on SQL Server 2005 Express Edition.

The CREATE ENDPOINT Statement

As the keywords suggest, SQL Server 2005's `CREATE ENDPOINT` statement allows you to create endpoints. The following is the format for creating an HTTP endpoint that exposes your stored procedures and user-defined functions as SOAP web service methods:

```
CREATE ENDPOINT end_point_name
[ AUTHORIZATION login ]
[ STATE = { STARTED | STOPPED | DISABLED } ]
AS HTTP (
    PATH = 'url' ,
    AUTHENTICATION = ( { BASIC | DIGEST | INTEGRATED | NTLM | KERBEROS }
        [ , ... n ] ) ,
    PORTS = ( { CLEAR | SSL } [ , ... n ] )
    [ , SITE = { '*' | '+' | 'web_site' } ]
    [ , CLEAR_PORT = clear_port ]
    [ , SSL_PORT = ssl_port ]
    [ , AUTH_REALM = { 'realm' | NONE } ]
    [ , DEFAULT_LOGON_DOMAIN = { 'domain' | NONE } ]
    [ , COMPRESSION = { ENABLED | DISABLED } ]
)
FOR SOAP (
    [ { WEBMETHOD [ 'namespace' . ] 'method_alias'
        (
            NAME = 'database.schema.proc_name'
            [ , SCHEMA = { NONE | STANDARD | DEFAULT } ]
            [ , FORMAT = { ALL_RESULTS | ROWSETS_ONLY | NONE } ]
        )
      } [ , ... n ] ]
    [ , BATCHES = { ENABLED | DISABLED } ]
    [ , WSDL = { NONE | DEFAULT | 'sp_name' } ]
    [ , SESSIONS = { ENABLED | DISABLED } ]
    [ , LOGIN_TYPE = { MIXED | WINDOWS } ]
    [ , SESSION_TIMEOUT = timeout_interval | NEVER ]
    [ , DATABASE = { 'database_name' | DEFAULT } ]
    [ , NAMESPACE = { 'namespace' | DEFAULT } ]
    [ , SCHEMA = { NONE | STANDARD } ]
    [ , CHARACTER_SET = { SQL | XML } ]
    [ , HEADER_LIMIT = header_limit ]
);
```

■**Note** The `CREATE ENDPOINT` statement also supports creating TCP endpoints for Service Broker and database mirroring applications. I focus on HTTP SOAP endpoint creation in this chapter, but the full syntax for the `CREATE ENDPOINT` statement is available in BOL at `http://msdn2.microsoft.com/en-us/library/ms181591.aspx`. The full syntax includes all TCP, TSQL, Service Broker, and database mirroring options.

CREATE ENDPOINT Arguments

The base endpoint arguments for the `CREATE ENDPOINT` statement are the same regardless of the chosen protocol and application. The `CREATE ENDPOINT` arguments for HTTP SOAP endpoints are listed here:

- *`end_point_name`* is the name of the endpoint. This name is used to reference and manage the endpoint and must be a valid T-SQL identifier.
- *`login`* is the SQL Server or Windows login that is assigned ownership of the endpoint at creation time. The default, if the `AUTHORIZATION` clause is left off, is the caller. If *`login`* is not the same as the caller, the caller must have `IMPERSONATE` permission on the specified login.
- The `STATE` argument specifies the state of the endpoint at creation time. `STOPPED` is the default. The following are the valid states:
  - `STARTED`: The endpoint is started and begins listening for connections at creation time.
  - `STOPPED`: The endpoint is stopped. The server listens for connections on the endpoint when it is `STOPPED` but returns errors to the client when it receives requests.
  - `DISABLED`: The server does not listen for connections on the endpoint, and it doesn't respond to requests on the endpoint.

HTTP Protocol Arguments

The HTTP protocol arguments are used in the `AS HTTP` clause to specify HTTP as the transport protocol. These arguments include the following:

- `url` is a relative uniform resource locator (URL) specification to help SQL Server route HTTP SOAP requests appropriately. For the absolute URL `http://www.apress.com/SQLWebService1`, the portion specified by `PATH = 'url'` would be `PATH = '/SQLWebService1'`. The portion specified by the `SITE` option would be `www.apress.com`.

- The `AUTHENTICATION` option specifies how the server should authenticate clients. You can specify one or more of the following authentication types:

  - `BASIC` authentication is specified by the Internet Engineering Task Force (IETF) HTTP 1.1 specification (RFC 2617, `http://www.ietf.org/rfc/rfc2617.txt`). `BASIC` authentication is performed via a header containing the Base-64-encoded username and password. SQL Server requires that the `PORTS` value be set to `SSL`, that the username and password be mapped to a valid Windows login, and that a Secure Sockets Layer port be used for the connection if `BASIC` authentication is specified.

  - `DIGEST` authentication is the second form of authentication specified by the IETF HTTP 1.1 specification. `DIGEST` authentication hashes the username and password using the MD5 one-way hash algorithm before sending it to the server. The server then compares the hashed credentials sent to it with a hash of the same credentials stored locally. Windows-based `DIGEST` authentication is only supported over domain controllers that are running under Windows Server 2003, and local user accounts cannot be authenticated using this method. Only valid Windows domain accounts can be authenticated using this method.

  - `NTLM`, also known as Windows NT Challenge/Response authentication, is the authentication method supported by Windows 95, Windows 98, and Windows NT 4.0 (Server and Workstation). `NTLM` is a connection-based protocol that is more secure than either `BASIC` or `DIGEST`. Windows 2000 and later provide `NTLM` authentication by means of a Security Support Provider Interface (SSPI).

  - `KERBEROS` authentication is an Internet-standard authentication protocol developed at the Massachusetts Institute of Technology (`http://web.mit.edu/kerberos/`). `KERBEROS` authentication support is provided in Windows 2000 and later by means of an SSPI. When `KERBEROS` authentication is specified, a Service Principal Name (SPN) must be associated with the account it will be running on. More information on this is available in BOL at `http://msdn2.microsoft.com/en-us/library/ms178119.aspx`.

    - `INTEGRATED` authentication allows the client to request either `KERBEROS` or `NTLM` authentication. If the authentication type specified by the client does not succeed, the server terminates the connection. The server will not fall back and attempt to reauthenticate using the other allowable `INTEGRATED` authentication method.

  More than one authentication method can be specified in a comma-delimited list. If more than one authentication method is specified, the method specified by the client is used.

- The `PORTS` option specifies the type of listening port the endpoint will use. The following are valid values:

    - `CLEAR` port type specifies incoming requests must use the HTTP (`http://`) protocol.

    - `SSL` port type specifies incoming requests must use the Secure HTTP (`https://`) protocol.

  You may specify both port types for a single endpoint.

