MSSQLTips.com
brought to you by Edgewood Solutions

Your daily source for SQL Server Tips

How To Call a SharePoint Web Service from a SQL Server CLR Function

Written By: Ray Barley -- 7/23/2008

Problem
Our business users are storing some data in SharePoint lists that we need to access from our ETL process that updates our SQL Server data warehouse. Can you provide us with the details and an example of how we can do this from a SQL Server CLR function?

Solution
Windows SharePoint Services (WSS) provides an API and a number of web services that can be used to access the data in the SharePoint content database programmatically. While the content database is in fact a SQL Server database, the recommended way of accessing the data is via the API or a web service. When using the API your code has to be running on the SharePoint server; you can call the web service from just about anywhere. Therefore, the web service approach provides the most flexibility and is probably the best in most cases. For a SharePoint list you can use the Lists web service. Starting with SQL Server 2005 you can code stored procedures, triggers, user-defined functions, user-defined aggregates, and user-defined types using Microsoft .NET code; e.g. Visual Basic .NET or C#. For the task at hand, a table-value function (TVF) written using the CLR would be a good choice. A TVF is a function that returns a result set. You can use the function in a SQL statement as if it were a table.

Before we proceed to walk through the code for our TVF, let's get our environment ready. By default the CLR is not enabled in SQL Server. Please refer to our earlier tip How to Return a Result Set from a SQL Server 2005 CLR Stored Procedure for the steps you need to perform to enable the CLR on your SQL Server instance.

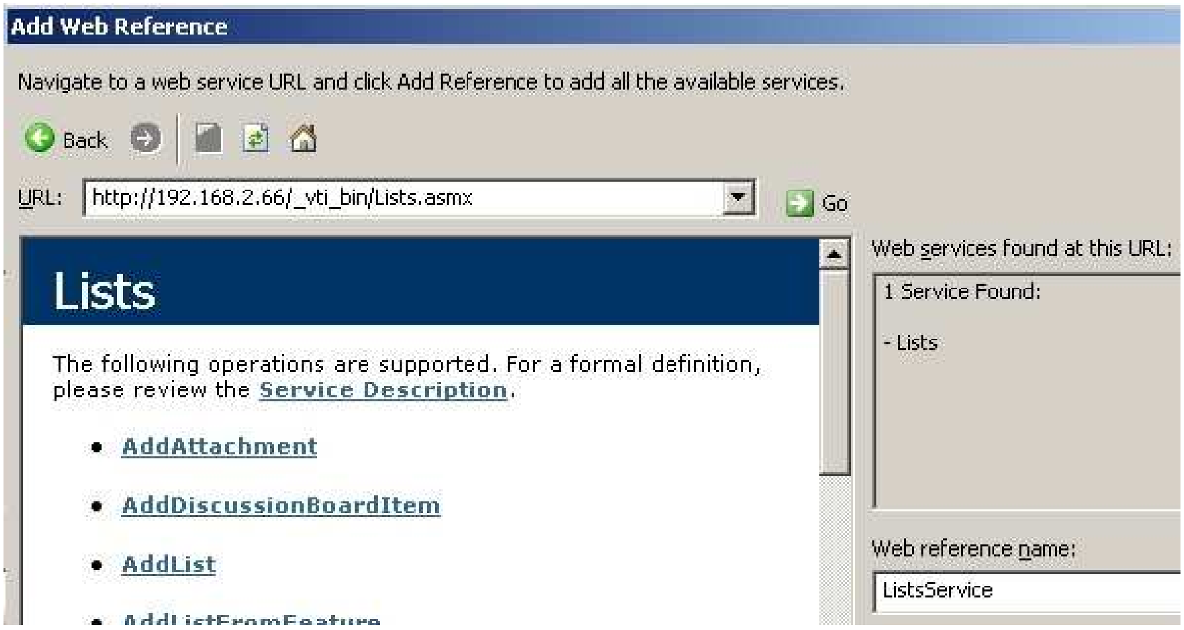
Now let's talk about what we are going to demonstrate in the code that follows. We need a TVF that returns a result set of the SharePoint lists for a particular site. It's a good idea to refer to a SharePoint list using it's Name which is a GUID. The textual name of a SharePoint list is the Title. Users can change the Title but the GUID remains unchanged. We need a second TVF to retrieve the contents of a particular SharePoint list.

