

Contents

Overview

Tutorials

- Create a simple data app by using ADO.NET

- Create a simple data app by using WPF and Entity Framework

- Connect to data from a C++ app

- Create LINQ to SQL classes using O/R Designer

- Create an n-tier data application

Concepts

- Compatible database systems

- Visual Studio data tools for .NET

- Entity Framework tools

DataSets

- DataSet tools

- Typed vs. untyped DataSets

- TableAdapters overview

- Work with DataSets in n-tier applications

- Database projects and data-tier applications (DAC)

- N-tier data applications overview

How-to guides

- Create a database

Connections

- Add connections

- Save and edit connection strings

- Connect to data in an Access database

- Add .NET data sources

LINQ to SQL tools

- Overview

- Change the return type of a DataContext method

- Create DataContext methods mapped to stored procedures

Configure inheritance by using the O/R Designer

Create LINQ to SQL classes mapped to tables and views

Extend Code Generated by the O/R Designer

Create an association between LINQ to SQL classes

Add validation to entity classes

Customize the insert, update, and delete behavior of entity classes

Assign stored procedures to perform updates, inserts, and deletes

Turn pluralization on and off

Use DataSet tools

Create DataSets

Create and configure DataSets

Create relationships between DataSets

Create a DataSet by using the DataSet Designer

Create a DataTable by using the DataSet Designer

Configure TableAdapters

Create and configure TableAdapters

Create parameterized TableAdapter queries

Directly access the database with a TableAdapter

Turn off constraints while filling a DataSet

Extend the functionality of a TableAdapter

Read XML data into a DataSet

Edit data in DataSets

Validate data in DataSets

Save data back to the database

Overview

Insert new records into a database

Update data by using a TableAdapter

Execute a hierarchical update

Handle a concurrency exception

Transactions

Save data by using a transaction

Walkthrough: Save data in a transaction

- Save data to a database (multiple tables)
- Save data from an object to a database
- Save data with the TableAdapter DBDirect methods
- Save a DataSet as XML

Query DataSets

N-tier applications

- Add validation to an n-tier DataSet
- Add code to DataSets in n-tier applications
- Add code to TableAdapters in n-tier applications
- Separate DataSets and TableAdapters into different projects

Controls

- Bind controls to data sources
- Set the control to be created when dragging from the Data Sources window
- Add custom controls to the Data Sources window

WPF controls

- Bind WPF controls to data
- Bind WPF controls to a DataSet
- Bind WPF controls to a WCF data service
- Create lookup tables
- Display related data
- Bind controls to pictures from a database

Windows Forms controls

- Bind Windows Forms controls to data
- Filter and sort data in a Windows Forms application
- Commit in-process edits on data-bound controls before saving data
- Create lookup tables
- Create a Windows Form to search data
- Create a user control that supports simple data binding
- Create a user control that supports complex data binding
- Create a user control that supports lookup data binding
- Pass data between forms

Bind custom objects

Customize how Visual Studio creates captions for data-bound controls

Windows Communication Foundation Services and WCF Data Services

Overview of WCF in Visual Studio

Work with a conceptual model

Connect to data in a service

Create a WCF Service in Windows Forms

Create a WCF Data Service with WPF and Entity Framework

Troubleshoot service references

Configure service reference dialog box

Add, update, or remove a WCF Data Service reference

Upgrade .mdf files

Reference

O/R Designer (Linq to SQL)

DataContext methods

Data class inheritance

O/R Designer messages

The selected class cannot be deleted because it is used as a return type for one or more DataContext methods

The connection string contains credentials with a clear text password and is not using integrated security

The property <property name> cannot be deleted because it is participating in the association <association name>

The property <property name> cannot be deleted

The connection property in the Application Settings file is missing or incorrect

Cannot create an association <association name> - property types do not match

Warning. Changes have been made to the Configure Behavior dialog box that have not been applied

You have selected a database object from an unsupported database provider

This related method is the backing method for the following default insert, update, or delete methods

One or more selected items contain a data type that is not supported by the designer

Changing the return type of a DataContext method cannot be undone

The designer cannot be modified while debugging

The selected connection uses an unsupported database provider

The objects you are adding to the designer use a different data connection than the designer is currently using

Cannot create an association <association name> - property listed twice

Could not retrieve schema information for database object

One or more selected database objects return a schema that does not match the schema of the target class

Work with data in Visual Studio

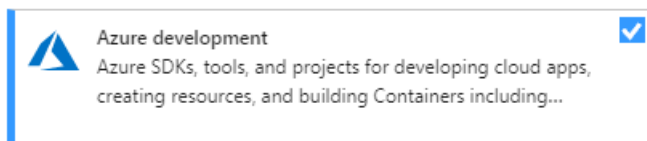
8/5/2021 • 7 minutes to read • [Edit Online](#)

In Visual Studio, you can create applications that connect to data in virtually any database product or service, in any format, anywhere—on a local machine, on a local area network, or in a public, private, or hybrid cloud.

For applications in JavaScript, Python, PHP, Ruby, or C++, you connect to data like you do anything else, by obtaining libraries and writing code. For .NET applications, Visual Studio provides tools that you can use to explore data sources, create object models to store and manipulate data in memory, and bind data to the user interface. Microsoft Azure provides SDKs for .NET, Java, Node.js, PHP, Python, Ruby, and mobile apps, and tools in Visual Studio for connecting to Azure Storage.

The following lists show just a few of the many database and storage systems that can be used from Visual Studio. The [Microsoft Azure](#) offerings are data services that include all provisioning and administration of the underlying data store. The **Azure development** workload in [Visual Studio 2017](#) enables you to work with Azure data stores directly from Visual Studio.

The following lists show just a few of the many database and storage systems that can be used from Visual Studio. The [Microsoft Azure](#) offerings are data services that include all provisioning and administration of the underlying data store. The **Azure development** workload in [Visual Studio 2019](#) enables you to work with Azure data stores directly from Visual Studio.



Most of the other SQL and NoSQL database products that are listed here can be hosted on a local machine, on a local network, or in Microsoft Azure on a virtual machine. If you host the database in a Microsoft Azure virtual machine, you're responsible for managing the database itself.

Microsoft Azure

- SQL Database
- Azure Cosmos DB
- Storage (blobs, tables, queues, files)
- SQL Data Warehouse
- SQL Server Stretch Database
- StorSimple
- And more...

SQL

- SQL Server 2005-2016 (includes Express and LocalDB)
- Firebird
- MariaDB
- MySQL
- Oracle
- PostgreSQL
- SQLite
- And more...

NoSQL

- Apache Cassandra
- CouchDB
- MongoDB
- NDatabase
- OrientDB
- RavenDB
- VelocityDB
- And more...

Many database vendors and third parties support Visual Studio integration by NuGet packages. You can explore the offerings on nuget.org or through the NuGet Package Manager in Visual Studio (**Tools** > **NuGet Package Manager** > **Manage NuGet Packages for Solution**). Other database products integrate with Visual Studio as an extension. You can browse these offerings in the [Visual Studio Marketplace](#) or by navigating to **Tools** > **Extensions and Updates** and then selecting **Online** in the left pane of the dialog box. For more information, see [Compatible database systems for Visual Studio](#).

Many database vendors and third parties support Visual Studio integration by NuGet packages. You can explore the offerings on nuget.org or through the NuGet Package Manager in Visual Studio (**Tools** > **NuGet Package Manager** > **Manage NuGet Packages for Solution**). Other database products integrate with Visual Studio as an extension. You can browse these offerings in the [Visual Studio Marketplace](#) or by navigating to **Extensions** > **Manage Extensions** and then selecting **Online** in the left pane of the dialog box. For more information, see [Compatible database systems for Visual Studio](#).

NOTE

Extended support for SQL Server 2005 ended on April 12, 2016. There is no guarantee that data tools in Visual Studio 2015 and later will continue to work with SQL Server 2005. For more information, see the [end-of-support announcement for SQL Server 2005](#).

.NET languages

All .NET data access, including in .NET Core, is based on ADO.NET, a set of classes that defines an interface for accessing any kind of data source, both relational and non-relational. Visual Studio has several tools and designers that work with ADO.NET to help you connect to databases, manipulate the data, and present the data to the user. The documentation in this section describes how to use those tools. You can also program directly against the ADO.NET command objects. For more information about calling the ADO.NET APIs directly, see [ADO.NET](#).

For data-access documentation related to ASP.NET, see [Working with Data](#) on the ASP.NET site. For a tutorial on using Entity Framework with ASP.NET MVC, see [Getting Started with Entity Framework 6 Code First using MVC 5](#).

Universal Windows Platform (UWP) apps in C# or Visual Basic can use the Microsoft Azure SDK for .NET to access Azure Storage and other Azure services. The `Windows.Web.HttpClient` class enables communication with any RESTful service. For more information, see [How to connect to an HTTP server using Windows.Web.Http](#).

For data storage on the local machine, the recommended approach is to use SQLite, which runs in the same process as the app. If an object-relational mapping (ORM) layer is required, you can use Entity Framework. For more information, see [Data access](#) in the Windows Developer Center.

If you are connecting to Azure services, be sure to download the latest [Azure SDK tools](#).

Data providers

For a database to be consumable in ADO.NET, it must have a custom *ADO.NET data provider* or else must expose an ODBC or OLE DB interface. Microsoft provides a [list of ADO.NET data providers](#) for SQL Server products, as well as ODBC and OLE DB providers.

Data modeling

In .NET, you have three choices for modeling and manipulating data in memory after you have retrieved it from a data source:

Entity Framework The preferred Microsoft ORM technology. You can use it to program against relational data as first-class .NET objects. For new applications, it should be the default first choice when a model is required. It requires custom support from the underlying ADO.NET provider.

LINQ to SQL An earlier-generation object-relational mapper. It works well for less complex scenarios but is no longer in active development.

Datasets The oldest of the three modeling technologies. It is designed primarily for rapid development of "forms over data" applications in which you are not processing huge amounts of data or performing complex queries or transformations. A DataSet object consists of DataTable and DataRow objects that logically resemble SQL database objects much more than .NET objects. For relatively simple applications based on SQL data sources, datasets might still be a good choice.

There is no requirement to use any of these technologies. In some scenarios, especially where performance is critical, you can simply use a DataReader object to read from the database and copy the values that you need into a collection object such as List<T>.

Native C++

C++ applications that connect to SQL Server should use the [Microsoft® ODBC Driver 13.1 for SQL Server](#) in most cases. If the servers are linked, then OLE DB is necessary and for that you use the [SQL Server Native Client](#). You can access other databases by using [ODBC](#) or OLE DB drivers directly. ODBC is the current standard database interface, but most database systems provide custom functionality that can't be accessed through the ODBC interface. OLE DB is a legacy COM data-access technology that is still supported but not recommended for new applications. For more information, see [Data Access in Visual C++](#).

C++ programs that consume REST services can use the [C++ REST SDK](#).

C++ programs that work with Microsoft Azure Storage can use the [Microsoft Azure Storage Client](#).

Data modeling—Visual Studio does not provide an ORM layer for C++. [ODB](#) is a popular open-source ORM for C++.

To learn more about connecting to databases from C++ apps, see [Visual Studio data tools for C++](#). For more information about legacy Visual C++ data-access technologies, see [Data Access](#).

JavaScript

JavaScript in Visual Studio is a first-class language for building cross-platform apps, UWP apps, cloud services, websites, and web apps. You can use Bower, Grunt, Gulp, npm, and NuGet from within Visual Studio to install your favorite JavaScript libraries and database products. Connect to Azure storage and services by downloading SDKs from the [Azure website](#). Edge.js is a library that connects server-side JavaScript (Node.js) to ADO.NET data sources.

Python

Install [Python support in Visual Studio](#) to create Python applications. The Azure documentation has several

tutorials on connecting to data, including the following:

- [Django and SQL Database on Azure](#)
- [Django and MySQL on Azure](#)
- Work with [blobs](#), [files](#), [queues](#), and [tables \(Cosmo DB\)](#).

Related topics

[Microsoft AI platform](#)—Provides an introduction to the Microsoft intelligent cloud, including Cortana Analytics Suite and support for Internet of Things.

[Microsoft Azure Storage](#)—Describes Azure Storage, and how to create applications by using Azure blobs, tables, queues, and files.

[Azure SQL Database](#)—Describes how to connect to Azure SQL Database, a relational database as a service.

[SQL Server Data Tools](#)—Describes the tools that simplify design, exploration, testing, and deploying of data-connected applications and databases.

[ADO.NET](#)—Describes the ADO.NET architecture and how to use the ADO.NET classes to manage application data and interact with data sources and XML.

[ADO.NET Entity Framework](#)—Describes how to create data applications that allow developers to program against a conceptual model instead of directly against a relational database.

[WCF Data Services 4.5](#)—Describes how to use WCF Data Services to deploy data services on the web or an intranet that implement the [Open Data Protocol \(OData\)](#).

[Data in Office Solutions](#)—Contains links to topics that explain how data works in Office solutions. This includes information about schema-oriented programming, data caching, and server-side data access.

[LINQ \(Language-Integrated Query\)](#)—Describes the query capabilities built into C# and Visual Basic, and the common model for querying relational databases, XML documents, datasets, and in-memory collections.

[XML Tools in Visual Studio](#)—Discusses working with XML data, debugging XSLT, .NET XML features, and the architecture of XML Query.

[XML Documents and Data](#)—Provides an overview to a comprehensive and integrated set of classes that work with XML documents and data in .NET.

Create a simple data application by using ADO.NET

8/5/2021 • 19 minutes to read • [Edit Online](#)

When you create an application that manipulates data in a database, you perform basic tasks such as defining connection strings, inserting data, and running stored procedures. By following this topic, you can discover how to interact with a database from within a simple Windows Forms "forms over data" application by using Visual C# or Visual Basic and ADO.NET. All .NET data technologies—including datasets, LINQ to SQL, and Entity Framework—ultimately perform steps that are very similar to those shown in this article.