- The `SITE` argument specifies the name of the host computer. The default is the asterisk (*). The following are valid values:

    - The asterisk (*) indicates that the endpoint should listen for all possible hostnames for the computer that are not explicitly reserved. Namespaces may be explicitly reserved using the `sp_reserve_http_namespace` stored procedure. This procedure is described in BOL at `http://msdn2.microsoft.com/en-us/library/ms190614.aspx`.

    - The plus sign (+) indicates that the endpoint should listen for all possible hostnames for the computer.

    - An explicit *`web_site`* name indicates the endpoint should listen for the specified hostname for the computer.

- The `CLEAR_PORT` option is used when the `PORTS = (CLEAR)` option has been specified. The *`clear_port`* value is the port number to be used for HTTP communication. The default is the standard HTTP port number 80.

- The `SSL_PORT` option is used when the `PORTS = (SSL)` option has been specified. The *`ssl_port`* value is the port number to be used for Secure HTTP communication. The default is the standard Secure HTTP port number 443.

- The AUTH_REALM option specifies a hint returned to the client (as part of the HTTP challenge) when AUTHENTICATION = (DIGEST) has been specified. The default is NONE.
- The DEFAULT_LOGON_DOMAIN specifies the default domain to log on when AUTHENTICATION = (BASIC) is specified. The default is NONE.
- The COMPRESSION option tells SQL Server to return a GZip-compressed response when a request specifies GZip compression in its headers. The default is DISABLED.

SOAP Arguments

The SOAP arguments are used in the FOR SOAP clause and are specific to the SOAP protocol configuration for the endpoint. The following are valid arguments:

- WEBMETHOD specifies an alias for a web method exposed via SOAP. The *method_alias* can be preceded by an optional namespace. Note that WEBMETHOD is an optional argument and you can declare an HTTP SOAP endpoint with no web methods. The WEBMETHOD argument has its own list of subarguments:
  - Every WEBMETHOD needs a NAME argument to specify the three-part name of a stored procedure or user-defined function to implement the SOAP web method. Note that the *database* and *schema* parts of the stored procedure or user-defined function name are mandatory.
  - SCHEMA is an optional argument that specifies whether an inline XSD schema is returned for the current web method in the SOAP response. The following are valid values:
    - NONE specifies no inline XSD schema is returned.
    - STANDARD specifies an inline XSD schema is returned.
    - DEFAULT specifies the endpoint SCHEMA setting should be used.
  - FORMAT specifies how results should be returned to the client. The default is ALL_RESULTS. The following are valid FORMAT values:

  - ALL_RESULTS specifies that a result set, row count, warnings, and error messages will all be returned to the client as an array of .NET System.Objects.

  - ROWSETS_ONLY specifies that only result sets are returned to the client. This option should be used when you want to return a .NET System.Data.DataSet object instead of an Object array.

  - NONE suppresses SOAP-specific markup in the response. In this mode, the application is responsible for generating well-formed raw XML. The NONE option has several restrictions and limitations, all of which are described in detail in BOL at http://msdn2.microsoft.com/en-us/library/ms181591.aspx.

- The BATCHES argument enables or disables ad hoc SQL queries on the endpoint via the sql:sqlbatch web method. The sql:sqlbatch method allows parameterized queries, which I will discuss in the section "Executing HTTP Endpoint Ad Hoc Queries." The default is DISABLED.

- The WSDL argument specifies whether the endpoint can generate WSDL documents. The default value of DEFAULT specifies that a WSDL response is generated for WSDL queries to this endpoint. A value of NONE specifies that no WSDL response is generated. Under specific circumstances you can specify a stored procedure (*sp_name*) to generate a modified WSDL document if you need custom WSDL support.

- SESSIONS can be either ENABLED or DISABLED. When enabled, the endpoint can treat multiple SOAP request/response pairs as a single SOAP session. The default is DISABLED.

- LOGIN_TYPE is the SQL Server authentication type for the endpoint and can be either MIXED or WINDOWS. The default is WINDOWS. When MIXED is used, the endpoint must be configured to use SSL.

- SESSION_TIMEOUT specifies an integer time-out value (in seconds) before a SOAP session expires at the server. The default *timeout_interval* is 60 seconds. A value of NEVER indicates SOAP sessions should never time out.

- The DATABASE argument specifies the database context under which the SOAP method should execute. The default is the default database for the login.

- NAMESPACE specifies a namespace for the endpoint. The default namespace is http://tempuri.org. The optional namespace, if included in the WEBMETHOD declaration, overrides this NAMESPACE declaration.

- The `SCHEMA` argument specifies whether an inline XSD schema should be included in the SOAP responses returned to the client. If set to `NONE`, no inline XSD schema is included; if set to `STANDARD`, an inline XSD schema is included in responses. The value specified for `SCHEMA` in the `WEBMETHOD` argument overrides this setting on a per-method basis, unless the `WEBMETHOD` specifies `DEFAULT`, in which case this endpointwide setting is used. The `STANDARD` setting is required to map SOAP results to a .NET `System.Data.DataSet`, and the default is `STANDARD`.

- `CHARACTER_SET` can specify either the `SQL` or `XML` character set. If `SQL` is specified, characters that are not valid character references are encoded and returned in the result. If `XML` is specified, characters are encoded according to the XML specification. The default is `XML`.

- The `HEADER_LIMIT` specifies the maximum size of the header section in bytes. The default is 8,192 bytes. If the header section is larger than this limit, the server will generate an error.

Creating an HTTP Endpoint

Before we create our first HTTP endpoint, we need to create stored procedures and user-defined functions that implement the web methods we want to expose. We will create three methods: two stored procedures, and a scalar user-defined function. We will expose all three as web methods.

■ **Note** Technically you don't have to create the stored procedures and/or user-defined functions first. You can create the endpoint and add methods later, if you prefer, with the `ALTER ENDPOINT` statement. However, for our purposes, creating the stored procedures and user-defined functions is as good a place to start as any.

For this demonstration we'll create a stored procedure named `Sales.GetSalespersonList`. This stored procedure will retrieve a list of all AdventureWorks salespeople's names and their ID numbers. Listing 16-1 is the `Sales.GetSalespersonList` procedure.

Listing 16-1. *Sales.GetSalespersonList Stored Procedure*

```
USE AdventureWorks;
GO
CREATE PROCEDURE Sales.GetSalespersonList
AS
BEGIN
    SELECT s.SalesPersonID,
        s.LastName + ', ' + s.FirstName + ' ' +
            COALESCE(s.MiddleName, '') AS FullName
    FROM Sales.vSalesPerson s
    ORDER BY s.LastName, s.FirstName, s.MiddleName;
END;
GO
```

While the `Sales.GetSalespersonList` stored procedure does not accept any parameters, the second method will. For the second method you'll define another stored procedure that accepts a single salesperson ID number as a parameter and returns a summary listing of that salesperson's sales. Listing 16-2 is the `Sales.GetSalespersonSales` procedure listing.