We will walk through the following steps in our code example:

- Setup the web service proxy class
- Review the C# code sample for the CLR functions
- Generate a static serialization assembly
- Deploy the CLR functions

Web Service Proxy Class

When you write .NET code that uses a web service, you invoke the web service methods via a proxy class. The proxy class has methods that mirror the web service; you call the methods on the proxy class and the proxy class actually communicates with the web service. To create the proxy class you simply need the URL of the web service. In a Visual Studio project you can create the web service proxy by right clicking on the Web References node in the Solution Explorer. Alternatively you can create the web service proxy by running the WSDL utility which is a command line tool. In a Visual Studio project you fill in the following dialog:


Add Web Reference

Navigate to a web service URL and click Add Reference to add all the available services.

URL: http://192.168.2.66/_vti_bin/Lists.asmx

Lists

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

- AddAttachment
- AddDiscussionBoardItem
- AddList

Web services found at this URL:

1 Service Found:

- Lists

Web reference name:

ListsService

Note that the URL for your web service will be different than the one shown above which exists in a Virtual PC. You can specify anything you like for the Web reference name.

The command to create the web service proxy using the WSDL utility is as follows:

```
WSDL /o:ListsService.cs /n:WSS http://192.168.2.66/_vti_bin/Lists.asmx
```

The output from WSDL is a class file that you manually add to your Visual Studio project or include in your compile. The /n option specifies the namespace for the generated code. The advantage of using WSDL is that you can fine tune the generation of the proxy code; there are quite a few command line options available. Whether you use Visual Studio or WSDL, you wind up with the proxy class.

You can find WSDL.EXE in the folder C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin (assuming Visual Studio 2005).

CLR Code Sample

The following C# function (and one that follows it) will return a result set of the SharePoint lists in a particular site:

```
    [SqlFunction(SystemDataAccess = SystemDataAccessKind.Read,
                 FillRowMethodName = "GetListInfo")]
    public static IEnumerable GetListCollection(SqlString url)
    {
        DataTable t = new DataTable();
        WindowsImpersonationContext ctx = null;
        WindowsIdentity id = SqlContext.WindowsIdentity;
        try
        {
            // impersonate the caller
            ctx = id.Impersonate();
            // create instance of web service proxy class
            WSS.Lists svc = new WSS.Lists();
            svc.Url = url.ToString();
            svc.Credentials = CredentialCache.DefaultNetworkCredentials;
            // call web service method; return as a DataTable
            XmlNode node = svc.GetListCollection();
            XmlTextReader rdr = new XmlTextReader(node.OuterXml,
                                        XmlNodeType.Element, null);
            DataSet ds = new DataSet();
            ds.ReadXml(rdr);
            t = ds.Tables[0];
        }
        finally
        {
            // undo the impersonation
            if (ctx != null)
                ctx.Undo();
        }
        return t.Rows;
```

The main points in the above code are:

- The [SqlFunction] which precedes the actual function is a .NET attribute. It is used to specify metadata about the .NET object which it adorns; in this case the class. The [SqlFunction] attribute is used for a CLR function. The SystemDataAccess property is required because we will be impersonating the caller and this requires access to system tables. The FillRowMethodName property is the name of a companion function (to be discussed below); this function actually returns the result set a row at a time.
- A CLR TVF must return an object that implements the IEnumerable interface. This is a standard interface implemented by .NET collection classes.
- The parameter to the function is the URL of the Lists service for a site; e.g. http://servername /_vti_bin/Lists.asmx.
- DataTable is a class in the .NET framework that provides an in-memory representation of a table or result set. It's base class implements the IEnumerable interface required to be returned by the function.
- The SqlContext object is automatically available in a CLR function; we use it to get the WindowsIdentity of the caller.
- The default behavior when accessing resources outside of SQL Server is to use the credentials of the SQL Server service; in many cases this isn't appropriate as the SQL Server service's account has the bare minimum permissions required. In many cases you will need to impersonate the caller who then must have the appropriate permissions.
- After creating the instance of the web service proxy class, we set the URL based on the value passed to the function and specify to use the current credentials which are now the caller's after the impersonation code is executed.
- We call the web service method which returns an XML node object.
- We transform the XML into a Data Table via the DataSet object, which is a standard class in the .NET framework which implements an in-memory copy of a collection of tables or result sets.
- In the finally block we undo the impersonation, reverting back to the credentials of the SQL Server service account.
- In the last line of code we return the collection of rows in the DataTable.