This article demonstrates a simple way to get data out of a database in a fast manner. If your application needs to modify data in non-trivial ways and update the database, you should consider using Entity Framework and using data binding to automatically sync user interface controls to changes in the underlying data.

IMPORTANT

To keep the code simple, it doesn't include production-ready exception handling.

Prerequisites

To create the application, you'll need:

- Visual Studio.
- SQL Server Express LocalDB. If you don't have SQL Server Express LocalDB, you can install it from the [SQL Server Express download page](#).

This topic assumes that you're familiar with the basic functionality of the Visual Studio IDE and can create a Windows Forms application, add forms to the project, put buttons and other controls on the forms, set properties of the controls, and code simple events. If you aren't comfortable with these tasks, we suggest that you complete the [Getting started with Visual C# and Visual Basic](#) topic before you start this walkthrough.

Set up the sample database

Create the sample database by following these steps:

1. In Visual Studio, open the **Server Explorer** window.
2. Right-click on **Data Connections** and choose **Create New SQL Server Database**.
3. In the **Server name** text box, enter **(localdb)\mssqllocaldb**.
4. In the **New database name** text box, enter **Sales**, then choose **OK**.

The empty **Sales** database is created and added to the Data Connections node in Server Explorer.

5. Right-click on the **Sales** data connection and select **New Query**.

A query editor window opens.

6. Copy the [Sales Transact-SQL script](#) to your clipboard.
7. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes running and the database objects are created. The database contains

two tables: Customer and Orders. These tables contain no data initially, but you can add data when you run the application that you'll create. The database also contains four simple stored procedures.

Create the forms and add controls

1. Create a project for a Windows Forms application, and then name it **SimpleDataApp**.

Visual Studio creates the project and several files, including an empty Windows form that's named **Form1**.

2. Add two Windows forms to your project so that it has three forms, and then give them the following names:

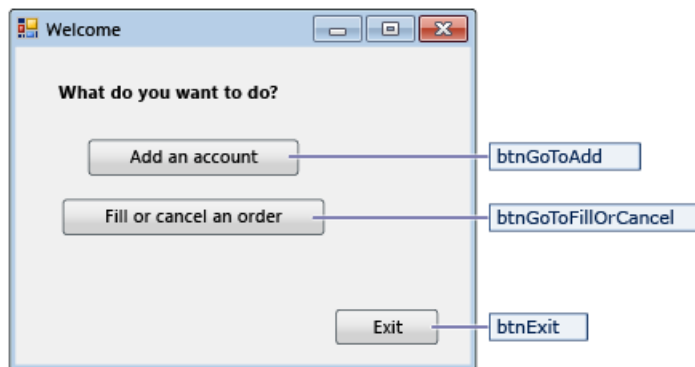
- **Navigation**
- **NewCustomer**
- **FillOrCancel**

3. For each form, add the text boxes, buttons, and other controls that appear in the following illustrations. For each control, set the properties that the tables describe.

NOTE

The group box and the label controls add clarity but aren't used in the code.

Navigation form

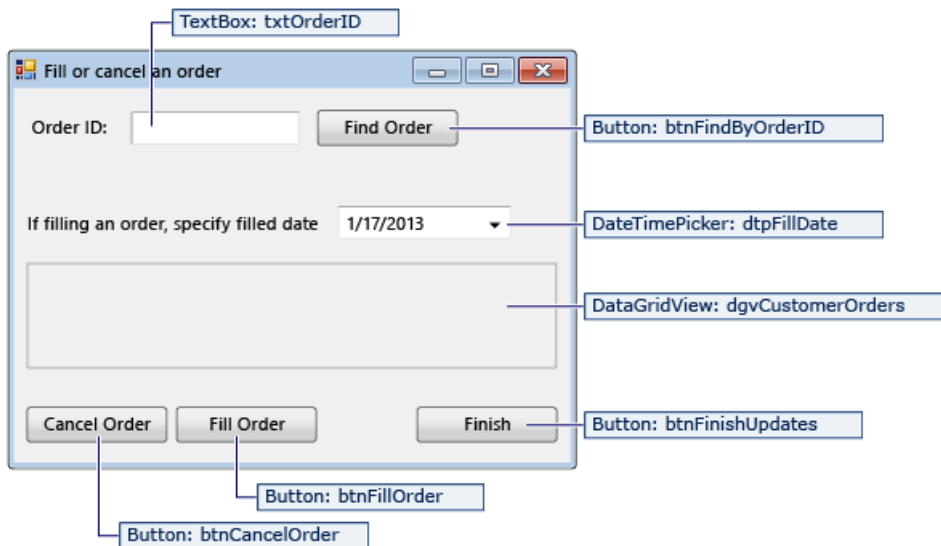


| CONTROLS FOR THE NAVIGATION FORM | PROPERTIES |
|----------------------------------|----------------------------|
| Button | Name = btnGoToAdd |
| Button | Name = btnGoToFillOrCancel |
| Button | Name = btnExit |

NewCustomer form

| CONTROLS FOR THE NEWCUSTOMER FORM | PROPERTIES |
|-----------------------------------|--|
| TextBox | Name = txtCustomerName |
| TextBox | Name = txtCustomerID ReadOnly = True |
| Button | Name = btnCreateAccount |
| NumericUpDown | DecimalPlaces = 0 Maximum = 5000 Name = numOrderAmount |
| DateTimePicker | Format = Short Name = dtpOrderDate |
| Button | Name = btnPlaceOrder |
| Button | Name = btnAddAnotherAccount |
| Button | Name = btnAddFinish |

FillOrCancel form



| CONTROLS FOR THE FILLORCANCEL FORM | PROPERTIES |
|------------------------------------|--|
| TextBox | Name = txtOrderID |
| Button | Name = btnFindByOrderID |
| DateTimePicker | Format = Short Name = dtpFillDate |
| DataGridView | Name = dgvCustomerOrders Readonly = True RowHeadersVisible = False |
| Button | Name = btnCancelOrder |
| Button | Name = btnFillOrder |
| Button | Name = btnFinishUpdates |

Store the connection string

When your application tries to open a connection to the database, your application must have access to the connection string. To avoid entering the string manually on each form, store the string in the *App.config* file in your project, and create a method that returns the string when the method is called from any form in your application.

You can find the connection string by right-clicking on the **Sales** data connection in **Server Explorer** and choosing **Properties**. Locate the **ConnectionString** property, then use **Ctrl+A**, **Ctrl+C** to select and copy the string to the clipboard.

1. If you're using C#, in **Solution Explorer**, expand the **Properties** node under the project, and then open the **Settings.settings** file. If you're using Visual Basic, in **Solution Explorer**, click **Show All Files**, expand the **My Project** node, and then open the **Settings.settings** file.
2. In the **Name** column, enter `connString`.
3. In the **Type** list, select **(Connection String)**.

4. In the **Scope** list, select **Application**.
5. In the **Value** column, enter your connection string (without any outside quotes), and then save your changes.

NOTE

In a real application, you should store the connection string securely, as described in [Connection strings and configuration files](#).

Write the code for the forms

This section contains brief overviews of what each form does. It also provides the code that defines the underlying logic when a button on the form is clicked.

Navigation form

The Navigation form opens when you run the application. The **Add an account** button opens the NewCustomer form. The **Fill or cancel orders** button opens the FillOrCancel form. The **Exit** button closes the application.

Make the Navigation form the startup form

If you're using C#, in **Solution Explorer**, open **Program.cs**, and then change the `Application.Run` line to this:

```
Application.Run(new Navigation());
```

If you're using Visual Basic, in **Solution Explorer**, open the **Properties** window, select the **Application** tab, and then select **SimpleDataApp.Navigation** in the **Startup form** list.

Create auto-generated event handlers

Double-click the three buttons on the Navigation form to create empty event handler methods. Double-clicking the buttons also adds auto-generated code in the Designer code file that enables a button click to raise an event.

Add code for the Navigation form logic

In the code page for the Navigation form, complete the method bodies for the three button click event handlers as shown in the following code.

```

/// <summary>
/// Opens the NewCustomer form as a dialog box,
/// which returns focus to the calling form when it is closed.
/// </summary>
private void btnGoToAdd_Click(object sender, EventArgs e)
{
    Form frm = new NewCustomer();
    frm.Show();
}

/// <summary>
/// Opens the FillOrCancel form as a dialog box.
/// </summary>
private void btnGoToFillOrCancel_Click(object sender, EventArgs e)
{
    Form frm = new FillOrCancel();
    frm.ShowDialog();
}

/// <summary>
/// Closes the application (not just the Navigation form).
/// </summary>
private void btnExit_Click(object sender, EventArgs e)
{
    this.Close();
}

```

```

''' <summary>
''' Opens the NewCustomer form as a dialog box, which returns focus to the calling form when it closes.
''' </summary>
Private Sub btnGoToAdd_Click(sender As Object, e As EventArgs) Handles btnGoToAdd.Click
    Dim frm As Form = New NewCustomer()
    frm.Show()
End Sub

''' <summary>
''' Opens the FillOrCancel form as a dialog box.
''' </summary>
Private Sub btnGoToFillOrCancel_Click(sender As Object, e As EventArgs) Handles btnGoToFillOrCancel.Click
    Dim frm As Form = New FillOrCancel()
    frm.ShowDialog()
End Sub

''' <summary>
''' Closes the application (not just the navigation form).
''' </summary>
Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
    Me.Close()
End Sub

```

NewCustomer form

When you enter a customer name and then select the **Create Account** button, the NewCustomer form creates a customer account, and SQL Server returns an IDENTITY value as the new customer ID. You can then place an order for the new account by specifying an amount and an order date and selecting the **Place Order** button.

Create auto-generated event handlers

Create an empty Click event handler for each button on the NewCustomer form by double-clicking on each of the four buttons. Double-clicking the buttons also adds auto-generated code in the Designer code file that enables a button click to raise an event.

Add code for the NewCustomer form logic

To complete the NewCustomer form logic, follow these steps.

1. Bring the `System.Data.SqlClient` namespace into scope so that you don't have to fully qualify the names of its members.

```
using System.Data.SqlClient;
```

```
Imports System.Data.SqlClient
```

2. Add some variables and helper methods to the class as shown in the following code.


```

// Storage for IDENTITY values returned from database.
private int parsedCustomerID;
private int orderID;

/// <summary>
/// Verifies that the customer name text box is not empty.
/// </summary>
private bool IsCustomerNameValid()
{
    if (txtCustomerName.Text == "")
    {
        MessageBox.Show("Please enter a name.");
        return false;
    }
    else
    {
        return true;
    }
}

/// <summary>
/// Verifies that a customer ID and order amount have been provided.
/// </summary>
private bool IsOrderDataValid()
{
    // Verify that CustomerID is present.
    if (txtCustomerID.Text == "")
    {
        MessageBox.Show("Please create customer account before placing order.");
        return false;
    }
    // Verify that Amount isn't 0.
    else if ((numOrderAmount.Value < 1))
    {
        MessageBox.Show("Please specify an order amount.");
        return false;
    }
    else
    {
        // Order can be submitted.
        return true;
    }
}

/// <summary>
/// Clears the form data.
/// </summary>
private void ClearForm()
{
    txtCustomerName.Clear();
    txtCustomerID.Clear();
    dtpOrderDate.Value = DateTime.Now;
    numOrderAmount.Value = 0;
    this.parsedCustomerID = 0;
}

```

```

' Storage for ID values returned from the database.
Private parsedCustomerID As Integer
Private orderID As Integer

''' <summary>
''' Verifies that the customer name text box is not empty.
''' </summary>
Private ReadOnly Property IsCustomerNameValid As Boolean
    Get
        If txtCustomerName.Text = "" Then
            MessageBox.Show("Please enter a name.")
            Return False
        Else
            Return True
        End If
    End Get
End Property

''' <summary>
''' Verifies the order data is valid.
''' </summary>
Private Function IsOrderDataValid() As Boolean

    ' Verify that CustomerID is present.
    If txtCustomerID.Text = "" Then
        MessageBox.Show("Please create a customer account before placing order.")
        Return False

        ' Verify that order amount isn't 0.
    ElseIf (numOrderAmount.Value < 1) Then
        MessageBox.Show("Please specify an order amount.")
        Return False
    Else
        ' Order can be submitted.
        Return True
    End If
End Function

''' <summary>
''' Clears values from controls.
''' </summary>
Private Sub ClearForm()
    txtCustomerName.Clear()
    txtCustomerID.Clear()
    dtpOrderDate.Value = DateTime.Now
    numOrderAmount.Value = 0
    Me.parsedCustomerID = 0
End Sub

```

3. Complete the method bodies for the four button click event handlers as shown in the following code.

```

''' <summary>
''' Creates a new customer by calling the Sales.uspNewCustomer stored procedure.
''' </summary>
private void btnCreateAccount_Click(object sender, EventArgs e)
{
    if (IsCustomerNameValid())
    {
        // Create the connection.
        using (SqlConnection connection = new SqlConnection(Properties.Settings.Default.connString))
        {
            // Create a SqlCommand, and identify it as a stored procedure.
            using (SqlCommand sqlCommand = new SqlCommand("Sales.uspNewCustomer", connection))
            {
                sqlCommand.CommandType = CommandType.StoredProcedure;
            }
        }
    }
}

