Providing SQL data as a web service using SQL Server 2005 Native Web Services

Part 1

One of the more interesting features introduced in SQL Server 2005 was Native Web Services. This feature makes it possible to expose SQL data to any client or application capable of sending a SOAP request.

I say “expose”, but in fact any data you make available in this way is well protected, with several different layers of permissions that need to be applied before data becomes available to the web service consumer.

In this article, I’ll address the steps necessary to make the results of a SQL stored procedure available to a Window-authenticated client for a named user on your own domain. A future article will describe how to call this data from an ASP.NET application, while another article will show you how to set up a SQL native web service using SSL certificates and SQL authentication to expose data outside your firewall to non-domain or anonymous users.

SQL 2005 gives you the opportunity to establish web services in a variety of ways. For now though, let’s focus on a simple SQL native web service on your Windows domain.

Preparation

You’ll need a SQL Server 2005 (not Express) server environment running on Windows Server 2003 in a Windows Active Directory domain to start with. You don’t need to have IIS running on this server. Depending on how the system HTTP listening process driver (Http.sys) is configured and whether or not your SQL Server service is running under the LocalSystem account, you may need Windows local admin permissions in order to configure the server to listen for SQL native web service requests.

First, let’s set up some test data. Connect to or create a test database, then run the following code to create and populate a data table. In all the example code in this article I refer to the database as TestDB; replace all references to TestDB with your own database name if this differs.

```
USE [TestDB]
GO

create table [dbo].[tbTestData](
	ID int not null identity(1, 1),
	[OrderNr] int not null,
	[OrderDate] datetime not null,
	[CustNr] int not null,
	[Processed] bit not null)
on [primary]

go
set ansi_padding off


INSERT INTO [TESTDB].[dbo].[tbTestData]
           ([OrderNr], [OrderDate], [CustNr], [Processed])
	VALUES
	     (1, getdate(), 1010, 0)

INSERT INTO [TESTDB].[dbo].[tbTestData]
           ([OrderNr], [OrderDate], [CustNr], [Processed])
	VALUES
	     (2, getdate(), 1013, 0)

INSERT INTO [TESTDB].[dbo].[tbTestData]
           ([OrderNr], [OrderDate], [CustNr], [Processed])
	VALUES
	     (3, getdate()+1, 1013, 0)

INSERT INTO [TESTDB].[dbo].[tbTestData]
           ([OrderNr], [OrderDate], [CustNr], [Processed])
	VALUES
	     (4, getdate()-1, 1005, 1)

INSERT INTO [TESTDB].[dbo].[tbTestData]
           ([OrderNr], [OrderDate], [CustNr], [Processed])
	VALUES
	     (5, getdate()+2, 1012, 0)
```

Now we've got a small test data set, let's create a basic stored procedure to query it:

```
USE [TestDB]
GO

create procedure dbo.stWSQuery

as
begin

	SELECT [ID], [OrderNr], [OrderDate], [CustNr], [Processed]
	FROM 		[dbo].[tbTestData]

end
```

The data returned by this stored procedure is the data we're going to make available in our web service.

Creating the endpoint

Execute this system stored procedure to reserve the namespace for applications calling your web service and to avoid conflicts:

```
exec sp_reserve_http_namespace N'http://*:80/testWS/'
```

The '*' simply means that all possible hostnames (not already reserved) are reserved, otherwise use your servername instead. Port 80 is the default port for non-encrypted HTTP traffic and is also used by IIS (when installed). 'testWS' in this instance is a virtual path we'll use to distinguish our web service endpoint from other endpoints (for database mirroring, for instance). You might also choose to implement security by granting connect permissions to one group of users to one endpoint, and another group connect rights for another endpoint.

You can use the httpcfg command-line utilty to view which namespaces are reserved on your server. From the command line, type:

```
httpcfg query urlacl
```

…to view a list of reserved namespaces and the access control lists defined for them.

The next step is to create the endpoint.

```
create endpoint WebTest
state = started
as http(
        path = '/testWS',
        authentication = (integrated),
        ports = (clear),
        site = '*'
        )
for soap (webmethod 'WS01'
                (name='TestDB.dbo.stWSQuery',
                schema = standard,
                format=rowsets_only),
        wsdl = default,
        schema = standard,
        database = 'TestDB')
```

The full syntax of CREATE ENDPOINT is described in some detail in Books Online so I'm not going to go through all the permutations here. However, a few elements are worth noting.

- The `path = '/testWS'` clause is the virtual path to be used by the web service, and should match the path defined in the previous step while reserving the namespace.
- The authentication clause declares that Windows integrated security is used for authentication.
- The `site = '*'` clause defines the endpoint site and should match the value used when calling the sp_reserve_http_namespace stored procedure.
- The **webmethod** clause defines the method that will be made available to consumers of the web service. Multiple methods can be defined per endpoint, each exposing a

stored procedure or SQL function. The **name** element defines (in this case) the stored procedure we want to execute when we call the web service.

- The **format** element of the webmethod defines how datasets are returned by the web service. The default setting is ALL_RESULTS, in which case our web service returns three datasets: the result set of the stored procedure, a row count rowset, and an error status rowset. In this example to keep things simple we just want to get the SP result set back, so we explicitly define the return format by using the clause `format=rowsets_only`

Security

The endpoint should now be present and active. Have a look in the Management Studio Object Explorer, you'll find it under (Your server name)\Server Objects\Endpoints\SOAP. However, in order to connect to it externally you'll need to define the appropriate permissions. For this example, we'll assume that we want to allow the user Fred, a user in our domain **bedrock**, to call our web service.

1. Create a SQL Server login for Fred on your server:

   ```
   exec master..sp_grantlogin 'BEDROCK\Fred'
   ```

2. Add Fred as a user in the database:

   ```
   use TestDB
   go

   exec sp_adduser 'BEDROCK\Fred'
   ```

3. Grant connect permissions on the endpoint to Fred:

   ```
   grant connect on endpoint::WebTest to [BEDROCK\Fred];
   ```

4. Grant execute permissions on the stored procedure to Fred:

   ```
   grant exec on stWSQuery to [BEDROCK\Fred];
   ```

Once Fred is logged in, he should now be able to see the web service's WSDL file by entering the address in his web browser:

http://[servername]/testWS?wsdl
or
http://[server IP address]/testWS?wsdl

In the next article in this series, we'll look at calling this web service from an ASP.NET application.