The second part of our code sample is the GetListInfo function which was specified in the SqlFunction attribute in the FillRowMethodName property:

```
    public static void GetListInfo(
                        object obj,
                        out SqlString name,
                        out SqlString title,
                        out SqlString url )
    {
        DataRow r = (DataRow)obj;
        name = new SqlString(r["Name"].ToString());
        title = new SqlString(r["Title"].ToString());
        url = new SqlString(r["DefaultViewUrl"].ToString());
    }
```

The main points in the above code are:

- This function is called behind the scenes; we will connect the GetListCollection function above to the TVF when we deploy (in the next section). The GetListCollection function retrieves the data to be returned by the TVF; this function is called to return the result set one row at a time.
- The object parameter value is actually a DataRow; the GetListCollection function returned a collection of DataRow objects.
- We use the SqlString object instead of the standard .NET String class.
- Each column in the result set is returned as an out parameter in the function.

The above functions should be implemented in a single source file (e.g. SharePointList.cs) and compiled to create a class library DLL. The sample code is available here. You can use a Visual Studio project to create the class library DLL or compile with the following command:

```
CSC /target:library /out:MSSQLTipsCLRLib.dll *.cs
```

You can find CSC.EXE in the folder C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727 (assuming v2.0 of the .NET framework).

Generate a Static Serialization Assembly

When writing .NET code that accesses a web service, there are some behind the scenes things that happen to dynamically generate code used to pass data to and from the web service. This is called serialization and just think of it as a special packaging of data to and from the web service. When calling a web service from a CLR function, the SQL Server CLR implementation doesn't allow this dynamic serialization. The work around is to use the SGEN utility to create a static serialization assembly (i.e. DLL). The command line for SGEN is as follows (assuming we compiled our earlier functions into a class library called MSSQLTipsCLRLib):

```
SGEN /a:MSSQLTipsCLRLib.dll
```

SGEN creates a DLL; in this case MSSQLTipsCLRLib.XmlSerializers.dll. Both MSSQLTipsCLRLib.dll and MSSQLTipsCLRLib.XmlSerializers.dll must be deployed to SQL Server, which we will do in the next section.

You can find SGEN.EXE in the folder C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin (assuming Visual Studio 2005).

Deploy the CLR Functions to SQL Server

Execute the following T-SQL script to deploy the DLLs to SQL Server and create the CLR function (create the mssqltips database if necessary or substitute an existing database name):

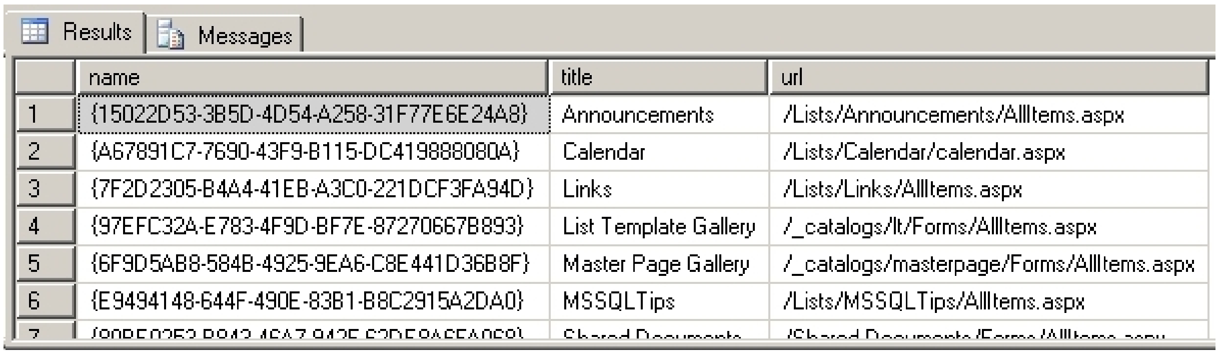
```
ALTER DATABASE mssqltips
SET TRUSTWORTHY ON
GO
USE mssqltips
GO
CREATE ASSEMBLY MSSQLTipsCLRLib
FROM 'C:\mssqltips\MSSQLTipsCLRLib.dll'
WITH PERMISSION_SET = UNSAFE
GO
CREATE ASSEMBLY [MSSQLTipsCLRLib.XmlSerializers]
FROM 'C:\mssqltips\MSSQLTipsCLRLib.XmlSerializers.dll'
WITH PERMISSION_SET = SAFE
GO
CREATE FUNCTION fn_GetSharePointLists(@url nvarchar(256))
RETURNS TABLE
(
name nvarchar(256),
title nvarchar(256),
url nvarchar(256)
)
AS
EXTERNAL NAME MSSQLTipsCLRLib.SharePointList.GetListCollection
GO
```