```

```

        // Add input parameter for the stored procedure and specify what to use as its value.
        sqlCommand.Parameters.Add(new SqlParameter("@CustomerName", SqlDbType.NVarChar, 40));
        sqlCommand.Parameters["@CustomerName"].Value = txtCustomerName.Text;

        // Add the output parameter.
        sqlCommand.Parameters.Add(new SqlParameter("@CustomerID", SqlDbType.Int));
        sqlCommand.Parameters["@CustomerID"].Direction = ParameterDirection.Output;

        try
        {
            connection.Open();

            // Run the stored procedure.
            sqlCommand.ExecuteNonQuery();

            // Customer ID is an IDENTITY value from the database.
            this.parsedCustomerID = (int)sqlCommand.Parameters["@CustomerID"].Value;

            // Put the Customer ID value into the read-only text box.
            this.txtCustomerID.Text = Convert.ToString(parsedCustomerID);
        }
        catch
        {
            MessageBox.Show("Customer ID was not returned. Account could not be created.");
        }
        finally
        {
            connection.Close();
        }
    }
}

/// <summary>
/// Calls the Sales.uspPlaceNewOrder stored procedure to place an order.
/// </summary>
private void btnPlaceOrder_Click(object sender, EventArgs e)
{
    // Ensure the required input is present.
    if (IsOrderDataValid())
    {
        // Create the connection.
        using (SqlConnection connection = new SqlConnection(Properties.Settings.Default.connString))
        {
            // Create SqlCommand and identify it as a stored procedure.
            using (SqlCommand sqlCommand = new SqlCommand("Sales.uspPlaceNewOrder", connection))
            {
                sqlCommand.CommandType = CommandType.StoredProcedure;

                // Add the @CustomerID input parameter, which was obtained from uspNewCustomer.
                sqlCommand.Parameters.Add(new SqlParameter("@CustomerID", SqlDbType.Int));
                sqlCommand.Parameters["@CustomerID"].Value = this.parsedCustomerID;

                // Add the @OrderDate input parameter.
                sqlCommand.Parameters.Add(new SqlParameter("@OrderDate", SqlDbType.DateTime, 8));
                sqlCommand.Parameters["@OrderDate"].Value = dtpOrderDate.Value;

                // Add the @Amount order amount input parameter.
                sqlCommand.Parameters.Add(new SqlParameter("@Amount", SqlDbType.Int));
                sqlCommand.Parameters["@Amount"].Value = numOrderAmount.Value;

                // Add the @Status order status input parameter.
                // For a new order, the status is always 0 (open).
                sqlCommand.Parameters.Add(new SqlParameter("@Status", SqlDbType.Char, 1));
                sqlCommand.Parameters["@Status"].Value = "0";

                // Add the return value for the stored procedure, which is the order ID.
                sqlCommand.Parameters.Add(new SqlParameter("@RC", SqlDbType.Int));
            }
        }
    }
}

```

```

        sqlCommand.Parameters["@RC"].Direction = ParameterDirection.ReturnValue;

        try
        {
            //Open connection.
            connection.Open();

            // Run the stored procedure.
            sqlCommand.ExecuteNonQuery();

            // Display the order number.
            this.orderID = (int)sqlCommand.Parameters["@RC"].Value;
            MessageBox.Show("Order number " + this.orderID + " has been submitted.");
        }
        catch
        {
            MessageBox.Show("Order could not be placed.");
        }
        finally
        {
            connection.Close();
        }
    }
}

/// <summary>
/// Clears the form data so another new account can be created.
/// </summary>
private void btnAddAnotherAccount_Click(object sender, EventArgs e)
{
    this.ClearForm();
}

/// <summary>
/// Closes the form/dialog box.
/// </summary>
private void btnAddFinish_Click(object sender, EventArgs e)
{
    this.Close();
}

```

```

''' <summary>
''' Creates a new account by executing the Sales.uspNewCustomer
''' stored procedure on the database.
''' </summary>
Private Sub btnCreateAccount_Click(sender As Object, e As EventArgs) Handles btnCreateAccount.Click

    ' Ensure a customer name has been entered.
    If IsCustomerNameValid Then

        ' Create the SqlConnection object.
        Using connection As New SqlConnection(My.Settings.connString)

            ' Create a SqlCommand, and identify the command type as a stored procedure.
            Using sqlCommand As New SqlCommand("Sales.uspNewCustomer", connection)
                sqlCommand.CommandType = CommandType.StoredProcedure

                ' Add the customer name input parameter for the stored procedure.
                sqlCommand.Parameters.Add(New SqlParameter("@CustomerName", SqlDbType.NVarChar, 40))
                sqlCommand.Parameters("@CustomerName").Value = txtCustomerName.Text

                ' Add the customer ID output parameter.
                sqlCommand.Parameters.Add(New SqlParameter("@CustomerID", SqlDbType.Int))
                sqlCommand.Parameters("@CustomerID").Direction = ParameterDirection.Output
            End Using
        End Using
    End If
End Sub

```

```

Try
    ' Open the connection.
    connection.Open()

    ' Run the stored procedure.
    sqlCommand.ExecuteNonQuery()

    ' Convert the Customer ID value to an Integer.
    Me.parsedCustomerID = CInt(sqlCommand.Parameters("@CustomerID").Value)

    ' Insert the customer ID into the corresponding text box.
    Me.txtCustomerID.Text = Convert.ToString(parsedCustomerID)
Catch
    MessageBox.Show("Customer ID was not returned. Account could not be created.")
Finally
    ' Close the connection.
    connection.Close()
End Try
End Using
End Using
End If
End Sub

''' <summary>
''' Places the order by executing the Sales.uspPlaceNewOrder
''' stored procedure on the database.
''' </summary>
Private Sub btnPlaceOrder_Click(sender As Object, e As EventArgs) Handles btnPlaceOrder.Click

    If IsOrderDataValid() Then

        ' Create the connection.
        Using connection As New SqlConnection(My.Settings.connString)

            ' Create SqlCommand and identify it as a stored procedure.
            Using sqlCommand As New SqlCommand("Sales.uspPlaceNewOrder", connection)
                sqlCommand.CommandType = CommandType.StoredProcedure

                ' Add the @CustomerID parameter, which was an output parameter from uspNewCustomer.
                sqlCommand.Parameters.Add(New SqlParameter("@CustomerID", SqlDbType.Int))
                sqlCommand.Parameters("@CustomerID").Value = Me.parsedCustomerID

                ' Add the @OrderDate parameter.
                sqlCommand.Parameters.Add(New SqlParameter("@OrderDate", SqlDbType.DateTime, 8))
                sqlCommand.Parameters("@OrderDate").Value = dtpOrderDate.Value

                ' Add the @Amount parameter.
                sqlCommand.Parameters.Add(New SqlParameter("@Amount", SqlDbType.Int))
                sqlCommand.Parameters("@Amount").Value = numOrderAmount.Value

                ' Add the @Status parameter. For a new order, the status is always 0 (open).
                sqlCommand.Parameters.Add(New SqlParameter("@Status", SqlDbType.[Char], 1))
                sqlCommand.Parameters("@Status").Value = "0"

                ' Add a return value parameter for the stored procedure, which is the orderID.
                sqlCommand.Parameters.Add(New SqlParameter("@RC", SqlDbType.Int))
                sqlCommand.Parameters("@RC").Direction = ParameterDirection.ReturnValue

            Try
                ' Open connection.
                connection.Open()

                ' Run the stored procedure.
                sqlCommand.ExecuteNonQuery()

                ' Display the order number.
                Me.orderID = CInt(sqlCommand.Parameters("@RC").Value)
                MessageBox.Show("Order number " & (Me.orderID).ToString & " has been submitted.")
            Catch

```

```

        ' A simple catch.
        MessageBox.Show("Order could not be placed.")
    Finally
        ' Always close a connection after you finish using it,
        ' so that it can be released to the connection pool.
        connection.Close()
    End Try
End Using
End Using
End If
End Sub

''' <summary>
''' Resets the form for another new account.
''' </summary>
Private Sub btnAddAnotherAccount_Click(sender As Object, e As EventArgs) Handles
btnAddAnotherAccount.Click
    Me.ClearForm()
End Sub

''' <summary>
''' Closes the NewCustomer form and returns focus to the Navigation form.
''' </summary>
Private Sub btnAddFinish_Click(sender As Object, e As EventArgs) Handles btnAddFinish.Click
    Me.Close()
End Sub

```

FillOrCancel form

The FillOrCancel form runs a query to return an order when you enter an order ID and then click the **Find Order** button. The returned row appears in a read-only data grid. You can mark the order as canceled (X) if you select the **Cancel Order** button, or you can mark the order as filled (F) if you select the **Fill Order** button. If you select the **Find Order** button again, the updated row appears.

Create auto-generated event handlers

Create empty Click event handlers for the four buttons on the FillOrCancel form by double-clicking the buttons. Double-clicking the buttons also adds auto-generated code in the Designer code file that enables a button click to raise an event.

Add code for the FillOrCancel form logic

To complete the FillOrCancel form logic, follow these steps.

1. Bring the following two namespaces into scope so that you don't have to fully qualify the names of their members.

```

using System.Data.SqlClient;
using System.Text.RegularExpressions;

```

```

Imports System.Data.SqlClient
Imports System.Text.RegularExpressions

```

2. Add a variable and helper method to the class as shown in the following code.

```

// Storage for the order ID value.
private int parsedOrderID;

/// <summary>
/// Verifies that an order ID is present and contains valid characters.
/// </summary>
private bool IsOrderIDValid()
{
    // Check for input in the Order ID text box.
    if (txtOrderID.Text == "")
    {
        MessageBox.Show("Please specify the Order ID.");
        return false;
    }

    // Check for characters other than integers.
    else if (Regex.IsMatch(txtOrderID.Text, @"^\d*$"))
    {
        // Show message and clear input.
        MessageBox.Show("Customer ID must contain only numbers.");
        txtOrderID.Clear();
        return false;
    }
    else
    {
        // Convert the text in the text box to an integer to send to the database.
        parsedOrderID = Int32.Parse(txtOrderID.Text);
        return true;
    }
}

```

```

' Storage for OrderID.
Private parsedOrderID As Integer

''' <summary>
''' Verifies that OrderID is valid.
''' </summary>
Private Function IsOrderIDValid() As Boolean

    ' Check for input in the Order ID text box.
    If txtOrderID.Text = "" Then
        MessageBox.Show("Please specify the Order ID.")
        Return False

    ' Check for characters other than integers.
    ElseIf Regex.IsMatch(txtOrderID.Text, "^\d*$") Then
        ' Show message and clear input.
        MessageBox.Show("Please specify integers only.")
        txtOrderID.Clear()
        Return False
    Else
        ' Convert the text in the text box to an integer to send to the database.
        parsedOrderID = Int32.Parse(txtOrderID.Text)
        Return True
    End If
End Function

```

3. Complete the method bodies for the four button click event handlers as shown in the following code.

```

/// <summary>
/// Executes a t-SQL SELECT statement to obtain order data for a specified
/// order ID, then displays it in the DataGridView on the form.
/// </summary>
private void btnFindByOrderID_Click(object sender, EventArgs e)

```

```

{
    if (IsOrderIDValid())
    {
        using (SqlConnection connection = new SqlConnection(Properties.Settings.Default.connString))
        {
            // Define a t-SQL query string that has a parameter for orderID.
            const string sql = "SELECT * FROM Sales.Orders WHERE orderID = @orderID";

            // Create a SqlCommand object.
            using (SqlCommand sqlCommand = new SqlCommand(sql, connection))
            {
                // Define the @orderID parameter and set its value.
                sqlCommand.Parameters.Add(new SqlParameter("@orderID", SqlDbType.Int));
                sqlCommand.Parameters["@orderID"].Value = parsedOrderID;

                try
                {
                    connection.Open();

                    // Run the query by calling ExecuteReader().
                    using (SqlDataReader dataReader = sqlCommand.ExecuteReader())
                    {
                        // Create a data table to hold the retrieved data.
                        DataTable dataTable = new DataTable();

                        // Load the data from SqlDataReader into the data table.
                        dataTable.Load(dataReader);

                        // Display the data from the data table in the data grid view.
                        this.dgvCustomerOrders.DataSource = dataTable;

                        // Close the SqlDataReader.
                        dataReader.Close();
                    }
                }
                catch
                {
                    MessageBox.Show("The requested order could not be loaded into the form.");
                }
                finally
                {
                    // Close the connection.
                    connection.Close();
                }
            }
        }
    }
}

/// <summary>
/// Cancels an order by calling the Sales.uspCancelOrder
/// stored procedure on the database.
/// </summary>
private void btnCancelOrder_Click(object sender, EventArgs e)
{
    if (IsOrderIDValid())
    {
        // Create the connection.
        using (SqlConnection connection = new SqlConnection(Properties.Settings.Default.connString))
        {
            // Create the SqlCommand object and identify it as a stored procedure.
            using (SqlCommand sqlCommand = new SqlCommand("Sales.uspCancelOrder", connection))
            {
                sqlCommand.CommandType = CommandType.StoredProcedure;

                // Add the order ID input parameter for the stored procedure.
                sqlCommand.Parameters.Add(new SqlParameter("@orderID", SqlDbType.Int));
                sqlCommand.Parameters["@orderID"].Value = parsedOrderID;
            }
        }
    }
}

```



```

        try
        {
            // Open the connection.
            connection.Open();

            // Run the command to execute the stored procedure.
            sqlCommand.ExecuteNonQuery();
        }
        catch
        {
            MessageBox.Show("The cancel operation was not completed.");
        }
        finally
        {
            // Close connection.
            connection.Close();
        }
    }
}

}

/// <summary>
/// Fills an order by calling the Sales.uspFillOrder stored
/// procedure on the database.
/// </summary>
private void btnFillOrder_Click(object sender, EventArgs e)
{
    if (IsOrderIDValid())
    {
        // Create the connection.
        using (SqlConnection connection = new SqlConnection(Properties.Settings.Default.connString))
        {
            // Create command and identify it as a stored procedure.
            using (SqlCommand sqlCommand = new SqlCommand("Sales.uspFillOrder", connection))
            {
                sqlCommand.CommandType = CommandType.StoredProcedure;

                // Add the order ID input parameter for the stored procedure.
                sqlCommand.Parameters.Add(new SqlParameter("@orderID", SqlDbType.Int));
                sqlCommand.Parameters["@orderID"].Value = parsedOrderID;

                // Add the filled date input parameter for the stored procedure.
                sqlCommand.Parameters.Add(new SqlParameter("@FilledDate", SqlDbType.DateTime, 8));
                sqlCommand.Parameters["@FilledDate"].Value = dtpFillDate.Value;

                try
                {
                    connection.Open();

                    // Execute the stored procedure.
                    sqlCommand.ExecuteNonQuery();
                }
                catch
                {
                    MessageBox.Show("The fill operation was not completed.");
                }
                finally
                {
                    // Close the connection.
                    connection.Close();
                }
            }
        }
    }
}

/// <summary>
/// Closes the form.

