# Data Connection Dialog Help

This document provides guidelines to use Data Connection Dialog source code.

Contents

[Data Connection Dialog Help 1](#_Toc249511083)

[Introduction 1](#_Toc249511084)

[How to Use Data Connection Dialog 3](#_Toc249511085)

[Sample 1: Build Connection String 3](#_Toc249511086)

[Sample 2: Modify Existing Connection String: 4](#_Toc249511087)

[How to Configure Data Connection Dialog 4](#_Toc249511088)

[Configure DataSource List 5](#_Toc249511089)

[Configure Selection Status 5](#_Toc249511090)

[How to Write a Custom Data Provider 6](#_Toc249511091)

[Implementing the connection properties 7](#_Toc249511092)

[Implementing the connection control 7](#_Toc249511093)

[Known Issues 8](#_Toc249511094)

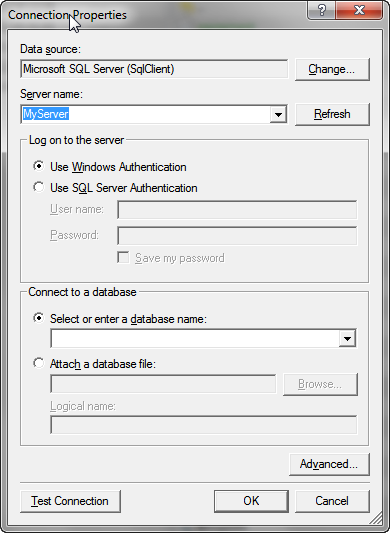
## Introduction

Previously, the data connection dialog was a database tool component that released with Visual Studio. It allows users to build connection strings and to connect to specific data sources. Details can be found at the following links:

1. [How to: Connect to a Database from Server Explorer](http://msdn.microsoft.com/en-us/library/0wbea1ae.aspx)
2. [Introduction to Visual Studio Data Designer Extensibility (DDEX)](http://msdn.microsoft.com/en-us/library/ms379576(VS.80).aspx#ddexintro_topic3b)

In order to use the data connection dialog independent of VS, we released the standalone source code. You can integrate and redistribute the source code with your application. This document elaborates how to modify the source code and how to write a custom data provider as well.

Following is the Data Connection dialog UI:



Overall, there are five projects in the source code solution:

|  |  |
| --- | --- |
| Project Name | Description |
| Microsoft.Data.ConnectionUI | Provides interface definitions. |
| Microsoft.Data.ConnectionUI.Dialog | Provides the implementation of the connection dialog including the UI, several common data sources and data providers. |
| Microsoft.Data.ConnectionUI.Sample | This is a C# sample project demonstrating how to use the connection dialog in code. Also, it demonstrates how to configure the data connection dialog. |
| Microsoft.Data.ConnectionUI.Sample.VB | This is a VB sample project demonstrating how to use the connection dialog in code. Also, it demonstrates how to configure the data connection dialog. |
| Microsoft.Data.ConnectionUI.SqlCeDataProvider | This is a sample project demonstrating how to write a custom data provider. The sample uses SQL Server Compact Edition for demonstration. |

## How to Use the Data Connection Dialog

The main implementation of the data connection dialog is at the project ***Microsoft.Data.ConnectionUI.Dialog***, which contains dialogs, common data providers and their corresponding UIs. The controller part is the ***DataConnectionDialog*** object which maintains logics and status, and manages navigation between UIs.

In order to fully leverage the source code, you can follow these steps:

1. Create a C# or VB Console Application or Windows Form Application. Skip this step if you already have an application.

If you choose console application, make sure to add ***[STAThread]*** attribute to the main entry point to invoke the dialog box correctly.

1. Add references to ***Microsoft.Data.ConnectionUI*** and ***Microsoft.Data.ConnectionUI.Dialog*** to your project.

If you are writing a VB application, you can build out these two projects and add built-out binaries as assembly references.

1. Copy the following two files to your project:

For C# the files needed to copy are ***IDataConnectionConfiguration.cs*** and ***DataConnectionConfiguration.cs*** from ***Microsoft.Data.ConnectionUI.Sample***.