The TRUSTWORTHY option is set on to allow the external access outside of SQL Server; the default is SAFE. CREATE ASSEMBLY deploys the DLL to SQL Server. The FROM clause must point to the actual path of the DLL. PERMISSION_SET must be set to UNSAFE because of using certain .NET serialization attributes associated with the web service. Finally the EXTERNAL NAME of the CREATE FUNCTION creates the TVF and associates it to the assembly, class and function. When you include the TVF in the FROM clause of a SELECT statement, the .NET code is run and it returns a result set.

To execute the TVF, execute the following script (substitute the URL to your SharePoint site):

```
SELECT * FROM dbo.GetListCollection 'http://192.168.2.66/_vti_bin/Lists.asmx'
```

You will see output that looks something like this:



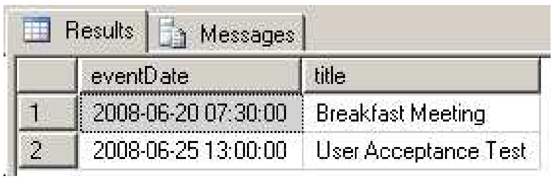
	name	title	url
1	{15022D53-3B5D-4D54-A258-31F77E6E24A8}	Announcements	/Lists/Announcements/AllItems.aspx
2	{A67891C7-7690-43F9-B115-DC419888080A}	Calendar	/Lists/Calendar/calendar.aspx
3	{7F2D2305-B4A4-41EB-A3C0-221DCF3FA94D}	Links	/Lists/Links/AllItems.aspx
4	{97EFC32A-E783-4F9D-BF7E-87270667B893}	List Template Gallery	/_catalogs/lt/Forms/AllItems.aspx
5	{6F9D5AB8-584B-4925-9EA6-C8E441D36B8F}	Master Page Gallery	/_catalogs/masterpage/Forms/AllItems.aspx
6	{E9494148-644F-490E-83B1-B8C2915A2DA0}	MSSQLTips	/Lists/MSSQLTips/AllItems.aspx
7	[illegible]	Shared Documents	[illegible]

The second function that was discussed was a TVF that would retrieve the items from a SharePoint list and return them as a result set. The code is practically the same as above, but calls a different web service method to retrieve the list of items from a Calendar. You can review the code in the download here. You also have to execute a T-SQL command to create the function; it is in the SQL script file in the download.

To execute the TVF, execute the following script:

```
SELECT *
FROM dbo.fn_GetSharePointCalendar(
           'http://192.168.2.66/_vti_bin/Lists.asmx',
           '{A67891C7-7690-43F9-B115-DC419888080A}')
```

The second parameter is the list name which is a GUID; you get this value by executing the fn_GetSharePointLists function. You will see output that looks something like this (only two columns are returned):



	eventDate	title
1	2008-06-20 07:30:00	Breakfast Meeting
2	2008-06-25 13:00:00	User Acceptance Test

Next Steps

- Take a look at our earlier tips on CLR String Sort Function in SQL Server 2005, CLR function to delete older backup and log files in SQL Server, and How to Return a Result Set from a SQL Server 2005 CLR Stored Procedure for additional details on CLR functions and stored procedures in SQL Server.
- Download the sample script here and experiment with invoking a SharePoint web service from a CLR function.
- Keep in mind that when there is a function in the Microsoft .NET framework that does what you need, using the CLR in SQL Server may be a good solution.
- Visual Studio 2005 has a SQL Server project template which simplifies the creation and deployment of the CLR integration code.