Developing Connected Smart Device Applications with SqlClient

Prashant Dhingra
Microsoft
September 2007

The Microsoft .NET SqlClient class provides a consistent set of APIs for both .NET Framework and .NET Compact Framework applications. SqlClient is a data provider for Microsoft SQL Server. SqlServerCe is a data provider for accessing Microsoft SQL Server Compact Edition database.

The SqlClient namespace exists both in the full .NET Framework and in .NET Compact Framework. Generally speaking there are three high-level ways of providing data access using the Microsoft data providers:

1. Using a .NET Framework-based SqlClient data provider to access data in a SQL Server on from a PC
2. Using a .NET Compact Framework-based SqlClient data provider to access data in a SQL Server from a device.

Accessing a SQL Server database from a device application is very similar to accessing a SQL Server database from a desktop application (methods 1 and 2 above). This exercise will look at number 2 by demonstrating how you can query and update a backend SQL Server Database using a Compact Framework-based Smart Device application.

In this exercise you will develop a .NET Compact Framework application using Visual Studio 2005. The application will allow you to see real time data stored in SQL Server 2005. Since the application is directly accessing and updating SQL Server, it does not require the use of SQL Server Compact Edition nor do you need to configure replication. It does require that the device have reliable connectivity with the backend server while working on this application.

From Device you can connect to SQL Server using TCP/IP connection. You need to configure SQL Server to accept remote connections over TCP/IP.

Enable Remote Connections

First you need to allow remote connections access to your SQL Server 2005 instance. To allow remote connections, follow the steps below.

1. Click on Start | All Programs | Microsoft SQL Server 2005 | Configuration Tools | SQL Server Surface Area configuration.
2. On the surface area configuration dialog click on "Surface Area Configuration for Services and Connections" option.

3. Click on Remote Connections in Database engine node.

4. Enable Remote connections by clicking on Local and remote connections.
5. Select sub option "Using both TCP/IP and name pipes"


**Configure Windows Firewall**

If your system has the Windows Firewall configured, it must be configured to allow the remote connection to communicate with the SQL Server executable. Follow the steps given below to ensure connectivity through the firewall.

1. Start the Control Panel on the computer running SQL Server.
2. Double Click on Windows Firewall.
3. Click on the Exceptions tab.
4. Click on the Add Program button.
5. Add the path of sqlsrver.exe. By default the sqlsrver.exe is installed at `<Drive>\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\sqlsrver.exe`
Develop a SQL Server Management Studio application for Smart Devices

Now that you have configured your SQL Server to allow remote connections we’ll walk through creating a simple SQL Server Management Studio-like application that will run on a Smart Device. This application will allow you to run SQL statements against the remote SQL Server directly from a connected device.

This example uses the AdventureWorksDW database to allow users to run ad-hoc queries against the database without having to be at a desktop computer. You can alter the connection string in the code to target any other database, or alternatively you might extend the code to allow the user to enter the database name through the user interface.

Follow these steps to create the Smart Device application:

3. Design the user interface for project as shown in figure.
4. Use Toolbox to create a Tab control with 4 tabs – Objects, SQL, Result and Notes.
5. Create a Tree node on Objects Tab to display SQL Server database objects.
6. Create a Text box in SQL Tab to run queries.
7. Create a DataGrid on Result tab to display result.
8. Create a Text box on Notes Tab.
9. Create a button to execute queries.
10. Optionally create multiple buttons to store queries
11. You can make a user interface similar to SQL Server Compact Edition Query Analyzer that connects to SQL Server instead of SQL Server Compact Edition database.

The code below demonstrates how to access the remote SQL server using the SqlClient provider objects and display the results in a DataGrid.

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace SSMS_Device
{
    public partial class SSMS : Form
    {
        SqlConnection myConn;
        String connStr;

        public SSMS()
        {
            InitializeComponent();
            connStr = @"Server='PDHINGRA';Database=AdventureWorksDW;"
                    + "User Id=MyLogin; Password=MyPassword";

            try
            {
                myConn = new SqlConnection(connStr);
                myConn.Open();
                FillList();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Connection error: ";
                                + ex.ToString());
            }
        }
    }