Listing 16-2. *Sales.GetSalespersonSales Procedure*

```
USE AdventureWorks;
GO
CREATE PROCEDURE Sales.GetSalespersonSales (@SalespersonID INT)
AS
BEGIN
    SELECT soh.SalesOrderID,
        soh.CustomerID,
        soh.OrderDate,
        soh.SubTotal
    FROM Sales.SalesOrderHeader soh
    WHERE soh.SalesPersonID = @SalespersonID
    ORDER BY soh.SalesOrderID;
END;
GO
```

The third method is a scalar user-defined function that also accepts an AdventureWorks salesperson's ID number and returns the total dollar amount of sales for that salesperson. Listing 16-3 is the listing for the `Sales.GetSalesTotal` UDF.

Listing 16-3. *Sales.GetSalesTotal Scalar User-Defined Function*

```
USE AdventureWorks;
GO
CREATE FUNCTION Sales.GetSalesTotal(@SalespersonID INT)
RETURNS MONEY
AS
BEGIN
    RETURN (
        SELECT SUM(soh.SubTotal)
        FROM Sales.SalesOrderHeader soh
        WHERE SalesPersonID = @SalespersonID
    );
END;
GO
```

■**Note** SQL Server HTTP endpoints support exposing stored procedures and scalar UDFs as web methods. Table-valued functions and extended stored procedures, however, cannot be exposed as web methods.

Now that three methods have been implemented, it's time to turn them into SOAP web methods with the `CREATE ENDPOINT` statement. The `CREATE ENDPOINT` statement I'll use to expose these three methods is shown in Listing 16-4.

Listing 16-4. *CREATE ENDPOINT Statement*

```
USE AdventureWorks;
GO
CREATE ENDPOINT AdvSalesEndpoint
    STATE = STARTED
AS HTTP
(
    PATH = N'/AdvSalesSql',
    AUTHENTICATION = (INTEGRATED),
    PORTS = (CLEAR),
    SITE = N'*'
)
```

```
FOR SOAP
(
    WEBMETHOD N'GetSalespersonList'
    (
        NAME = N'AdventureWorks.Sales.GetSalespersonList',
        FORMAT = ROWSETS_ONLY
    ),
    WEBMETHOD N'GetSalesPersonSales'
    (
        NAME = N'AdventureWorks.Sales.GetSalesPersonSales',
        FORMAT = ROWSETS_ONLY
    ),
    WEBMETHOD 'GetSalesTotal'
    (
        NAME = N'AdventureWorks.Sales.GetSalesTotal'
    ),
    WSDL = DEFAULT,
    DATABASE = N'AdventureWorks',
    SCHEMA = STANDARD
);
GO
```

The endpoint definition begins by defining the alias `AdvSalesEndpoint` and setting the endpoint state to `STARTED`:

```
CREATE ENDPOINT AdvSalesEndpoint
    STATE = STARTED
```

The `AS HTTP` clause declares HTTP as the endpoint transport protocol. The HTTP arguments define the `PATH` as `/AdvSalesSql` and set the `AUTHENTICATION` type to `INTEGRATED`. Also the `PORTS` are set to `CLEAR` and the `SITE` is set to *. HTTP port number 80 is used by default:

```
AS HTTP
(
    PATH = N'/AdvWorksSql',
    AUTHENTICATION = (INTEGRATED),
    PORTS = (CLEAR),
    SITE = N'*'
)
```

Tip You may run into problems using the default port if you are running IIS and SQL Server 2005 on the same computer. IIS has a habit of intercepting web requests directed at the server before SQL Server can get to them. If you do run into these problems, you can turn IIS off or use a different port number for your HTTP Endpoint. In the previous example you could change the HTTP endpoint port number by using the HTTP `CLEAR_PORT` argument.

The `FOR SOAP` clause is where the web methods are mapped to the stored procedures and user-defined functions via the `WEBMETHOD` arguments:

```
FOR SOAP
(
    WEBMETHOD N'GetSalespersonList'
    (
        NAME = N'AdventureWorks.Sales.GetSalespersonList',
        FORMAT = ROWSETS_ONLY
    ),
    WEBMETHOD N'GetSalesPersonSales'
    (
        NAME = N'AdventureWorks.Sales.GetSalesPersonSales',
        FORMAT = ROWSETS_ONLY
    ),
    WEBMETHOD N'GetSalesTotal'
    (
        NAME = N'AdventureWorks.Sales.GetSalesTotal'
    ),
```

The first `WEBMETHOD` argument maps a web method named `GetSalespersonList` to the `AdventureWorks.Sales.GetSalespersonList` stored procedure. The second and third `WEBMETHOD` arguments map web methods named `GetSalesPersonSales` and `GetSalesTotal` to their respective procedures and user-defined functions. For the two stored procedures, the `FORMAT` is defined as `ROWSETS_ONLY`, which allows you to retrieve the results as .NET `System.Data.DataSets`. The remaining arguments set endpointwide settings. The `WSDL` argument specifies that the endpoint can generate default WSDL documents. The `DATABASE` argument sets the default database context to the AdventureWorks database, and the `SCHEMA` argument is set to `STANDARD` so that inline XSD schemas will be included in the SOAP responses:

```
    WSDL = DEFAULT,
    DATABASE = N'AdventureWorks',
    SCHEMA = STANDARD
);
GO
```

You can use the SSMS Object Explorer to verify that the HTTP endpoint was created. It will be listed in the Endpoints folder under Server Objects. The newly created HTTP endpoint is shown in Figure 16-1.