```

```

''' </summary>
private void btnFinishUpdates_Click(object sender, EventArgs e)
{
    this.Close();
}

```

```

''' <summary>
''' Executes a t-SQL SELECT query on the database to
''' obtain order data for a specified order ID.
''' </summary>
Private Sub btnFindByOrderID_Click(sender As Object, e As EventArgs) Handles btnFindByOrderID.Click

    ' Prepare the connection and the command.
    If IsOrderIDValid() Then

        ' Create the connection.
        Using connection As New SqlConnection(My.Settings.connString)

            ' Define the query string that has a parameter for orderID.
            Const sql As String = "select * from Sales.Orders where orderID = @orderID"

            ' Create a SqlCommand object.
            Using sqlCommand As New SqlCommand(sql, connection)

                ' Define the @orderID parameter and its value.
                sqlCommand.Parameters.Add(New SqlParameter("@orderID", SqlDbType.Int))
                sqlCommand.Parameters("@orderID").Value = parsedOrderID

                Try
                    ' Open connection.
                    connection.Open()

                    ' Execute the query.
                    Dim dataReader As SqlDataReader = sqlCommand.ExecuteReader()

                    ' Create a data table to hold the retrieved data.
                    Dim dataTable As New DataTable()

                    ' Load the data from SqlDataReader into the data table.
                    dataTable.Load(dataReader)

                    ' Display the data from the data table in the data grid view.
                    Me.dgvCustomerOrders.DataSource = dataTable

                    ' Close the SqlDataReader.
                    dataReader.Close()
                Catch
                    MessageBox.Show("The requested order could not be loaded into the form.")
                Finally
                    ' Close the connection.
                    connection.Close()
                End Try
            End Using
        End Using
    End If
End Sub

''' <summary>
''' Fills an order by running the Sales.uspFillOrder stored procedure on the database.
''' </summary>
Private Sub btnFillOrder_Click(sender As Object, e As EventArgs) Handles btnFillOrder.Click

    ' Set up and run stored procedure only if OrderID is valid.
    If IsOrderIDValid() Then

        ' Create the connection.
        Using connection As New SqlConnection(My.Settings.connString)

```

```

' Create command and identify it as a stored procedure.
Using sqlCommand As New SqlCommand("Sales.uspFillOrder", connection)

    sqlCommand.CommandType = CommandType.StoredProcedure

    ' Add input parameter for the stored procedure.
    sqlCommand.Parameters.Add(New SqlParameter("@orderID", SqlDbType.Int))
    sqlCommand.Parameters("@orderID").Value = parsedOrderID

    ' Add second input parameter.
    sqlCommand.Parameters.Add(New SqlParameter("@FilledDate", SqlDbType.DateTime, 8))
    sqlCommand.Parameters("@FilledDate").Value = dtpFillDate.Value

Try
    ' Open the connection.
    connection.Open()

    ' Run the SqlCommand.
    sqlCommand.ExecuteNonQuery()
Catch
    ' A simple catch.
    MessageBox.Show("The fill operation was not completed.")
Finally
    ' Close the connection.
    connection.Close()
End Try
End Using
End Using
End If
End Sub

''' <summary>
''' Cancels an order by running the Sales.uspCancelOrder stored procedure on the database.
''' </summary>
Private Sub btnCancelOrder_Click(sender As Object, e As EventArgs) Handles btnCancelOrder.Click

    ' Set up and run the stored procedure only if OrderID is ready.
    If IsOrderIDValid() Then

        ' Create the connection.
        Using connection As New SqlConnection(My.Settings.connString)

            ' Create the command and identify it as a stored procedure.
            Using sqlCommand As New SqlCommand("Sales.uspCancelOrder", connection)
                sqlCommand.CommandType = CommandType.StoredProcedure

                ' Add input parameter for the stored procedure.
                sqlCommand.Parameters.Add(New SqlParameter("@orderID", SqlDbType.Int))
                sqlCommand.Parameters("@orderID").Value = parsedOrderID

            Try
                ' Open the connection.
                connection.Open()

                ' Run the SqlCommand.
                sqlCommand.ExecuteNonQuery()
            Catch
                ' A simple catch.
                MessageBox.Show("The cancel operation was not completed.")
            Finally
                ' Close connection.
                connection.Close()
            End Try
        End Using
    End Using
End Using
End If
End Sub

```

```
''' <summary>
''' Closes the form and returns focus to the Navigation form.
''' </summary>
Private Sub btnFinishUpdates_Click(sender As Object, e As EventArgs) Handles btnFinishUpdates.Click
    Me.Close()
End Sub
```

Test your application

Select the F5 key to build and test your application after you code each Click event handler, and then after you finish coding.

See also

- [Visual Studio data tools for .NET](#)

Create a simple data application with WPF and Entity Framework 6

8/5/2021 • 19 minutes to read • [Edit Online](#)

This walkthrough shows how to create a basic "forms over data" application in Visual Studio. The app uses SQL Server LocalDB, the Northwind database, Entity Framework 6 (not Entity Framework Core), and Windows Presentation Foundation for .NET Framework (not .NET Core). It shows how to do basic databinding with a master-detail view, and it also has a custom Binding Navigator with buttons for **Move Next**, **Move Previous**, **Move to beginning**, **Move to end**, **Update** and **Delete**.

This article focuses on using data tools in Visual Studio, and does not attempt to explain the underlying technologies in any depth. It assumes that you have a basic familiarity with XAML, Entity Framework, and SQL. This example also does not demonstrate Model-View-ViewModel (MVVM) architecture, which is standard for WPF applications. However, you can copy this code into your own MVVM application with few modifications.

Install and connect to Northwind

This example uses SQL Server Express LocalDB and the Northwind sample database. If the ADO.NET data provider for that product supports Entity Framework, it should work with other SQL database products just as well.

1. If you don't have SQL Server Express LocalDB, install it either from the [SQL Server Express download page](#), or through the **Visual Studio Installer**. In the **Visual Studio Installer**, you can install SQL Server Express LocalDB as part of the **.NET desktop development** workload or as an individual component.
2. Install the Northwind sample database by following these steps:
 - a. In Visual Studio, open the **SQL Server Object Explorer** window. (**SQL Server Object Explorer** is installed as part of the **Data storage and processing** workload in the **Visual Studio Installer**.) Expand the **SQL Server** node. Right-click on your LocalDB instance and select **New Query**.

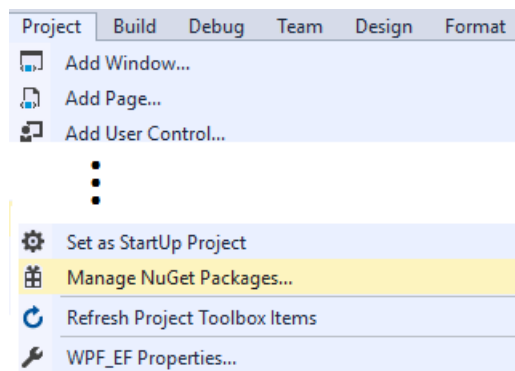
A query editor window opens.
 - b. Copy the [Northwind Transact-SQL script](#) to your clipboard. This T-SQL script creates the Northwind database from scratch and populates it with data.
 - c. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes running and the Northwind database is created.

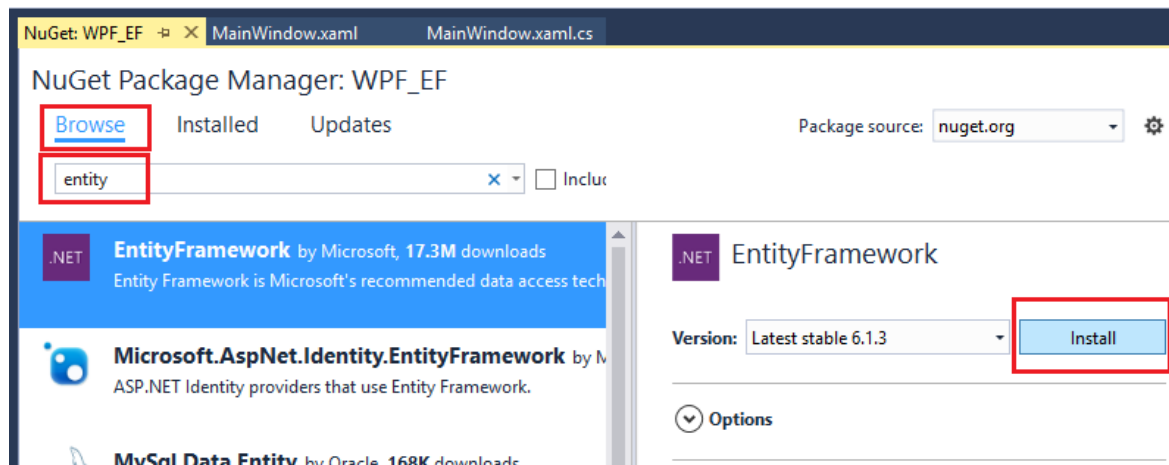
3. [Add new connections](#) for Northwind.

Configure the project

1. In Visual Studio, create a new C# **WPF App** project.
2. Add the NuGet package for Entity Framework 6. In **Solution Explorer**, select the project node. In the main menu, choose **Project > Manage NuGet Packages**.



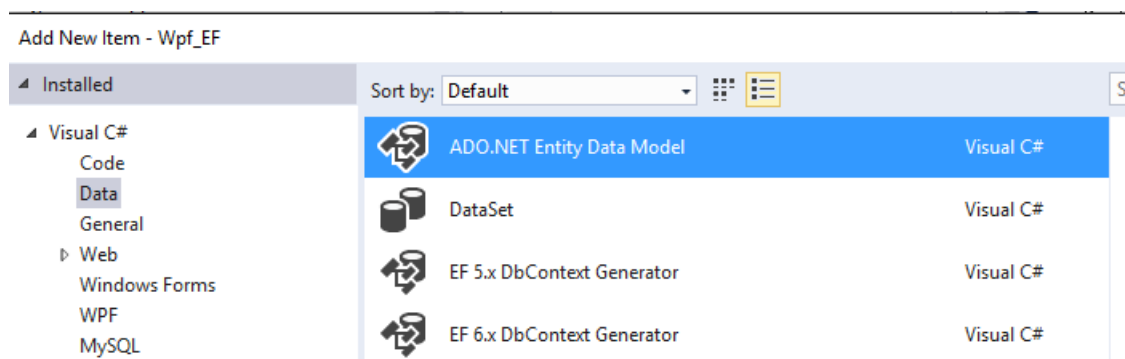
3. In the **NuGet Package Manager**, click on the **Browse** link. Entity Framework is probably the top package in the list. Click **Install** in the right pane and follow the prompts. The Output window tells you when the install is finished.



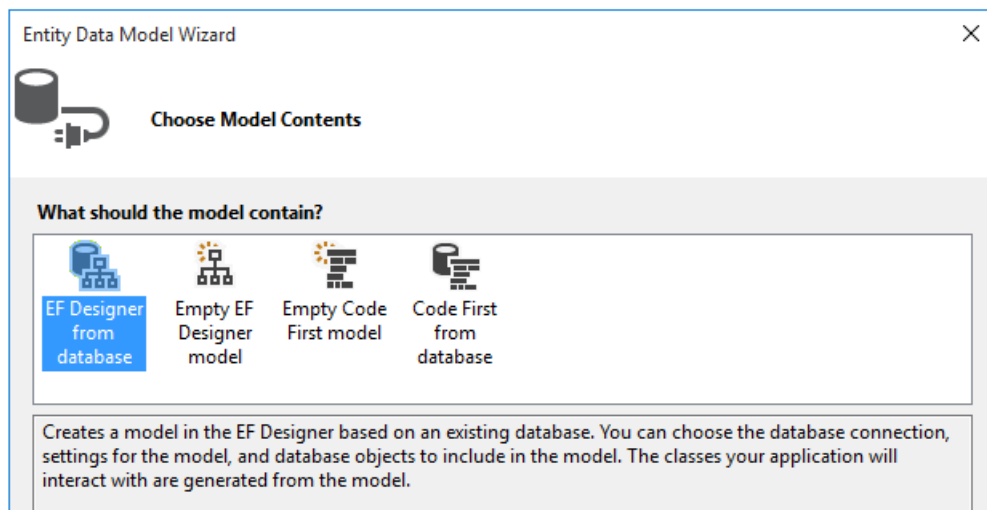
4. Now you can use Visual Studio to create a model based on the Northwind database.

Create the model

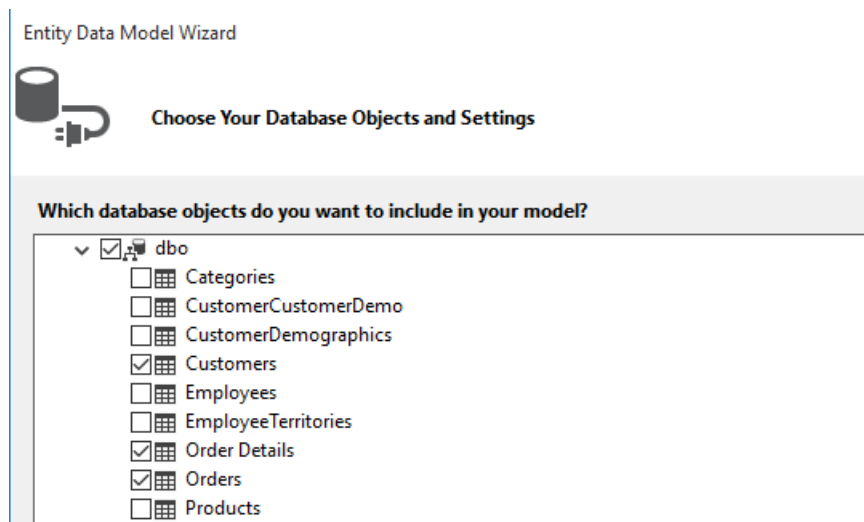
1. Right-click on the project node in **Solution Explorer** and choose **Add > New Item**. In the left pane, under the C# node, choose **Data** and in the middle pane, choose **ADO.NET Entity Data Model**.