For VB, the files needed to copy are ***IDataConnectionConfiguration.VB and IDataConnectionConfiguration.VB*** from the project ***Microsoft.Data.ConnectionUI.Sample.VB.***

1. Add a namespace(or imports in VB) using statement for Microsoft.Data.ConnectionUI.

After completing the previous steps, you can use the data connection dialog:

### Sample 1: Build Connection String

The first thing that can do with data connection dialog is to build a connection string. Here is the code snippet:

#### C# Sample:

[STAThread]

static void Main(string[] args)

{

DataConnectionDialog dcd = new DataConnectionDialog();

DataConnectionConfiguration dcs = new DataConnectionConfiguration(null);

dcs.LoadConfiguration(dcd);

if (DataConnectionDialog.Show(dcd) == DialogResult.OK)

{

// load tables

using (SqlConnection connection = new SqlConnection(dcd.ConnectionString))

{

connection.Open();

SqlCommand cmd = new SqlCommand("SELECT \* FROM sys.Tables", connection);

using (SqlDataReader reader = cmd.ExecuteReader())

{

while (reader.Read())

{

Console.WriteLine(reader.HasRows);

}

}

}

}

dcs.SaveConfiguration(dcd);

}

#### VB Sample:

<STAThread()> \_

Public Sub Main(ByVal args As String())

Dim dcd As New DataConnectionDialog()

Dim dcs As New DataConnectionConfiguration(Nothing)

dcs.LoadConfiguration(dcd)

If DataConnectionDialog.Show(dcd) = DialogResult.OK Then

' load tables

Using connection As New SqlConnection(dcd.ConnectionString)

connection.Open()

Dim cmd As New SqlCommand("SELECT \* FROM sys.Tables", connection)

Using reader As SqlDataReader = cmd.ExecuteReader()

While reader.Read()

Console.WriteLine(reader.HasRows)

End While

End Using

End Using

End If

dcs.SaveConfiguration(dcd)

End Sub

The code snippet initializes a data connection dialog, loads its configuration, and displays the dialog (the configuration details will be explained in next section). After completing the dialog and clicking OK, you can use the ***ConnectionString*** property of the dialog to connect to data sources and execute corresponding operations.

### Sample 2: Modify Existing Connection String:

The dialog box can also be used to modify existing connection strings:

#### C# Sample:

DataConnectionDialog dcd = new DataConnectionDialog();  
DataConnectionConfiguration dcs = new DataConnectionConfiguration(null);  
dcs.LoadConfiguration(dcd);  
dcd.ConnectionString = "Data Source=.\SqlExpress;Initial Catalog=tempdb;Integrated Security=True";  
// The remaining is the same as sample 1

#### VB Sample:

Dim dcd As New DataConnectionDialog()

Dim dcs As New DataConnectionConfiguration(Nothing)

dcs.LoadConfiguration(dcd)

dcd.ConnectionString = "Data Source=ziz-vspro-sql05;Initial Catalog=Northwind;Persist Security Info=True;User ID=sa;Password=Admin\_007";

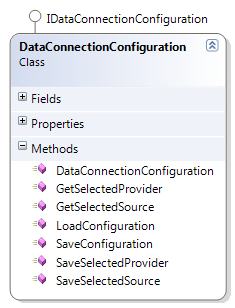
' The remaining is the same as sample 1

Here the ConnectionString is passed to the dialog before showing the dialog. The caveat is that you should set ***ConnectionString*** after loading the configuration.

## How to Configure Data Connection Dialog

In previous sections, we have gone through how to use the Data Connection dialog. The default dialog contains data sources for SQL Server, SQL Server Database File, Access Database File, ODBC Data Source and Oracle Database. A custom data source for SQL Compact Edition is provided as well. If the default list is suitable for your needs, you don’t need to customize the dialog.

In this section, we will show how to customize the data connection dialog.



Besides the default behavior of the dialog, there are two parts that can be customized. The customization mechanism is implemented in ***DataConnectionConfiguration*** which inherits from ***IDataConnectionConfiguration***. In fact, this class decouples the standalone version of data connection dialog from Visual Studio.