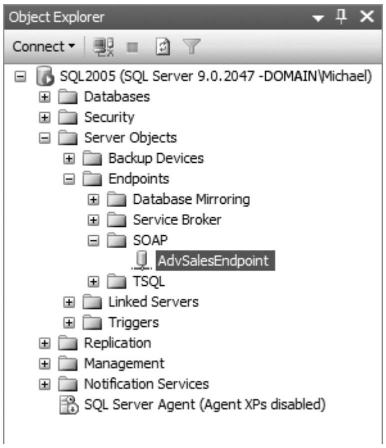


Figure 16-1. *HTTP endpoint in SSMS Object Explorer*

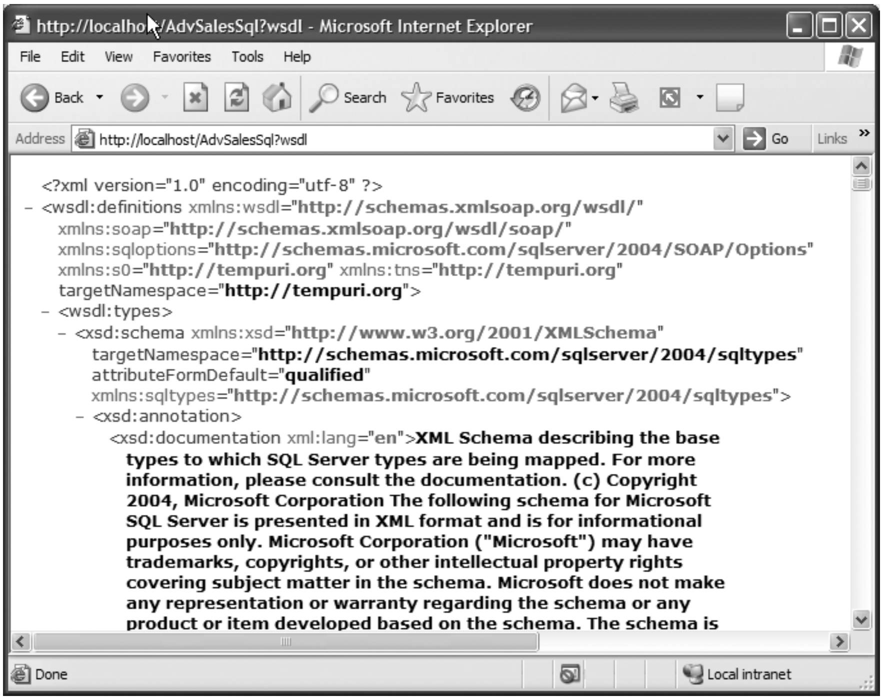
WSDL Documents

The W3C WSDL standard describes an "XML language for describing Web services" (`http://www.w3.org/TR/2006/CR-wsdl20-20060327/`). SQL Server HTTP endpoints can generate two types of WSDL documents for your web services: a standard WSDL document or a simple WSDL document. The `CREATE ENDPOINT` statement in Listing 16-4 sets the `WSDL` argument to `DEFAULT`, so the endpoint can generate WSDL documents as required. The WSDL documents can be viewed by pointing Internet Explorer at the endpoint URL with a `?wsdl` or a `?wsdlsimple` parameter like this:

```
http://localhost/AdvSalesSql?wsdl
http://localhost/AdvSalesSql?wsdlsimple
```

The full URL includes the website name, the port number (if ports other than the defaults are used), the relative URL path, and a `?wsdl` or `?wsdlsimple` parameter. The `?wsdl` parameter tells SQL Server to generate a fully decorated XSD schema with complex types, while `?wsdlsimple` uses standard simple XSD data types.

Figure 16-2 shows a portion of the WSDL document generated by the sample HTTP endpoint.


http://localhost/AdvSalesSql?wsdl - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/AdvSalesSql?wsdl
Go Links
Done
Local intranet

```
  <?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:sqloptions="http://schemas.microsoft.com/sqlserver/2004/SOAP/Options"
    xmlns:s0="http://tempuri.org" xmlns:tns="http://tempuri.org"
    targetNamespace="http://tempuri.org">
  - <wsdl:types>
    - <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://schemas.microsoft.com/sqlserver/2004/sqltypes"
        attributeFormDefault="qualified"
        xmlns:sqltypes="http://schemas.microsoft.com/sqlserver/2004/sqltypes">
      - <xsd:annotation>
          <xsd:documentation xml:lang="en">XML Schema describing the base
            types to which SQL Server types are being mapped. For more
            information, please consult the documentation. (c) Copyright
            2004, Microsoft Corporation The following schema for Microsoft
            SQL Server is presented in XML format and is for informational
            purposes only. Microsoft Corporation ("Microsoft") may have
            trademarks, copyrights, or other intellectual property rights
            covering subject matter in the schema. Microsoft does not make
            any representation or warranty regarding the schema or any
            product or item developed based on the schema. The schema is
```

Figure 16-2. *Sample WSDL document*

Tip Internet Explorer is a handy way to view WSDL documents for your HTTP endpoints, but they can be retrieved by other applications using standard HTTP GET requests.

WSDL document details are available from the W3C Web Services Description Working Group home page at `http://www.w3.org/2002/ws/desc/`.

Creating a Web Service Consumer

Now that you have an HTTP SOAP endpoint with exposed methods configured, it's time to create a web service consumer. Here you'll create a simple web services client with Visual Basic. The first step is to create a new Windows Application project in Visual Studio. Then you'll drag a `DataGridView`, a `ComboBox`, a `TextBox`, and a `Button` control onto the form, as shown in Figure 16-3.

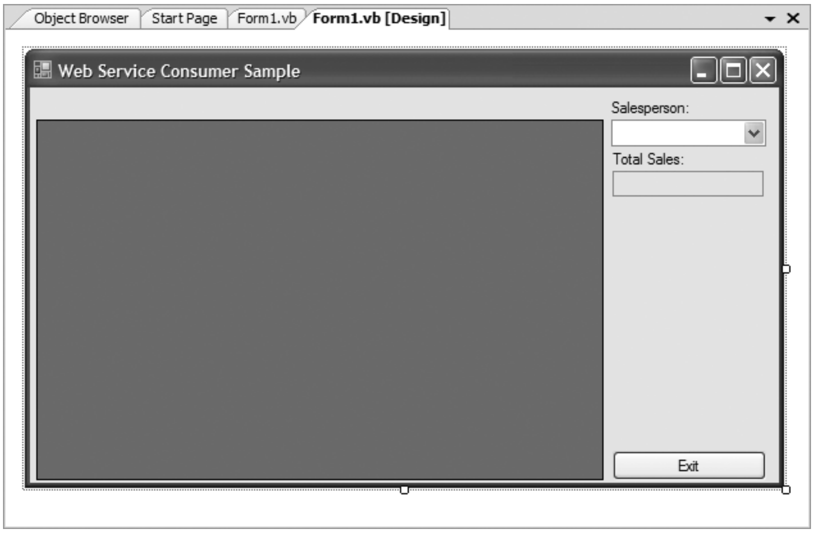
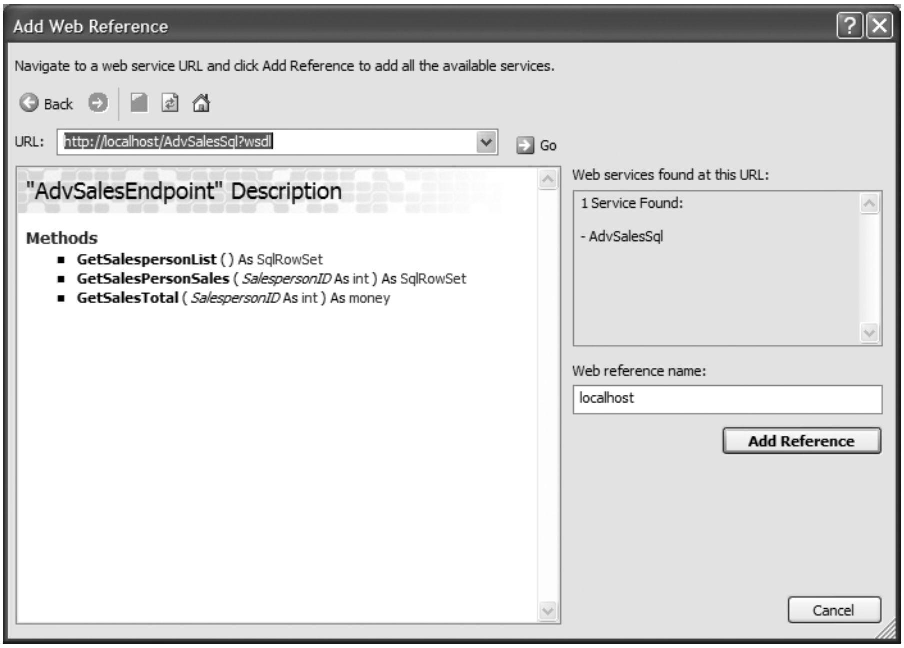