    private void button10_Click(object sender, EventArgs e)
    {
        try
        {
            myConn = new SqlConnection(connStr);
            myConn.Open();
            FillList();
        }
        catch (Exception ex)
        {
            MessageBox.Show("Connection error: ";
                            + ex.ToString());
        }
    }
}
```

©2007 OpenNETCF Consulting
For more information visit http://community.OpenNETCF.com
finally
{
    myConn.Close();
}
}

private void FillList()
{
    SqlDataAdapter myAdapter =
        new SqlDataAdapter(txtSQL.Text, myConn);
    DataSet myDataSet =
        new DataSet();
    myAdapter.Fill(myDataSet, "Result");
dataGridView1.DataSource =
    myDataSet.Tables["Result"].DefaultView;
}

private void FillObject()
{
    try
    {
        myConn =
            new SqlConnection(connStr);
        myConn.Open();
        SqlDataReader myDataReader;
        SqlCommand myCmd = myConn.CreateCommand();

        myCmd.CommandText =
            "SELECT TABLE_SCHEMA, TABLE_NAME FROM"
            + " INFORMATION_SCHEMA.TABLES"
            + " WHERE TABLE_TYPE = 'BASE TABLE'";
        treeView1.BeginUpdate();
        treeView1.Nodes.Clear();
        treeView1.Nodes.Add(new TreeNode("Tables"));
        TreeNode childNode = treeView1.Nodes[0];

        int childCount = 0;
        myDataReader = myCmd.ExecuteReader();
        while (myDataReader.Read())
        {
            string TableName =
                (myDataReader[0].ToString()) + "." +
                (myDataReader[1].ToString());

            childNode.Nodes.Insert
                (childCount, new TreeNode(TableName));
            childCount += 1;
        }
        myDataReader.Close();
        treeView1.Nodes.Add(new TreeNode("Views"));

        SqlCommand myCmdView = myConn.CreateCommand();
        myCmdView.CommandText =
            "SELECT TABLE_SCHEMA, TABLE_NAME FROM"
            + " INFORMATION_SCHEMA.TABLES"
            + " WHERE TABLE_TYPE = 'VIEW'";
        TreeNode childNodeView = treeView1.Nodes[1];
```csharp
int childCountView = 0;

myDataReader = myCmdView.ExecuteReader();
while (myDataReader.Read())
{
    string ViewName = (myDataReader[0].ToString()) + 
    "." + (myDataReader[1].ToString());
    childNodeView.Nodes.Insert
    (childCount, new TreeNode(ViewName));
    childCountView += 1;
}

myDataReader.Close();

treeView1.ExpandAll();
treeView1.EndUpdate();
myConn.Close();
} // try

catch (SqlException myexception)
{
    foreach (SqlError err in myexception.Errors)
    {
        MessageBox.Show(err.Message);
    }
}
} // catch

} //Fill Object Tree

private void SSMS_Load(object sender, EventArgs e)
{
    FillObject();
}
```
1. Build and compile the application.
2. Deploy the application to an emulator or a device.
3. The application will start and create a connection to backend SQL Server.
4. The application then fetches the table and view names from the INFORMATION_SCHEMA.TABLES view.
5. The tables and views are then displayed in a TreeView as shown in the figure below.
6. Click on the SQL Tab and enter a SQL query such as that shown in the figure below.
7. Click on the Execute Button (“>”) to run your query.

8. When the Execute B is clicked, the application creates a connection to the backend AdventureWorks database and executes the query. The result are then populated in the Results Tab as shown in figure below.
9. Click on the Results Tab to see the result.

The code in the download doesn’t use the Notes tab, but you could add information to it such as the time required to execute the query, the number of rows affected, etc.

**Difference between SqlClient provider in .NET Framework and .in NET Compact Framework**

The System.Data.SqlClient provider is a collection of classes to access SQL Server 2005 databases. You can use the SqlClient provider in a Smart Device platform in the same way you use the SqlClient provider in a Desktop application, however there are few differences in the .NET Compact Framework version of the provider that are important to note. For the .NET Compact Framework

- The SqlConnectionPermission and SqlConnectionPermissionAttribute classes are not supported.

©2007 OpenNETCF Consulting
For more information visit http://community.OpenNETCF.com
- Connection Pooling is not supported
- Transactions spanning to multiple databases are not supported
- Direct TCP/IP connections to a SQL Server instance are supported
- Windows authentication is supported, however you must specify the user id and password in the connection string.
- Encrypted connections to SQL Server instances are not supported. The connection will fail if computer running SQL Server has an SSL certificate installed

In this exercise you built a tool similar to SQL Server Management Studio to run on Mobile devices. You can use the tool to execute SQL queries from a mobile device directly against a backend SQL Server. This exercise also demonstrated how to build connected enterprise applications for mobile devices.

**Building an Enterprise Solution – Connected vs Disconnected**

This exercise has shown that it’s straightforward to create an enterprise application with a direct connection to a back-end SQL Server. An alternative is to maintain a local SQL CE Compact Edition data store on the Smart Device itself and use the SqlServerCe provider for data access.