The ***DataConnectionConfiguration*** class offers two functions : configuring the data sources list and maintaining selection status. So as long as you implement the same functionality, you can replace the configuration class with your own.

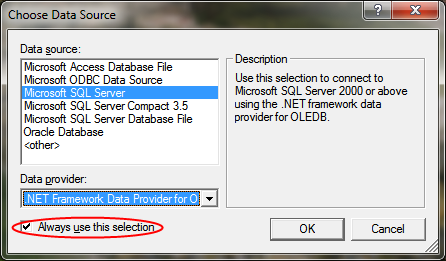
### Configure DataSource List

The ***DataConnectionConfiguration*** provides the ***LoadConfiguration*** method to load data sources and data providers. You can add custom data providers, for example, a SQL CE data provider to the dialog box with one line of code,

dialog.DataSources.Add(SqlCe.SqlCeDataSource);

### Configure Selection Status

When users check “*Always use this selection*”, in Visual Studio, the Data Connection Dialog’s configuration will be saved to the registry together with some other Visual Studio settings.



In order to decouple from VS, in the sample code, it provides saving configuration into an xml file: DataConnection.xml. You can choose where to save this file. By default, this file will be saved to the same folder as the application.

<?xml version="1.0" encoding="utf-8"?>

<ConnectionDialog>

<DataSourceSelection>

<SelectedSource>MicrosoftSqlServer</SelectedSource>

<SelectedProvider>System.Data.SqlClient</SelectedProvider>

</DataSourceSelection>

</ConnectionDialog>

The schema of this xml is straight forward. It saves both the data source type and data provider type.

As described above, you can save the configuration to other places such as the registry or the application configuration file with your own ***IDataConnectionConfiguration***.

## How to Write a Custom Data Provider

If the default list doesn’t meet your requirements (For example, you want a connection string to a MySQL data source), you can write a custom data provider. We provide a sample for SQL CE data source which is included in the source code solution.

There are two interfaces needing to be implemented: ***IDataConnectionProperties***, ***IDataConnectionUIControl***.

The interfaces enable the Data Connection dialog box to interact with connection properties for a specific data provider, allowing users to input and edit connection property values.

|  |  |
| --- | --- |
| Interface | Description |
| [IDataConnectionProperties](http://msdn.microsoft.com/en-us/library/microsoft.data.connectionui.idataconnectionproperties.aspx) | Provides a set of methods and properties that enable the **Data Connection** dialog box to interact with a specific data provider's connection properties. |
| [IDataConnectionUIControl](http://msdn.microsoft.com/en-us/library/microsoft.data.connectionui.idataconnectionuicontrol.aspx) | Provides a set of methods and properties through which the **Data Connection** dialog box interacts with a third-party data connection user interface (UI) control, which is shown as the body of the **Data Connection** dialog box. |

### Implementing the connection properties

For an ADO.NET 2.0-compliant data provider, the **IDataConnectionProperties** implementation can inherit from **AdoDotNetConnectionProperties** class to minimize the work. Make sure to override the ***IsComplete*** method, which indicates that the UI is ready for a connection.

On the other hand, when the user control is not specified, the dialog hosts a property grid containing the properties exposed by the **IDataConnectionProperties** object. When the **IDataConnectionProperties** object is not supported, the dialog hosts a property grid containing a single property, **ConnectionString**, in which a user can directly enter a connection string.

### Implementing the connection control

The first step when implementing the connection control is to create an assembly to contain the user control. To create this assembly, do the following:

1. Create a new class library project in Visual Studio and add references to Microsoft.Data.ConnectionUI.dll and Microsoft.Data.ConnectionUI.Dialog.dll.
2. Add a user control to the project.
3. Close the designer for this user control and reopen the document in code view.
4. Add a namespace(or imports in VB) using statement for Microsoft.Data.ConnectionUI.
5. Build the solution.
6. Reopen the designer for the user control

Before populating this control, there are two important things to note. The first regards the contract between the user control and the hosting dialog box. The second regards style guidelines on the dialog box, to ensure a connection UI control does not look out of place when hosted in the dialog. The following paragraphs address these points.