Figure 16-3. *Web services consumer form design*

The next step is to add a web reference to the project. Right-click in the Visual Studio Solution Explorer and choose Add Web Reference from the pop-up menu. The Add Web Reference window then requires you to type in the URL to retrieve the WSDL document from the endpoint and press the Go button. In this instance the URL is `http://localhost/AdvSalesSql?wsdl`. The three web methods exposed by the HTTP endpoint will be displayed. Next, click the Add Reference button to add the web reference. Figure 16-4 shows the Add Web Reference window.

After the Windows form is set up and the web reference has been added to the project, it's a simple matter to add the code to reference the web methods from your Visual Basic code. There are four basic steps that need to be performed to use a web method:

1. Create a web service proxy.
2. Set the proxy security credentials.
3. Invoke the web method.
4. Retrieve the results.


Add Web Reference
Navigate to a web service URL and click Add Reference to add all the available services.
Back
URL: http://localhost/AdvSalesSql?wsdl
Go
Web services found at this URL:
Web reference name:
Add Reference
Cancel

"AdvSalesEndpoint" Description

Methods

- **GetSalespersonList** ( ) As SqlRowSet
- **GetSalesPersonSales** (*SalespersonID* As int) As SqlRowSet
- **GetSalesTotal** (*SalespersonID* As int) As money

1 Service Found:

- AdvSalesSql

localhost

Figure 16-4. *Add Web Reference window*

All of these steps are shown in the VB code in Listing 16-5.

Listing 16-5. *VB Code for Web Service Consumer*

```
Imports System.Data
Imports System.Net

Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
        System.EventArgs) _
        Handles MyBase.Load

        ' Create a web service proxy using the SQL Server HTTP endpoint
        Dim proxy As New localhost.AdvSalesEndpoint

        ' Set the integrated security credentials
        proxy.Credentials = CredentialCache.DefaultCredentials
        ' Bind the combo box to the results of the web method call
        Me.cboSalesPerson.ValueMember = "SalespersonId"
        Me.cboSalesPerson.DisplayMember = "FullName"

        ' Call the web method
        Me.cboSalesPerson.DataSource = proxy.GetSalespersonList().Tables(0)

    End Sub

    Private Sub cboSalesPerson_SelectedIndexChanged(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles cboSalesPerson.SelectedIndexChanged

        ' Create a web service proxy using the SQL Server HTTP endpoint
        Dim proxy As New localhost.AdvSalesEndpoint

        ' Set the integrated security credentials
        proxy.Credentials = CredentialCache.DefaultCredentials

        ' Bind the data grid view to the results of the web method call
        Me.dgvSales.DataSource = proxy.GetSalesPersonSales(New _
            SqlTypes.SqlInt32(Me.cboSalesPerson.SelectedValue)).Tables(0)

        ' Populate the text box with the results of the second web method call
        Me.txtTotalSales.Text = proxy.GetSalesTotal(New _
            SqlTypes.SqlInt32(Me.cboSalesPerson.SelectedValue)).Value.ToString("C")

    End Sub

    Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As _
        System.EventArgs)  _
        Handles btnExit.Click

        ' Exit the application
        Application.Exit()

    End Sub
End Class
```

The form `Load` event creates a web service proxy, binds the `ComboBox`, and then calls the `GetSalesPersonList` web method to populate the `ComboBox`. Creating the proxy is as simple as declaring a new instance of the proxy class:

```
        ' Create a web service proxy using the SQL Server HTTP endpoint
        Dim proxy As New localhost.AdvSalesEndpoint
```

The security credentials are set using `System.Net.CredentialCache.DefaultCredentials`. This assigns the credentials of the currently logged-in user to the proxy object:

```
        ' Set the integrated security credentials
        proxy.Credentials = CredentialCache.DefaultCredentials
```

The `ComboBox` `ValueMember` and `DisplayMember` are assigned the names of the columns in the table returned by the web method:

```
        ' Bind the combo box to the results of the web method call
        Me.cboSalesPerson.ValueMember = "SalespersonId"
        Me.cboSalesPerson.DisplayMember = "FullName"
```

Finally, the `GetSalespersonList` web method is called, and the results are bound to the `ComboBox`:

```
        ' Call the web method
        Me.cboSalesPerson.DataSource = proxy.GetSalespersonList().Tables(0)

    End Sub
```

The `ComboBox` `SelectedIndexChanged` event is up next. This event, like the form `Load` event, creates a web service proxy object and assigns the appropriate security credentials to it using `System.Net.CredentialCache.DefaultCredentials`:

```
    Private Sub cboSalesPerson_SelectedIndexChanged(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles cboSalesPerson.SelectedIndexChanged

        ' Create a web service proxy using the SQL Server HTTP endpoint
        Dim proxy As New localhost.AdvSalesEndpoint

        ' Set the integrated security credentials
        proxy.Credentials = CredentialCache.DefaultCredentials
```

Next, the `SelectedIndexChanged` event passes the ID number of the selected salesperson to the `GetSalesPersonSales` web method. It binds the `DataGridView` to the results returned by the web method:

```
    ' Bind the data grid view to the results of the web method call
    Me.dgvSales.DataSource = proxy.GetSalesPersonSales(New _
        SqlTypes.SqlInt32(Me.cboSalesPerson.SelectedValue)).Tables(0)
```