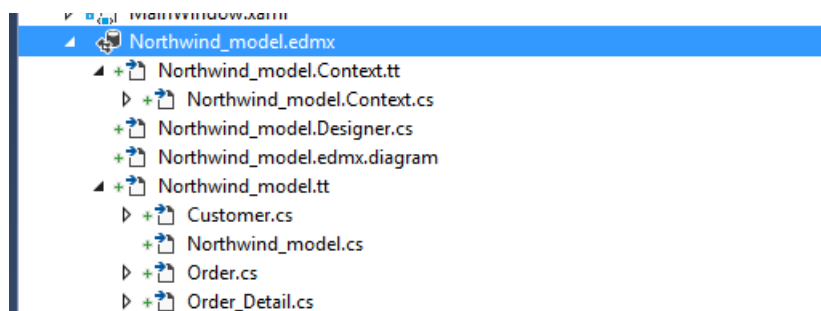
2. Call the model `Northwind_model1` and choose OK. The Entity Data Model Wizard opens. Choose EF Designer from database and then click Next.



3. In the next screen, enter or choose your LocalDB Northwind connection (for example, (localdb)\MSSQLLocalDB), specify the Northwind database, and click **Next**.
4. In the next page of the wizard, choose which tables, stored procedures, and other database objects to include in the Entity Framework model. Expand the dbo node in the tree view and choose **Customers**, **Orders**, and **Order Details**. Leave the defaults checked and click **Finish**.



5. The wizard generates the C# classes that represent the Entity Framework model. The classes are plain old C# classes and they are what we databind to the WPF user interface. The *.edmx* file describes the relationships and other metadata that associates the classes with objects in the database. The *.tt* files are T4 templates that generate the code that operates on the model and saves changes to the database. You can see all these files in **Solution Explorer** under the Northwind_model node:



The designer surface for the *.edmx* file enables you to modify some properties and relationships in the model. We are not going to use the designer in this walkthrough.

6. The *.tt* files are general purpose and you need to tweak one of them to work with WPF databinding, which requires *ObservableCollections*. In **Solution Explorer**, expand the Northwind_model node until you find

Northwind_model.tt. (Make sure you are not in the *.Context.tt* file, which is directly below the *.edmx* file.)

- Replace the two occurrences of `ICollection` with `ObservableCollection<T>`.
- Replace the first occurrence of `HashSet<T>` with `ObservableCollection<T>` around line 51. Do not replace the second occurrence of `HashSet`.
- Replace the only occurrence of `System.Collections.Generic` (around line 431) with `System.Collections.ObjectModel`.

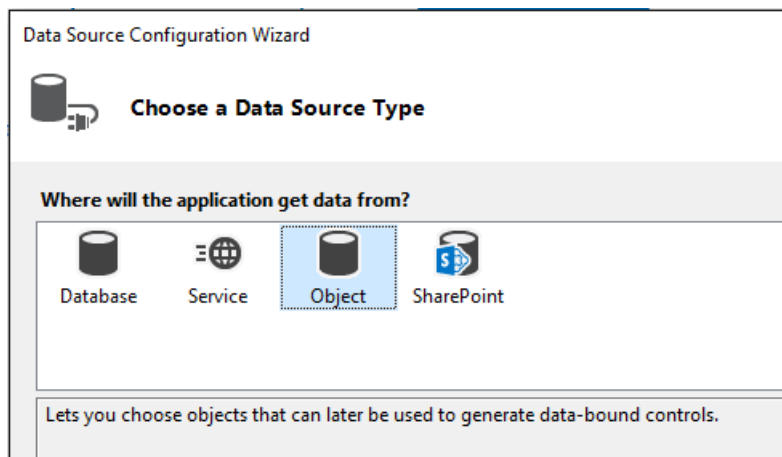
7. Press **Ctrl+Shift+B** to build the project. When the build finishes, the model classes are visible to the data sources wizard.

Now you are ready to hook up this model to the XAML page so that you can view, navigate, and modify the data.

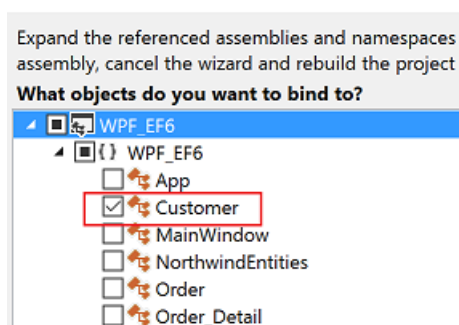
Databind the model to the XAML page

It is possible to write your own databinding code, but it is much easier to let Visual Studio do it for you.

1. From the main menu, choose **Project > Add new data source** to bring up the **Data Source Configuration Wizard**. Choose **Object** because you are binding to the model classes, not to the database:



2. Expand the node for your project, and select **Customer**. (Sources for Orders are automatically generated from the Orders navigation property in Customer.)



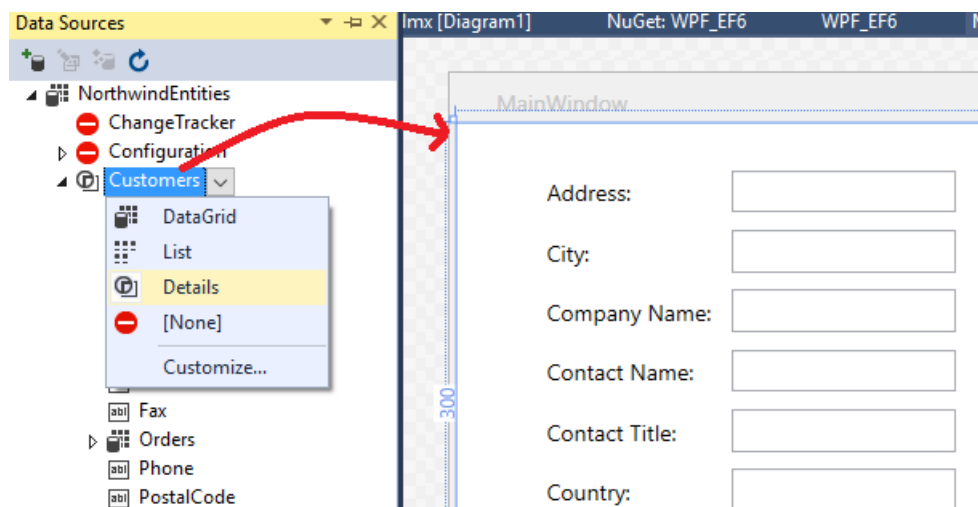
3. Click **Finish**.
4. Navigate to *MainWindow.xaml* in Code View. We're keeping the XAML simple for the purposes of this example. Change the title of *MainWindow* to something more descriptive, and increase its Height and Width to 600 x 800 for now. You can always change it later. Now add these three row definitions to the main grid, one row for the navigation buttons, one for the customer's details, and one for the grid that shows their orders:


```

<Grid.RowDefinitions>
    <RowDefinition Height="auto"/>
    <RowDefinition Height="auto"/>
    <RowDefinition Height="*/"/>
</Grid.RowDefinitions>

```

- Now open *MainWindow.xaml* so that you're viewing it in the designer. This causes the **Data Sources** window to appear as an option in the Visual Studio window margin next to the **Toolbox**. Click on the tab to open the window, or else press **Shift+Alt+D** or choose **View > Other Windows > Data Sources**. We are going to display each property in the Customers class in its own individual text box. First, click on the arrow in the **Customers** combo box and choose **Details**. Then, drag the node onto the middle part of the design surface so that the designer knows you want it to go in the middle row. If you misplace it, you can specify the row manually later in the XAML. By default, the controls are placed vertically in a grid element, but at this point, you can arrange them however you like on the form. For example, it might make sense to put the **Name** text box on top, above the address. The sample application for this article reorders the fields and rearranges them into two columns.



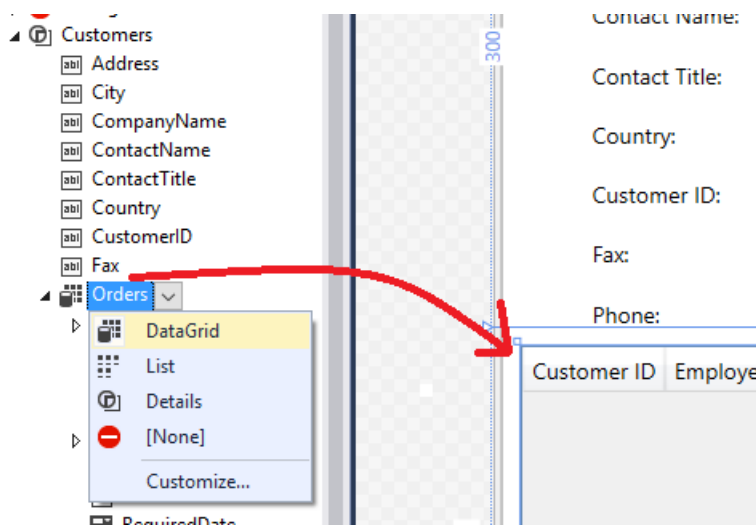
In the code view, you can now see a new `Grid` element in row 1 (the middle row) of the parent Grid. The parent Grid has a `DataContext` attribute that refers to a `CollectionViewSource` that's been added to the `Windows.Resources` element. Given that data context, when the first text box binds to **Address**, that name is mapped to the `Address` property in the current `Customer` object in the `CollectionViewSource`.

```

<Grid DataContext="{StaticResource customerViewSource}">

```

- When a customer is visible in the top half of the window, you want to see their orders in the bottom half. You show the orders in a single grid view control. For master-detail databinding to work as expected, it is important that you bind to the `Orders` property in the `Customers` class, not to the separate `Orders` node. Drag the `Orders` property of the `Customers` class to the lower half of the form, so that the designer puts it in row 2:



7. Visual Studio has generated all the binding code that connects the UI controls to events in the model. All you need to do, in order to see some data, is to write some code to populate the model. First, navigate to *MainWindow.xaml.cs* and add a data member to the *MainWindow* class for the data context. This object, which has been generated for you, acts something like a control that tracks changes and events in the model. You'll also add *CollectionViewSource* data members for customers and orders, and the associated constructor initialization logic. The top of the class should look like this:

```
public partial class MainWindow : Window
{
    NorthwindEntities context = new NorthwindEntities();
    CollectionViewSource custViewSource;
    CollectionViewSource ordViewSource;

    public MainWindow()
    {
        InitializeComponent();
        custViewSource = ((CollectionViewSource)FindResource("customerViewSource"));
        ordViewSource = ((CollectionViewSource)FindResource("customerOrdersViewSource"));
        DataContext = this;
    }
}
```

Add a `using` directive for `System.Data.Entity` to bring the `Load` extension method into scope:

```
using System.Data.Entity;
```

Now, scroll down and find the `Window_Loaded` event handler. Notice that Visual Studio has added a `CollectionViewSource` object. This represents the `NorthwindEntities` object that you selected when you created the model. You added that already, so you don't need it here. Let's replace the code in

`Window_Loaded` so that the method now looks like this:

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    // Load is an extension method on IQueryable,
    // defined in the System.Data.Entity namespace.
    // This method enumerates the results of the query,
    // similar to ToList but without creating a list.
    // When used with Linq to Entities, this method
    // creates entity objects and adds them to the context.
    context.Customers.Load();

    // After the data is loaded, call the DbSet<T>.Local property
    // to use the DbSet<T> as a binding source.
    custViewSource.Source = context.Customers.Local;
}
```

8. Press **F5**. You should see the details for the first customer that was retrieved into the `CollectionViewSource`. You should also see their orders in the data grid. The formatting isn't great, so let's fix that up. You can also create a way to view the other records and do basic CRUD operations.

Adjust the page design and add grids for new customers and orders

The default arrangement produced by Visual Studio is not ideal for your application, so we'll provide the final XAML here to copy into your code. You also need some "forms" (which are actually Grids) to enable the user to add a new customer or order. In order to be able to add a new customer and order, you need a separate set of text boxes that are not data-bound to the `CollectionViewSource`. You'll control which grid the user sees at any given time by setting the `Visible` property in the handler methods. Finally, you add a Delete button to each row in the Orders grid to enable the user to delete an individual order.

First, add these styles to the `Windows.Resources` element in *MainWindow.xaml*:

```
<Style x:Key="Label" TargetType="{x:Type Label}" BasedOn="{x:Null}">
    <Setter Property="HorizontalAlignment" Value="Left"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
    <Setter Property="Margin" Value="3"/>
    <Setter Property="Height" Value="23"/>
</Style>
<Style x:Key="CustTextBox" TargetType="{x:Type TextBox}" BasedOn="{x:Null}">
    <Setter Property="HorizontalAlignment" Value="Right"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
    <Setter Property="Margin" Value="3"/>
    <Setter Property="Height" Value="26"/>
    <Setter Property="Width" Value="120"/>
</Style>
```

Next, replace the entire outer Grid with this markup:

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="*/>
    </Grid.RowDefinitions>
    <Grid x:Name="existingCustomerGrid" Grid.Row="1" HorizontalAlignment="Left" Margin="5"
    Visibility="Visible" VerticalAlignment="Top" Background="AntiqueWhite" DataContext="{StaticResource
    customerViewSource}">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" MinWidth="233"/>
            <ColumnDefinition Width="Auto" MinWidth="397"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
```

```

        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Label Content="Customer ID:" Grid.Row="0" Style="{StaticResource Label}"/>
        <TextBox x:Name="customerIDTextBox" Grid.Row="0" Style="{StaticResource CustTextBox}"
            Text="{Binding CustomerID, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Company Name:" Grid.Row="1" Style="{StaticResource Label}"/>
        <TextBox x:Name="companyNameTextBox" Grid.Row="1" Style="{StaticResource CustTextBox}"
            Text="{Binding CompanyName, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Contact Name:" Grid.Row="2" Style="{StaticResource Label}"/>
        <TextBox x:Name="contactNameTextBox" Grid.Row="2" Style="{StaticResource CustTextBox}"
            Text="{Binding ContactName, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Contact title:" Grid.Row="3" Style="{StaticResource Label}"/>
        <TextBox x:Name="contactTitleTextBox" Grid.Row="3" Style="{StaticResource CustTextBox}"
            Text="{Binding ContactTitle, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Address:" Grid.Row="4" Style="{StaticResource Label}"/>
        <TextBox x:Name="addressTextBox" Grid.Row="4" Style="{StaticResource CustTextBox}"
            Text="{Binding Address, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="City:" Grid.Column="1" Grid.Row="0" Style="{StaticResource Label}"/>
        <TextBox x:Name="cityTextBox" Grid.Column="1" Grid.Row="0" Style="{StaticResource CustTextBox}"
            Text="{Binding City, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Country:" Grid.Column="1" Grid.Row="1" Style="{StaticResource Label}"/>
        <TextBox x:Name="countryTextBox" Grid.Column="1" Grid.Row="1" Style="{StaticResource CustTextBox}"
            Text="{Binding Country, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Fax:" Grid.Column="1" Grid.Row="2" Style="{StaticResource Label}"/>
        <TextBox x:Name="faxTextBox" Grid.Column="1" Grid.Row="2" Style="{StaticResource CustTextBox}"
            Text="{Binding Fax, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Phone:" Grid.Column="1" Grid.Row="3" Style="{StaticResource Label}"/>
        <TextBox x:Name="phoneTextBox" Grid.Column="1" Grid.Row="3" Style="{StaticResource CustTextBox}"
            Text="{Binding Phone, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Postal Code:" Grid.Column="1" Grid.Row="4" VerticalAlignment="Center" Style="{StaticResource Label}"/>
        <TextBox x:Name="postalCodeTextBox" Grid.Column="1" Grid.Row="4" Style="{StaticResource CustTextBox}"
            Text="{Binding PostalCode, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Region:" Grid.Column="1" Grid.Row="5" Style="{StaticResource Label}"/>
        <TextBox x:Name="regionTextBox" Grid.Column="1" Grid.Row="5" Style="{StaticResource CustTextBox}"
            Text="{Binding Region, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
    </Grid>
    <Grid x:Name="newCustomerGrid" Grid.Row="1" HorizontalAlignment="Left" VerticalAlignment="Top"
Margin="5" DataContext="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type Window}}},
Path=newCustomer, UpdateSourceTrigger=Explicit}" Visibility="Collapsed" Background="CornflowerBlue">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" MinWidth="233"/>
            <ColumnDefinition Width="Auto" MinWidth="397"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Label Content="Customer ID:" Grid.Row="0" Style="{StaticResource Label}"/>

```

```

        <TextBox x:Name="add_customerIDTextBox" Grid.Row="0" Style="{StaticResource CustTextBox}"
            Text="{Binding CustomerID, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Company Name:" Grid.Row="1" Style="{StaticResource Label}"/>
        <TextBox x:Name="add_companyNameTextBox" Grid.Row="1" Style="{StaticResource CustTextBox}"
            Text="{Binding CompanyName, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true }"/>
        <Label Content="Contact Name:" Grid.Row="2" Style="{StaticResource Label}"/>
        <TextBox x:Name="add_contactNameTextBox" Grid.Row="2" Style="{StaticResource CustTextBox}"
            Text="{Binding ContactName, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Contact title:" Grid.Row="3" Style="{StaticResource Label}"/>
        <TextBox x:Name="add_contactTitleTextBox" Grid.Row="3" Style="{StaticResource CustTextBox}"
            Text="{Binding ContactTitle, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Address:" Grid.Row="4" Style="{StaticResource Label}"/>
        <TextBox x:Name="add_addressTextBox" Grid.Row="4" Style="{StaticResource CustTextBox}"
            Text="{Binding Address, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="City:" Grid.Column="1" Grid.Row="0" Style="{StaticResource Label}"/>
        <TextBox x:Name="add_cityTextBox" Grid.Column="1" Grid.Row="0" Style="{StaticResource CustTextBox}"
            Text="{Binding City, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Country:" Grid.Column="1" Grid.Row="1" Style="{StaticResource Label}"/>
        <TextBox x:Name="add_countryTextBox" Grid.Column="1" Grid.Row="1" Style="{StaticResource
CustTextBox}"
            Text="{Binding Country, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Fax:" Grid.Column="1" Grid.Row="2" Style="{StaticResource Label}"/>
        <TextBox x:Name="add_faxTextBox" Grid.Column="1" Grid.Row="2" Style="{StaticResource CustTextBox}"
            Text="{Binding Fax, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Phone:" Grid.Column="1" Grid.Row="3" Style="{StaticResource Label}"/>
        <TextBox x:Name="add_phoneTextBox" Grid.Column="1" Grid.Row="3" Style="{StaticResource
CustTextBox}"
            Text="{Binding Phone, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Postal Code:" Grid.Column="1" Grid.Row="4" VerticalAlignment="Center" Style="{StaticResource Label}"/>
        <TextBox x:Name="add_postalCodeTextBox" Grid.Column="1" Grid.Row="4" Style="{StaticResource
CustTextBox}"
            Text="{Binding PostalCode, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Region:" Grid.Column="1" Grid.Row="5" Style="{StaticResource Label}"/>
        <TextBox x:Name="add_regionTextBox" Grid.Column="1" Grid.Row="5" Style="{StaticResource
CustTextBox}"
            Text="{Binding Region, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
    </Grid>
    <Grid x:Name="newOrderGrid" Grid.Row="1" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="5"
DataContext="{Binding Path=newOrder, Mode=TwoWay}" Visibility="Collapsed" Background="LightGreen">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" MinWidth="233"/>
            <ColumnDefinition Width="Auto" MinWidth="397"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Label Content="New Order Form" FontWeight="Bold"/>
        <Label Content="Employee ID:" Grid.Row="1" Style="{StaticResource Label}"/>
        <TextBox x:Name="add_employeeIDTextBox" Grid.Row="1" Style="{StaticResource CustTextBox}"
            Text="{Binding EmployeeID, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>

```

```

ValidatesOnExceptions=true"/>
        <Label Content="Order Date:" Grid.Row="2" Style="{StaticResource Label}"/>
        <DatePicker x:Name="add_orderDatePicker" Grid.Row="2" HorizontalAlignment="Right" Width="120"
            SelectedDate="{Binding OrderDate, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true, UpdateSourceTrigger=PropertyChanged}"/>
        <Label Content="Required Date:" Grid.Row="3" Style="{StaticResource Label}"/>
        <DatePicker x:Name="add_requiredDatePicker" Grid.Row="3" HorizontalAlignment="Right" Width="120"
            SelectedDate="{Binding RequiredDate, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true, UpdateSourceTrigger=PropertyChanged}"/>
        <Label Content="Shipped Date:" Grid.Row="4" Style="{StaticResource Label}"/>
        <DatePicker x:Name="add_shippedDatePicker" Grid.Row="4" HorizontalAlignment="Right" Width="120"
            SelectedDate="{Binding ShippedDate, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true, UpdateSourceTrigger=PropertyChanged}"/>
        <Label Content="Ship Via:" Grid.Row="5" Style="{StaticResource Label}"/>
        <TextBox x:Name="add_ShipViaTextBox" Grid.Row="5" Style="{StaticResource CustTextBox}"
            Text="{Binding ShipVia, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
        <Label Content="Freight" Grid.Row="6" Style="{StaticResource Label}"/>
        <TextBox x:Name="add_freightTextBox" Grid.Row="6" Style="{StaticResource CustTextBox}"
            Text="{Binding Freight, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"/>
    </Grid>
    <DataGrid x:Name="ordersDataGrid" SelectionUnit="Cell" SelectionMode="Single"
AutoGenerateColumns="False" CanUserAddRows="false" IsEnabled="True" EnableRowVirtualization="True"
Width="auto" ItemsSource="{Binding Source={StaticResource customerOrdersViewSource}}" Margin="10,10,10,10"
Grid.Row="2" RowDetailsVisibilityMode="VisibleWhenSelected">
        <DataGrid.Columns>
            <DataGridTemplateColumn>
                <DataGridTemplateColumn.CellTemplate>
                    <DataTemplate>
                        <Button Content="Delete" Command="{StaticResource DeleteOrderCommand}"
CommandParameter="{Binding}"/>
                    </DataTemplate>
                </DataGridTemplateColumn.CellTemplate>
            </DataGridTemplateColumn>
            <DataGridTextColumn x:Name="customerIDColumn" Binding="{Binding CustomerID}" Header="Customer
ID" Width="SizeToHeader"/>
            <DataGridTextColumn x:Name="employeeIDColumn" Binding="{Binding EmployeeID}" Header="Employee
ID" Width="SizeToHeader"/>
            <DataGridTextColumn x:Name="freightColumn" Binding="{Binding Freight}" Header="Freight"
Width="SizeToHeader"/>
            <DataGridTemplateColumn x:Name="orderDateColumn" Header="Order Date" Width="SizeToHeader">
                <DataGridTemplateColumn.CellTemplate>
                    <DataTemplate>
                        <DatePicker SelectedDate="{Binding OrderDate, Mode=TwoWay,
NotifyOnValidationError=true, ValidatesOnExceptions=true, UpdateSourceTrigger=PropertyChanged}"/>
                    </DataTemplate>
                </DataGridTemplateColumn.CellTemplate>
            </DataGridTemplateColumn>
            <DataGridTextColumn x:Name="orderIDColumn" Binding="{Binding OrderID}" Header="Order ID"
Width="SizeToHeader"/>
            <DataGridTemplateColumn x:Name="requiredDateColumn" Header="Required Date"
Width="SizeToHeader">
                <DataGridTemplateColumn.CellTemplate>
                    <DataTemplate>
                        <DatePicker SelectedDate="{Binding RequiredDate, Mode=TwoWay,
NotifyOnValidationError=true, ValidatesOnExceptions=true, UpdateSourceTrigger=PropertyChanged}"/>
                    </DataTemplate>
                </DataGridTemplateColumn.CellTemplate>
            </DataGridTemplateColumn>
            <DataGridTextColumn x:Name="shipAddressColumn" Binding="{Binding ShipAddress}" Header="Ship
Address" Width="SizeToHeader"/>
            <DataGridTextColumn x:Name="shipCityColumn" Binding="{Binding ShipCity}" Header="Ship City"
Width="SizeToHeader"/>
            <DataGridTextColumn x:Name="shipCountryColumn" Binding="{Binding ShipCountry}" Header="Ship
Country" Width="SizeToHeader"/>
            <DataGridTextColumn x:Name="shipNameColumn" Binding="{Binding ShipName}" Header="Ship Name"
Width="SizeToHeader"/>
            <DataGridTemplateColumn x:Name="shippedDateColumn" Header="Shipped Date" Width="SizeToHeader">

```

```

        <DataGridTemplateColumn.CellTemplate>
            <DataTemplate>
                <DatePicker SelectedDate="{Binding ShippedDate, Mode=TwoWay,
NotifyOnValidationError=true, ValidatesOnExceptions=true, UpdateSourceTrigger=PropertyChanged}"/>
            </DataTemplate>
        </DataGridTemplateColumn.CellTemplate>
    </DataGridTemplateColumn>
    <DataGridTextColumn x:Name="shipPostalCodeColumn" Binding="{Binding ShipPostalCode}"
Header="Ship Postal Code" Width="SizeToHeader"/>
    <DataGridTextColumn x:Name="shipRegionColumn" Binding="{Binding ShipRegion}" Header="Ship
Region" Width="SizeToHeader"/>
    <DataGridTextColumn x:Name="shipViaColumn" Binding="{Binding ShipVia}" Header="Ship Via"
Width="SizeToHeader"/>
    </DataGrid.Columns>
</DataGrid>
</Grid>

```

Add buttons to navigate, add, update, and delete

In Windows Forms applications, you get a `BindingNavigator` object with buttons for navigating through rows in a database and doing basic CRUD operations. WPF does not provide a `BindingNavigator`, but it is easy enough to create one. You do that with buttons inside a horizontal `StackPanel`, and associate the buttons with `Commands` that are bound to methods in the code behind.

There are four parts to the command logic: (1) the commands, (2) the bindings, (3) the buttons, and (4) the command handlers in the code-behind.