#### UI Control Contract

There is a strict contract between the Data Connection dialog box and the data connection UI controls that it hosts. Controls are constructed on demand, as a user initiates an action that would cause the control to show. Immediately after construction, the control's **Initialize** method is called with a **IDataConnectionProperties** instance to be used as the underlying store of properties surfaced by the control. These properties are available through the protected **ConnectionProperties** property. Then, at appropriate times, the **LoadProperties** method is called to indicate that current values stored in the **DataConnectionProperties** instance should be loaded into the controls on the user control.

#### UI Style Guidelines

Visual Studio has a set of recommended guidelines for layout and spacing of controls on dialog boxes. These guidelines have been followed by the data connection dialog and thus, to ensure neat integration with this dialog box, all user controls should observe these same guidelines. In addition to this, there are a couple of points of interest that will ensure that the user control looks correct when hosted by the dialog box:

* Set the AutoScaleMode to Font. This will ensure that scaling of your control will occur in sync with the rest of the dialog.
* Set the Margin to (0, 0, 0, 0) and when placing inner controls, allow them to touch the edges. For example, a label that should appear in the upper left corner of your user control should be located at position 0, 0 and a text box that should stretch the width of the user control should have a width equal to the width of the user control.
* Set the Size to something greater than or equal to (300, 1). This is the minimum size allowed for user controls so any controls smaller than this will not fill all the space allocated for it.

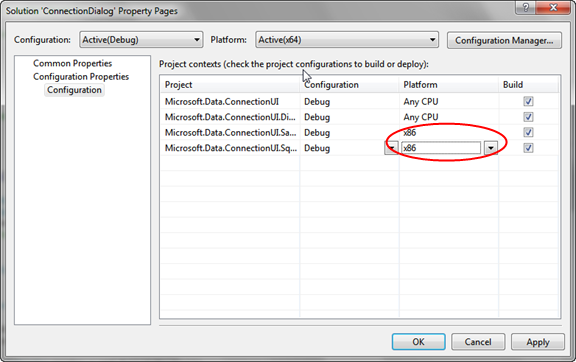
## Known Issues

These are two issues that may happen on an x64 platform machine using SQL CE data source.

1. When choosing SQL CE data source and specifying a local database file, clicking “Test Connection” may pop up following error message:

*"Unable to load DLL 'sqlceme35.dll': The specified module could not be found. (Exception from HRESULT: 0x8007007E)"*

This specific error happens because the version of SQL CE is not compatible with the host application itself. To resolve this issue, you can change the solution target platform configuration. For example, assuming the SQL CE is x86 version, following configuration can make the application run on an x64 platform:



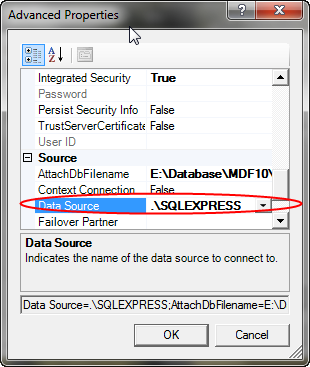
This issue will only appear when using SQL CE data provider. For more detailed information see the following article:

* + [Can’t find P/Invoke DLL sqlcemeNN.dll](http://blogs.msdn.com/sqlservercompact/archive/2007/10/26/can-t-find-p-invoke-dll-sqlcemenn-dll.aspx)

1. When data source is set to Microsoft SQL Server Database File (SqlClient) and has a .mdf extension file, clicking “Test Connection” may display the following error message:

*A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: Named Pipes Provider, error: 40 - Could not open a connection to SQL Server)*

This failure is because the “Data Source” property of the provider is set to empty instead of the default SQL Server instance installed on the target machine. In order to fix this issue, users can set the data source value in “***Advanced Properties***” dialog manually:



This issue may happen because of a contradiction between the application and the SQL Server version. For example, in an x64 OS machine, in order to fix the issue 1 mentioned above, the application target platform is set to x86 while the SQL Server installed is x64.