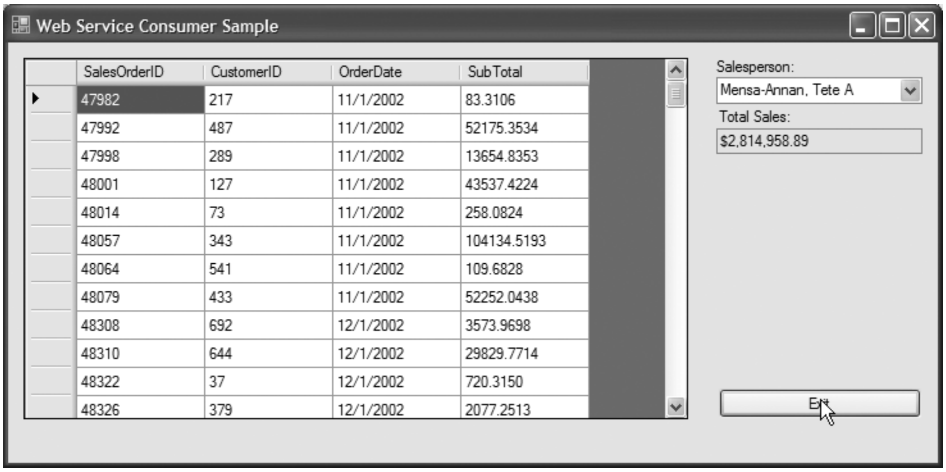
Then the `GetSalesTotal` web method is called with the selected salesperson ID as a parameter. The result is formatted as a currency string and displayed in the `TextBox`:

```
    ' Populate the text box with the results of the second web method call
    Me.txtTotalSales.Text = proxy.GetSalesTotal(New _
        SqlTypes.SqlInt32(Me.cboSalesPerson.SelectedValue)).Value.ToString("C")

End Sub
```

Finally, the `btnExit.Click` event exits the application:

```
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As _
    System.EventArgs) _
    Handles btnExit.Click

    ' Exit the application
    Application.Exit()

End Sub
```

Figure 16-5 shows a screenshot of the web service consumer application in action.



SalesOrderID	CustomerID	OrderDate	SubTotal
47982	217	11/1/2002	83.3106
47992	487	11/1/2002	52175.3534
47998	289	11/1/2002	13654.8353
48001	127	11/1/2002	43537.4224
48014	73	11/1/2002	258.0824
48057	343	11/1/2002	104134.5193
48064	541	11/1/2002	109.6828
48079	433	11/1/2002	52252.0438
48308	692	12/1/2002	3573.9698
48310	644	12/1/2002	29829.7714
48322	37	12/1/2002	720.3150
48326	379	12/1/2002	2077.2513

Salesperson: Mensa-Annan, Tete A

Total Sales: $2,814,958.89

Figure 16-5. *Web method consumer example in action*

In the sample application, selecting a different salesperson from the `ComboBox` automatically updates the `DataGridView` and Total Sales `TextBox` with results from the proper web methods.

As you can see, SQL Server HTTP endpoints make it easy to create and access stored procedure and user-defined function results from user applications.

Executing HTTP Endpoint Ad Hoc Queries

In addition to exposing stored procedures and user-defined functions, SQL Server HTTP endpoints provide the capability of performing ad hoc queries via web methods. You can enable ad hoc querying by specifying `BATCHES = ENABLED` when you create an HTTP endpoint.

■ **Caution** Ad hoc querying of SQL Server via web methods is a powerful feature, but it is also potentially dangerous. If you enable this feature, take extra care to make sure your server is properly secured.

Listing 16-6 creates an HTTP endpoint. No methods are declared explicitly, but the SOAP argument `BATCHES` is set to `ENABLED`. Because of this setting, a web method named `sqlbatch` is implicitly exposed.

Listing 16-6. *Endpoint Declared with Ad Hoc Querying Enabled*

```
CREATE ENDPOINT AdvAdHocEndpoint
    STATE = STARTED
AS HTTP (
    PATH = N'/AdvAdhocSql',
    AUTHENTICATION = (INTEGRATED),
    PORTS = (CLEAR),
    SITE = N'*'
)
FOR SOAP (
    WSDL = DEFAULT,
    DATABASE = N'AdventureWorks',
    SCHEMA = STANDARD,
    BATCHES = ENABLED
);
GO
```

Apart from the fact that this endpoint contains no explicit WEBMETHOD declarations, and the PATH argument is different, this declaration is very similar to the previous example. Once the endpoint is created, a web service consumer can be created to perform ad hoc queries against it. We'll start by creating a simple VB form like before. This one needs a TextBox, a Button, and a DataGridView control. It looks like Figure 16-6.

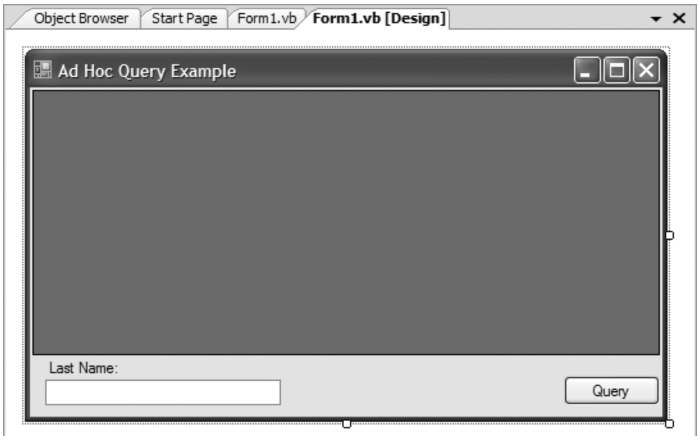


Figure 16-6. *Web method consumer form in design mode*

The sqlbatch Method

When you add the web reference to http://localhost/AdvAdhocSql?wsdl, the only method exposed is the sqlbatch web method. This method takes the following form:

```
sqlbatch ( BatchCommands As String,
    Parameters As ArrayOfSqlParameter ) As SqlResultStream
```

BatchCommands is a string containing the T-SQL statements/queries to be executed. These queries can be parameterized.

Parameters is an array of proxyclass.SqlParameter objects, where proxyclass is your web services proxy class. If your proxy class is named localhost (the default), a SqlParameter object would be declared as the following:

```
Dim p As New localhost.SqlParameter
```

The *Parameters* array represents the parameters to pass in for a parameterized query. If your query does not have parameters, pass Nothing for *Parameters*. Each proxyclass.SqlParameter object exposes a set of properties that need to be set prior to use. Table 16-1 lists some of the most common proxyclass.SqlParameter attributes.