Add commands, bindings, and buttons in XAML

- First, add the commands in the *MainWindow.xaml* file inside the `Windows.Resources` element:

```

<RoutedUICommand x:Key="FirstCommand" Text="First"/>
<RoutedUICommand x:Key="LastCommand" Text="Last"/>
<RoutedUICommand x:Key="NextCommand" Text="Next"/>
<RoutedUICommand x:Key="PreviousCommand" Text="Previous"/>
<RoutedUICommand x:Key="DeleteCustomerCommand" Text="Delete Customer"/>
<RoutedUICommand x:Key="DeleteOrderCommand" Text="Delete Order"/>
<RoutedUICommand x:Key="UpdateCommand" Text="Update"/>
<RoutedUICommand x:Key="AddCommand" Text="Add"/>
<RoutedUICommand x:Key="CancelCommand" Text="Cancel"/>

```

- A `CommandBinding` maps a `RoutedUICommand` event to a method in the code behind. Add this `CommandBindings` element after the `Windows.Resources` closing tag:

```

<Window.CommandBindings>
    <CommandBinding Command="{StaticResource FirstCommand}" Executed="FirstCommandHandler"/>
    <CommandBinding Command="{StaticResource LastCommand}" Executed="LastCommandHandler"/>
    <CommandBinding Command="{StaticResource NextCommand}" Executed="NextCommandHandler"/>
    <CommandBinding Command="{StaticResource PreviousCommand}" Executed="PreviousCommandHandler"/>
    <CommandBinding Command="{StaticResource DeleteCustomerCommand}"
Executed="DeleteCustomerCommandHandler"/>
    <CommandBinding Command="{StaticResource DeleteOrderCommand}"
Executed="DeleteOrderCommandHandler"/>
    <CommandBinding Command="{StaticResource UpdateCommand}" Executed="UpdateCommandHandler"/>
    <CommandBinding Command="{StaticResource AddCommand}" Executed="AddCommandHandler"/>
    <CommandBinding Command="{StaticResource CancelCommand}" Executed="CancelCommandHandler"/>
</Window.CommandBindings>

```

- Now, add the `StackPanel` with the navigation, add, delete, and update buttons. First, add this style to `Windows.Resources`:


```
<Style x:Key="NavButton" TargetType="{x:Type Button}" BasedOn="{x:Null}">
    <Setter Property="FontSize" Value="24"/>
    <Setter Property="FontFamily" Value="Segoe UI Symbol"/>
    <Setter Property="Margin" Value="2,2,2,0"/>
    <Setter Property="Width" Value="40"/>
    <Setter Property="Height" Value="auto"/>
</Style>
```

Second, paste this code just after the `RowDefinitions` for the outer `Grid` element, toward the top of the XAML page:

```
<StackPanel Orientation="Horizontal" Margin="2,2,2,0" Height="36" VerticalAlignment="Top"
Background="Gainsboro" DataContext="{StaticResource customerViewSource}"
d:LayoutOverrides="LeftMargin, RightMargin, TopMargin, BottomMargin">
    <Button Name="btnFirst" Content="|<" Command="{StaticResource FirstCommand}" Style="
{StaticResource NavButton}"/>
    <Button Name="btnPrev" Content="◀" Command="{StaticResource PreviousCommand}" Style="
{StaticResource NavButton}"/>
    <Button Name="btnNext" Content="▶" Command="{StaticResource NextCommand}" Style="{StaticResource
NavButton}"/>
    <Button Name="btnLast" Content=">|" Command="{StaticResource LastCommand}" Style="{StaticResource
NavButton}"/>
    <Button Name="btnDelete" Content="Delete Customer" Command="{StaticResource
DeleteCustomerCommand}" FontSize="11" Width="120" Style="{StaticResource NavButton}"/>
    <Button Name="btnAdd" Content="New Customer" Command="{StaticResource AddCommand}" FontSize="11"
Width="80" Style="{StaticResource NavButton}"/>
    <Button Content="New Order" Name="btnNewOrder" FontSize="11" Width="80" Style="{StaticResource
NavButton}" Click="NewOrder_click"/>
    <Button Name="btnUpdate" Content="Commit" Command="{StaticResource UpdateCommand}" FontSize="11"
Width="80" Style="{StaticResource NavButton}"/>
    <Button Content="Cancel" Name="btnCancel" Command="{StaticResource CancelCommand}" FontSize="11"
Width="80" Style="{StaticResource NavButton}"/>
</StackPanel>
```

Add command handlers to the MainWindow class

The code-behind is minimal except for the add and delete methods. Navigation is performed by calling methods on the `View` property of the `CollectionViewSource`. The `DeleteOrderCommandHandler` shows how to perform a cascade delete on an order. We have to first delete the `Order_Details` that are associated with it. The `UpdateCommandHandler` adds a new customer or order to the collection, or else just updates an existing customer or order with the changes that the user made in the text boxes.

Add these handler methods to the `MainWindow` class in *MainWindow.xaml.cs*. If your `CollectionViewSource` for the Customers table has a different name, then you need to adjust the name in each of these methods:

```
private void LastCommandHandler(object sender, ExecutedRoutedEventArgs e)
{
    custViewSource.View.MoveCurrentToLast();
}

private void PreviousCommandHandler(object sender, ExecutedRoutedEventArgs e)
{
    custViewSource.View.MoveCurrentToPrevious();
}

private void NextCommandHandler(object sender, ExecutedRoutedEventArgs e)
{
    custViewSource.View.MoveCurrentToNext();
}

private void FirstCommandHandler(object sender, ExecutedRoutedEventArgs e)
{
    custViewSource.View.MoveCurrentToFirst();
}
```



```

        custViewSource.View.MoveCurrentToFirst();
    }

    private void DeleteCustomerCommandHandler(object sender, ExecutedRoutedEventArgs e)
    {
        // If existing window is visible, delete the customer and all their orders.
        // In a real application, you should add warnings and allow the user to cancel the operation.
        var cur = custViewSource.View.CurrentItem as Customer;

        var cust = (from c in context.Customers
                    where c.CustomerID == cur.CustomerID
                    select c).FirstOrDefault();

        if (cust != null)
        {
            foreach (var ord in cust.Orders.ToList())
            {
                Delete_Order(ord);
            }
            context.Customers.Remove(cust);
        }
        context.SaveChanges();
        custViewSource.View.Refresh();
    }

    // Commit changes from the new customer form, the new order form,
    // or edits made to the existing customer form.
    private void UpdateCommandHandler(object sender, ExecutedRoutedEventArgs e)
    {
        if (newCustomerGrid.IsVisible)
        {
            // Create a new object because the old one
            // is being tracked by EF now.
            Customer newCustomer = new Customer
            {
                Address = add_addressTextBox.Text,
                City = add_cityTextBox.Text,
                CompanyName = add_companyNameTextBox.Text,
                ContactName = add_contactNameTextBox.Text,
                ContactTitle = add_contactTitleTextBox.Text,
                Country = add_countryTextBox.Text,
                CustomerID = add_customerIDTextBox.Text,
                Fax = add_faxTextBox.Text,
                Phone = add_phoneTextBox.Text,
                PostalCode = add_postalCodeTextBox.Text,
                Region = add_regionTextBox.Text
            };

            // Perform very basic validation
            if (newCustomer.CustomerID.Length == 5)
            {
                // Insert the new customer at correct position:
                int len = context.Customers.Local.Count();
                int pos = len;
                for (int i = 0; i < len; ++i)
                {
                    if (String.CompareOrdinal(newCustomer.CustomerID, context.Customers.Local[i].CustomerID) <
0)
                    {
                        pos = i;
                        break;
                    }
                }
                context.Customers.Local.Insert(pos, newCustomer);
                custViewSource.View.Refresh();
                custViewSource.View.MoveCurrentTo(newCustomer);
            }
            else
            {

```

```

        MessageBox.Show("CustomerID must have 5 characters.");
    }

    newCustomerGrid.Visibility = Visibility.Collapsed;
    existingCustomerGrid.Visibility = Visibility.Visible;
}
else if (newOrderGrid.IsVisible)
{
    // Order ID is auto-generated so we don't set it here.
    // For CustomerID, address, etc we use the values from current customer.
    // User can modify these in the datagrid after the order is entered.

    Customer currentCustomer = (Customer)custViewSource.View.CurrentItem;

    Order newOrder = new Order()
    {
        OrderDate = add_orderDatePicker.SelectedDate,
        RequiredDate = add_requiredDatePicker.SelectedDate,
        ShippedDate = add_shippedDatePicker.SelectedDate,
        CustomerID = currentCustomer.CustomerID,
        ShipAddress = currentCustomer.Address,
        ShipCity = currentCustomer.City,
        ShipCountry = currentCustomer.Country,
        ShipName = currentCustomer.CompanyName,
        ShipPostalCode = currentCustomer.PostalCode,
        ShipRegion = currentCustomer.Region
    };

    try
    {
        newOrder.EmployeeID = Int32.Parse(add_employeeIDTextBox.Text);
    }
    catch
    {
        MessageBox.Show("EmployeeID must be a valid integer value.");
        return;
    }

    try
    {
        // Exercise for the reader if you are using Northwind:
        // Add the Northwind Shippers table to the model.

        // Acceptable ShipperID values are 1, 2, or 3.
        if (add_ShipViaTextBox.Text == "1" || add_ShipViaTextBox.Text == "2"
            || add_ShipViaTextBox.Text == "3")
        {
            newOrder.ShipVia = Convert.ToInt32(add_ShipViaTextBox.Text);
        }
        else
        {
            MessageBox.Show("Shipper ID must be 1, 2, or 3 in Northwind.");
            return;
        }
    }
    catch
    {
        MessageBox.Show("Ship Via must be convertible to int");
        return;
    }

    try
    {
        newOrder.Freight = Convert.ToDecimal(add_freightTextBox.Text);
    }
    catch
    {
        MessageBox.Show("Freight must be convertible to decimal.");
        return;
    }
}

```

```

    }

    // Add the order into the EF model
    context.Orders.Add(newOrder);
    ordViewSource.View.Refresh();
}

// Save the changes, either for a new customer, a new order
// or an edit to an existing customer or order.
context.SaveChanges();
}

// Sets up the form so that user can enter data. Data is later
// saved when user clicks Commit.
private void AddCommandHandler(object sender, ExecutedRoutedEventArgs e)
{
    existingCustomerGrid.Visibility = Visibility.Collapsed;
    newOrderGrid.Visibility = Visibility.Collapsed;
    newCustomerGrid.Visibility = Visibility.Visible;

    // Clear all the text boxes before adding a new customer.
    foreach (var child in newCustomerGrid.Children)
    {
        var tb = child as TextBox;
        if (tb != null)
        {
            tb.Text = "";
        }
    }
}

private void NewOrder_click(object sender, RoutedEventArgs e)
{
    var cust = custViewSource.View.CurrentItem as Customer;
    if (cust == null)
    {
        MessageBox.Show("No customer selected.");
        return;
    }

    existingCustomerGrid.Visibility = Visibility.Collapsed;
    newCustomerGrid.Visibility = Visibility.Collapsed;
    newOrderGrid.UpdateLayout();
    newOrderGrid.Visibility = Visibility.Visible;
}

// Cancels any input into the new customer form
private void CancelCommandHandler(object sender, ExecutedRoutedEventArgs e)
{
    add_addressTextBox.Text = "";
    add_cityTextBox.Text = "";
    add_companyNameTextBox.Text = "";
    add_contactNameTextBox.Text = "";
    add_contactTitleTextBox.Text = "";
    add_countryTextBox.Text = "";
    add_customerIDTextBox.Text = "";
    add_faxTextBox.Text = "";
    add_phoneTextBox.Text = "";
    add_postalCodeTextBox.Text = "";
    add_regionTextBox.Text = "";

    existingCustomerGrid.Visibility = Visibility.Visible;
    newCustomerGrid.Visibility = Visibility.Collapsed;
    newOrderGrid.Visibility = Visibility.Collapsed;
}

private void Delete_Order(Order order)
{
    // Find the order in the EF model.

```

```

var ord = (from o in context.Orders.Local
           where o.OrderID == order.OrderID
           select o).FirstOrDefault();

// Delete all the order_details that have
// this Order as a foreign key
foreach (var detail in ord.Order_Details.ToList())
{
    context.Order_Details.Remove(detail);
}

// Now it's safe to delete the order.
context.Orders.Remove(ord);
context.SaveChanges();

// Update the data grid.
ordViewSource.View.Refresh();
}

private void DeleteOrderCommandHandler(object sender, ExecutedRoutedEventArgs e)
{
    // Get the Order in the row in which the Delete button was clicked.
    Order obj = e.Parameter as Order;
    Delete_Order(obj);
}

```

Run the application

To start debugging, press **F5**. You should see customer and order data populated in the grid, and the navigation buttons should work as expected. Click on **Commit** to add a new customer or order to the model after you have entered the data. Click on **Cancel** to back out of a new customer or new order form without saving the data. You can make edits to existing customers and orders directly in the text boxes, and those changes are written to the model automatically.

See also

- [Visual Studio data tools for .NET](#)
- [Entity Framework documentation](#)

Visual Studio data tools for C++

8/5/2021 • 3 minutes to read • [Edit Online](#)

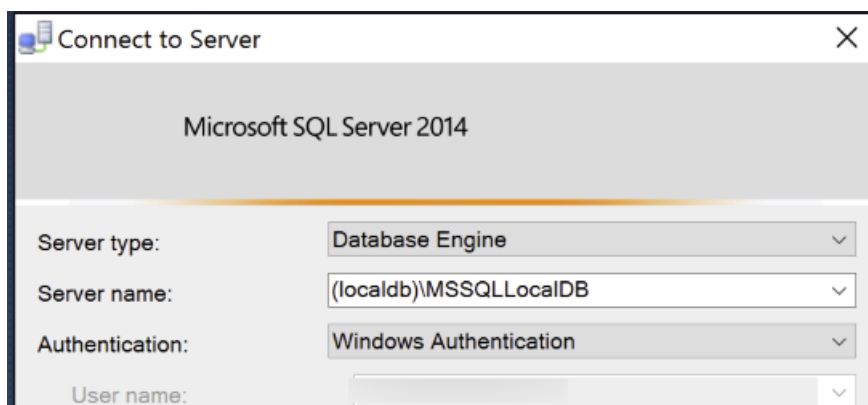
Native C++ can often provide the fastest performance when you are accessing data sources. However, data tooling for C++ applications in Visual Studio is not as rich as it is for .NET applications. For example, the **Data Sources** window cannot be used to drag and drop data sources onto a C++ design surface. If you need an object-relational layer, you will have to write your own, or use a third-party product. The same is true for data-binding functionality, although applications that use the Microsoft Foundation Class library can use some database classes, together with documents and views, to store data in memory and display it to the user. For more information, see [Data Access in Visual C++](#).

To connect to SQL databases, native C++ applications can use the ODBC and OLE DB drivers and the ADO provider that are included with Windows. These can connect to any database that supports those interfaces. The ODBC driver is the standard. OLE DB is provided for backward compatibility. For more information on those data technologies, see [Windows Data Access Components](#).