Table 16-1. *Common proxyclass.SqlParameter Properties*

Name	Type	Description
`direction`	`proxyclass.ParameterDirection`	Specifies whether the parameter is an `Input` parameter or an `InputOutput` parameter.
`maxLength`	`Long`	Specifies the max length of the parameter `Value`.
`name`	`String`	Specifies the name of the parameter. Because of XML naming convention requirements, you must leave off the leading `@` in the parameter name.
`precision`	`Byte`	Specifies the parameter `Value` precision.
`scale`	`Byte`	Specifies the parameter `Value` scale.
`sqlDbType`	`proxyclass.sqlDbTypeEnum`	Determines the type of the parameter. Valid values include all of the SQL valid T-SQL data types, including `Int`, `VarChar`, `Char`, etc.
`Value`	`Object`	Specifies the value assigned to the parameter.

The VB code to create a `proxyclass.SqlParameter` and assign it a `varchar` string value might look like this:

```
Dim p As New localhost.SqlParameter
p.sqlDbType = localhost.sqlDbTypeEnum.VarChar
p.maxLength = 50
p.name = "LastName"
p.Value = "Smith"
```

The sample in Listing 16-7 demonstrates how to use the `sqlbatch` web method to perform a simple parameterized query using the HTTP endpoint created in Listing 16-6.

Listing 16-7. *sqlbatch Web Method Client*

```
Imports System.Net
Imports System.Data

Public Class Form1

    Private Sub btnQuery_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnQuery.Click
        ' Create web services proxy class
        Dim proxy As New localhost.AdvAdHocEndpoint

        ' Assign credentials to proxy
        proxy.Credentials = CredentialCache.DefaultCredentials

        ' Define the parameterized query here. Notice the leading @ is left on
        ' the parameter name in the query
        Dim sql As String = "SELECT ContactID, FirstName, MiddleName, LastName " & _
            " FROM Person.Contact " & _
            " WHERE LastName = @LastName"

        ' Create a parameter for the parameterized query
        Dim p As New localhost.SqlParameter
        p.sqlDbType = localhost.sqlDbTypeEnum.VarChar
        p.maxLength = 50
        ' Notice the leading @ is stripped off the parameter name here
        p.name = "LastName"
        p.Value = txtLastName.Text

        ' Call the sqlbatch web method with the query string and parameter array.
        ' Notice we have to create an array and put the parameter in it here
        Dim ds As DataSet = CType(proxy.sqlbatch(sql, _
            New localhost.SqlParameter() {p})(0), DataSet)

        ' Bind the data grid view to the result
        dgvResults.DataSource = ds.Tables(0)
    End Sub
End Class
```

The `btnQuery.Click` event handler begins like the previous examples. It creates an instance of the web service proxy class and assigns the appropriate security credentials to it:

```
        ' Create web services proxy class
        Dim proxy As New localhost.AdvAdHocEndpoint

        ' Assign credentials to proxy
        proxy.Credentials = CredentialCache.DefaultCredentials
```

Next, it defines a parameterized query string. Notice that the parameter in the query string still has the leading `@` sign:

```
' Define the parameterized query here. Notice the leading @ is left on
' the parameter name in the query
Dim sql As String = "SELECT ContactID, FirstName, MiddleName, LastName " & _
    " FROM Person.Contact " & _
    " WHERE LastName = @LastName"
```

Then it creates a single parameter for the query and sets its properties accordingly:

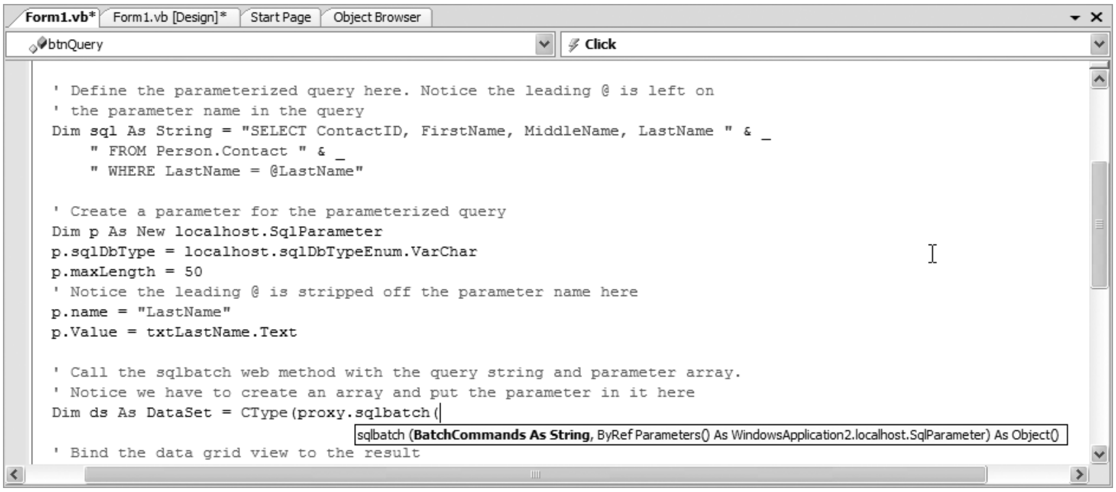
```
' Create a parameter for the parameterized query
Dim p As New localhost.SqlParameter
p.sqlDbType = localhost.sqlDbTypeEnum.VarChar
p.maxLength = 50
' Notice the leading @ is stripped off the parameter name here
p.name = "LastName"
p.Value = txtLastName.Text
```

Notice that in the name property of the parameter, the leading `@` sign is stripped off. The next step is to actually call the `sqlbatch` method and cast the result to a `DataSet`. The result is actually returned as an array of `Objects`, and the `Object` at element zero of the array is the `DataSet`. This could have been combined with the next step, but I split it up here to make it easier to read:

```
' Call the sqlbatch web method with the query string and parameter array.
' Notice we have to create an array and put the parameter in it here
Dim ds As DataSet = CType(proxy.sqlbatch(sql, _
    New localhost.SqlParameter() { p })(0), DataSet)
```

■**Note** The second parameter of the `sqlbatch` method is always an array of `SqlParameter` objects. Even if you are passing just one `SqlParameter`, it still needs to be a member of an array. In the example, I create an array of `SqlParameter` and initialize it on the fly with the following statement: `New localhost.SqlParameter() { p }`.

When using web methods, such as `sqlbatch`, Visual Studio provides IntelliSense support, as shown in Figure 16-7.



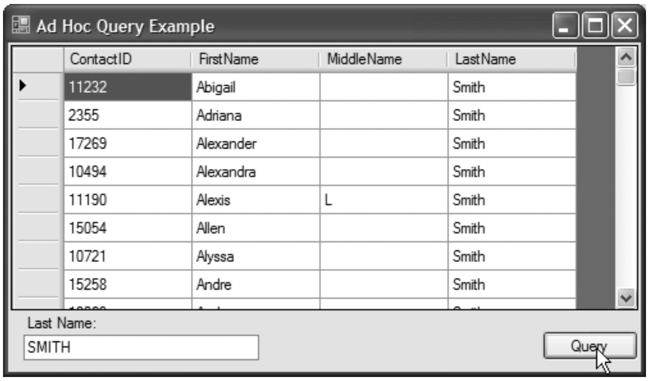
```
' Define the parameterized query here. Notice the leading @ is left on
' the parameter name in the query
Dim sql As String = "SELECT ContactID, FirstName, MiddleName, LastName " & _
    " FROM Person.Contact " & _
    " WHERE LastName = @LastName"

' Create a parameter for the parameterized query
Dim p As New localhost.SqlParameter
p.sqlDbType = localhost.sqlDbTypeEnum.VarChar
p.maxLength = 50
' Notice the leading @ is stripped off the parameter name here
p.name = "LastName"
p.Value = txtLastName.Text

' Call the sqlbatch web method with the query string and parameter array.
' Notice we have to create an array and put the parameter in it here
Dim ds As DataSet = CType(proxy.sqlbatch(
                         sqlbatch (BatchCommands As String, ByRef Parameters() As WindowsApplication2.localhost.SqlParameter) As Object()
' Bind the data grid view to the result
```

Figure 16-7. *Web method IntelliSense support*

The final step is to bind the results to the DataGridView. The actual rows returned by the query are in DataTable number 0 of the DataSet:

```
' Bind the data grid view to the result
dgvResults.DataSource = ds.Tables(0)
```

To test the application, just enter a last name in the TextBox and press the Query button. All contacts from the Person.Contact table with that last name are returned, as shown in Figure 16-8.



ContactID	FirstName	MiddleName	LastName
11232	Abigail		Smith
2355	Adriana		Smith
17269	Alexander		Smith
10494	Alexandra		Smith
11190	Alexis	L	Smith
15054	Allen		Smith
10721	Alyssa		Smith
15258	Andre		Smith

Figure 16-8. *Ad hoc query example in action*

Altering and Dropping Endpoints

You can drop an endpoint with the `DROP ENDPOINT` statement. The following is the format:

```
DROP ENDPOINT end_point_name;
```

The *end_point_name* is the name you gave the endpoint when you created it. To drop the endpoint you created for ad hoc queries, you would issue a `DROP ENDPOINT` statement like this:

```
DROP ENDPOINT AdvAdHocEndpoint;
```

Altering an endpoint requires the `ALTER ENDPOINT` statement. This statement has a format very similar to `CREATE ENDPOINT`. The following is the format:

```
ALTER ENDPOINT end_point_name
[ AUTHORIZATION login ]
[ STATE = { STARTED | STOPPED | DISABLED } ]
AS HTTP (
    [ [,] PATH = 'url' ]
    [ [,] AUTHENTICATION = ( { BASIC | DIGEST | INTEGRATED | NTLM | KERBEROS }
        [ , ... n ] ) ]
    [ [,] PORTS = ( { CLEAR | SSL } [ , ... n ] ) ]
    [ [,] SITE = { '*' | '+' | 'web_site' } ]
    [ [,] CLEAR_PORT = clear_port ]
    [ [,] SSL_PORT = ssl_port ]
    [ [,] AUTH_REALM = { 'realm' | NONE } ]
    [ [,] DEFAULT_LOGON_DOMAIN = { 'domain' | NONE } ]
    [ [,] COMPRESSION = { ENABLED | DISABLED } ]
)
FOR SOAP (
    [ [,] { ADD WEBMETHOD [ 'namespace' . ] 'method_alias'
        (
            NAME = 'database.schema.proc_name'
            [ , SCHEMA = { NONE | STANDARD | DEFAULT } ]
            [ , FORMAT = { ALL_RESULTS | ROWSETS_ONLY | NONE } ]
        )
      } [ , ... n ] ]
    [ [,] { ALTER WEBMETHOD [ 'namespace' . ] 'method_alias'
        (
            NAME = 'database.schema.proc_name'
            [ , SCHEMA = { NONE | STANDARD | DEFAULT } ]
            [ , FORMAT = { ALL_RESULTS | ROWSETS_ONLY | NONE } ]
        )
      } [ , ... n ] ]

    [ [,] DROP WEBMETHOD [ 'namespace' . ] 'method_alias' [ , ... n ] ]

    [ [,] BATCHES = { ENABLED | DISABLED } ]
    [ [,] WSDL = { NONE | DEFAULT | 'sp_name' } ]
    [ [,] SESSIONS = { ENABLED | DISABLED } ]
    [ [,] LOGIN_TYPE = { MIXED | WINDOWS } ]
    [ [,] SESSION_TIMEOUT = timeout_interval | NEVER ]
    [ [,] DATABASE = { 'database_name' | DEFAULT } ]
    [ [,] NAMESPACE = { 'namespace' | DEFAULT } ]
    [ [,] SCHEMA = { NONE | STANDARD } ]
    [ [,] CHARACTER_SET = { SQL | XML } ]
    [ [,] HEADER_LIMIT = header_limit ]
);
```

All the base HTTP endpoint arguments for `ALTER ENDPOINT` are the same as for `CREATE ENDPOINT`. These are described earlier in this chapter, so I won't relist them all here. The HTTP-specific arguments are also the same.

The SOAP-specific arguments are where `ALTER ENDPOINT` and `CREATE ENDPOINT` part ways. With `ALTER ENDPOINT` you can use `ADD WEBMETHOD` to add a new web method to your endpoint; `ALTER WEBMETHOD` to alter an existing web method; or `DROP WEBMETHOD` to drop an existing web method. The parameters for `ADD WEBMETHOD` and `ALTER WEBMETHOD` are the same as for `CREATE ENDPOINT`'s `WEBMETHOD` argument. The `DROP WEBMETHOD` accepts a *`method_alias`*.

All of the endpointwide SOAP-specific `ALTER ENDPOINT` arguments are the same as for `CREATE ENDPOINT`.

Tip When you issue an `ALTER ENDPOINT`, only include arguments that you actually want to change for the endpoint. Any arguments you leave out of the `ALTER ENDPOINT` statement will retain their current settings.

Summary

In this chapter I discussed SQL Server 2005 HTTP SOAP endpoints, including the following:

- Constructing the `CREATE ENDPOINT` statement
- Exposing stored procedures and scalar user-defined functions as web methods, both with and without parameters
- Adding a web service proxy to your Visual Studio project
- Accessing the HTTP endpoint web methods from Visual Basic
- Performing ad hoc queries over HTTP endpoints with the `sqlbatch` method
- Altering and dropping HTTP endpoints

At this point, I'd like to take a moment to thank you for reading this book. I hope you had as much fun reading it as I did writing it for you. I also hope that you find the content useful and informative, and I wish you well in your SQL Server 2005 development and all your other endeavors.