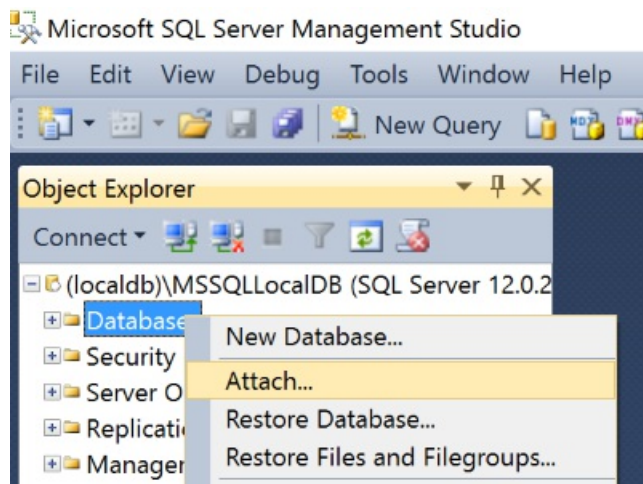
To take advantage of custom functionality in SQL Server 2005 and later, use the [SQL Server native client](#). The native client also contains the SQL Server ODBC driver and the SQL Server OLE DB provider in one native dynamic link library (DLL). These support applications using native-code APIs (ODBC, OLE DB and ADO) to Microsoft SQL Server. SQL Server Native Client installs with SQL Server Data Tools. The programming guide is here: [SQL Server native client programming](#).

To connect to localDB through ODBC and SQL Native Client from a C++ application

1. Install SQL Server Data Tools.
2. If you need a sample SQL database to connect to, download the Northwind database and unzip it to a new location.
3. Use SQL Server Management Studio to attach the unzipped *Northwind.mdf* file to localDB. When SQL Server Management Studio starts, connect to (localdb)\MSSQLLocalDB.



Then right-click on the localdb node in the left pane, and choose **Attach**.



4. Download the ODBC Windows SDK Sample, and unzip it to a new location. This sample shows the basic ODBC commands that are used to connect to a database and issue queries and commands. You can learn more about those functions in the [Microsoft Open Database Connectivity \(ODBC\)](#). When you first load the solution (it's in the C++ folder), Visual Studio will offer to upgrade the solution to the current version of Visual Studio. Click **Yes**.
5. To use the native client, you need its *header* file and *lib* file. These files contain functions and definitions specific to SQL Server, beyond the ODBC functions defined in `sql.h`. In **Project > Properties > VC++ Directories**, add the following include directory:

`%ProgramFiles%\Microsoft SQL Server\110\SDK\Include`

And this library directory:

`%ProgramFiles%\Microsoft SQL Server\110\SDK\Lib`

6. Add these lines in `odbcsql.cpp`. The `#define` prevents irrelevant OLE DB definitions from being compiled.

```
#define _SQLNCLI_ODBC_
#include <sqlncli.h>
```

Note that the sample does not actually use any of the native client functionality, so the preceding steps are not necessary for it to compile and run. But the project is now configured for you to use this functionality. For more information, see [SQL Server Native Client programming](#).

7. Specify which driver to use in the ODBC subsystem. The sample passes the DRIVER connection string attribute in as a command line argument. In **Project > Properties > Debugging**, add this command argument:

```
DRIVER="SQL Server Native Client 11.0"
```

8. Press **F5** to build and run the application. You should see a dialog box from the driver that prompts you to enter a database. Enter `(localdb)\MSSQLLocalDB`, and check **Use Trusted Connection**. Press **OK**. You should see a console with messages that indicate a successful connection. You should also see a command prompt where you can type in a SQL statement. The following screen shows an example query and the results:

```
C:\odbc\C++\Debug\odbcsql.exe
[01000] [Microsoft][SQL Server Native Client 11.0][SQL Server]Changed database context to 'master'. (5701)
[01000] [Microsoft][SQL Server Native Client 11.0][SQL Server]Changed language setting to us_english. (5703)
Connected!
Enter SQL commands, type (control)Z to exit
SQL COMMAND>select * from Northwind.dbo.Customers where CustomerId LIKE 'A%'
| CustomerID | CompanyName | ContactName | ContactTitle |
|-----|-----|-----|-----|
| ALFKI | Alfreds Futterkiste | Maria Anders | Sales Rep |
| ANATR | Ana Trujillo Emparedados y helados | Ana Trujillo | Sales Rep |
| ANTON | Antonio Moreno Taquería | Antonio Moreno | Sales Rep |
| AROUT | Around the Horn | Thomas Hardy | Sales Rep |
SQL COMMAND>
```

See also

- [Accessing data in Visual Studio](#)

Walkthrough: Create LINQ to SQL classes by using single-table inheritance (O/R Designer)

8/5/2021 • 4 minutes to read • [Edit Online](#)

The [LINQ to SQL tools in Visual Studio](#) supports single-table inheritance as it is typically implemented in relational systems. This walkthrough expands upon the generic steps provided in the [How to: Configure inheritance by using the O/R Designer](#) topic and provides some real data to demonstrate the use of inheritance in the O/R Designer.

During this walkthrough, you perform the following tasks:

- Create a database table and add data to it.
- Create a Windows Forms application.
- Add a LINQ to SQL file to a project.
- Create new entity classes.
- Configure the entity classes to use inheritance.
- Query the inherited class.
- Display the data on a Windows Form.

Create a table to inherit from

To see how inheritance works, you create a small `Person` table, use it as a base class, and then create an `Employee` object that inherits from it.

To create a base table to demonstrate inheritance

1. In **Server Explorer** or **Database Explorer**, right-click the **Tables** node and click **Add New Table**.

NOTE

You can use the Northwind database or any other database that you can add a table to.

2. In the **Table Designer**, add the following columns to the table:

| COLUMN NAME | DATA TYPE | ALLOW NULLS |
|-------------|---------------|-------------|
| ID | int | False |
| Type | int | True |
| FirstName | nvarchar(200) | False |
| LastName | nvarchar(200) | False |
| Manager | int | True |

3. Set the ID column as the primary key.

4. Save the table and name it **Person**.

Add data to the table

So that you can verify that inheritance is configured correctly, the table needs some data for each class in the single-table inheritance.

To add data to the table

1. Open the table in data view. (Right-click the **Person** table in **Server Explorer** or **Database Explorer** and click **Show Table Data**.)
2. Copy the following data into the table. (You can copy it and then paste it into the table by selecting the whole row in the **Results** Pane.)

| ID | TYPE | FIRSTNAME | LASTNAME | MANAGER |
|----|------|-----------|---------------|---------|
| 1 | 1 | Anne | Wallace | NULL |
| 2 | 1 | Carlos | Grilo | NULL |
| 3 | 1 | Yael | Peled | NULL |
| 4 | 2 | Gatis | Ozolins | 1 |
| 5 | 2 | Andreas | Hauser | 1 |
| 6 | 2 | Tiffany | Phuvasate | 1 |
| 7 | 2 | Alexey | Orekhov | 2 |
| 8 | 2 | Michał | Poliszkiewicz | 2 |
| 9 | 2 | Tai | Yee | 2 |
| 10 | 2 | Fabricio | Noriega | 3 |
| 11 | 2 | Mindy | Martin | 3 |
| 12 | 2 | Ken | Kwok | 3 |

Create a new project

Now that you have created the table, create a new project to demonstrate configuring inheritance.

To create the new Windows Forms application

1. In Visual Studio, on the **File** menu, select **New** > **Project**.
2. Expand either **Visual C#** or **Visual Basic** in the left-hand pane, then select **Windows Desktop**.
3. In the middle pane, select the **Windows Forms App** project type.
4. Name the project **InheritanceWalkthrough**, and then choose **OK**.

The **InheritanceWalkthrough** project is created, and added to **Solution Explorer**.

Add a LINQ to SQL classes file to the project

To add a LINQ to SQL file to the project

1. On the **Project** menu, click **Add New Item**.
2. Click the **LINQ to SQL Classes** template and then click **Add**.

The *.dbml* file is added to the project and the **O/R Designer** opens.

Create the inheritance by using the O/R Designer

Configure the inheritance by dragging an **Inheritance** object from the **Toolbox** onto the design surface.

To create the inheritance

1. In **Server Explorer** or **Database Explorer**, navigate to the **Person** table that you created earlier.
2. Drag the **Person** table onto the **O/R Designer** design surface.
3. Drag a second **Person** table onto the **O/R Designer** and change its name to **Employee**.
4. Delete the **Manager** property from the **Person** object.
5. Delete the **Type**, **ID**, **FirstName**, and **LastName** properties from the **Employee** object. (In other words, delete all properties except for **Manager**.)
6. From the **Object Relational Designer** tab of the **Toolbox**, create an **Inheritance** between the **Person** and **Employee** objects. To do this, click the **Inheritance** item in the **Toolbox** and release the mouse button. Next, click the **Employee** object and then the **Person** object in the **O/R Designer**. The arrow on the inheritance line then points to the **Person** object.
7. Click the **Inheritance** line on the design surface.
8. Set the **Discriminator Property** property to **Type**.
9. Set the **Derived Class Discriminator Value** property to 2.
10. Set the **Base Class Discriminator Value** property to 1.
11. Set the **Inheritance Default** property to **Person**.
12. Build the project.

Query the inherited class and display the data on the form

You now add some code to the form that queries for a specific class in the object model.

To create a LINQ query and display the results on the form

1. Drag a **ListBox** onto **Form1**.
2. Double-click the form to create a `Form1_Load` event handler.
3. Add the following code to the `Form1_Load` event handler:

```

Dim dc As New DataClasses1DataContext
Dim results = From emp In dc.Persons _
    Where TypeOf emp Is Employee _
    Select emp

For Each Emp As Employee In results
    ListBox1.Items.Add(Emp.LastName)
Next

```

```

NorthwindDataContext dc = new DataClasses1DataContext();
var results = from emp in dc.Persons
    where emp is Employee
    select emp;

foreach(Employee Emp in results)
{
    listBox1.Items.Add(Emp.LastName)
}

```

Test the application

Run the application and verify that the records displayed in the list box are all employees (records that have a value of 2 in their **Type** column).

To test the application

1. Press **F5**.
2. Verify that only records that have a value of 2 in their **Type** column are displayed.
3. Close the form. (On the **Debug** menu, click **Stop Debugging**.)

See also

- [LINQ to SQL tools in Visual Studio](#)
- [Walkthrough: Creating LINQ to SQL classes \(O/R Designer\)](#)
- [How to: Assign stored procedures to perform updates, inserts, and deletes \(O/R Designer\)](#)
- [LINQ to SQL](#)
- [How to: Generate the object model in Visual Basic or C#](#)

Walkthrough: Create an n-tier data application

8/5/2021 • 13 minutes to read • [Edit Online](#)

N-tier data applications are applications that access data and are separated into multiple logical layers, or *tiers*. Separating application components into discrete tiers increases the maintainability and scalability of the application. It does this by enabling easier adoption of new technologies that can be applied to a single tier without requiring you to redesign the whole solution. N-tier architecture includes a presentation tier, a middle-tier, and a data tier. The middle tier typically includes a data access layer, a business logic layer, and shared components such as authentication and validation. The data tier includes a relational database. N-tier applications usually store sensitive information in the data access layer of the middle-tier to maintain isolation from end users who access the presentation tier. For more information, see [N-tier data applications overview](#).

One way to separate the various tiers in an n-tier application is to create discrete projects for each tier that you want to include in your application. Typed datasets contain a `DataSet Project` property that determines which projects the generated dataset and `TableAdapter` code should go into.

This walkthrough demonstrates how to separate dataset and `TableAdapter` code into discrete class library projects by using the **Dataset Designer**. After you separate the dataset and `TableAdapter` code, you create a [Windows Communication Foundation Services and WCF Data Services in Visual Studio](#) service to call into the data access tier. Finally, you create a Windows Forms application as the presentation tier. This tier accesses data from the data service.

During this walkthrough, you perform the following steps:

- Create a new n-tier solution that contains multiple projects.
- Add two class library projects to the n-tier solution.
- Create a typed dataset by using the **Data Source Configuration Wizard**.
- Separate the generated [TableAdapters](#) and dataset code into discrete projects.
- Create a Windows Communication Foundation (WCF) service to call into the data access tier.
- Create functions in the service to retrieve data from the data access tier.
- Create a Windows Forms application to serve as the presentation tier.
- Create Windows Forms controls that are bound to the data source.
- Write code to populate the data tables.



For a video version of this topic, see [Video How to: Creating an n-tier data application](#).

Prerequisites

This walkthrough uses SQL Server Express LocalDB and the Northwind sample database.

1. If you don't have SQL Server Express LocalDB, install it either from the [SQL Server Express download page](#), or through the **Visual Studio Installer**. In the **Visual Studio Installer**, you can install SQL Server Express LocalDB as part of the **.NET desktop development** workload, or as an individual component.
2. Install the Northwind sample database by following these steps:

- a. In Visual Studio, open the **SQL Server Object Explorer** window. (SQL Server Object Explorer is installed as part of the **Data storage and processing** workload in the Visual Studio Installer.) Expand the **SQL Server** node. Right-click on your LocalDB instance and select **New Query**.

A query editor window opens.

- b. Copy the [Northwind Transact-SQL script](#) to your clipboard. This T-SQL script creates the Northwind database from scratch and populates it with data.
- c. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes running and the Northwind database is created.

Create the n-tier solution and class library to hold the dataset (DataEntityTier)

The first step of this walkthrough is to create a solution and two class library projects. The first class library holds the dataset (the generated typed `DataSet` class and DataTables that hold the application's data). This project is used as the data entity layer of the application and is typically located in the middle tier. The dataset creates the initial dataset and automatically separates the code into the two class libraries.

NOTE

Be sure to name the project and solution correctly before you click **OK**. Doing so will make it easier for you to complete this walkthrough.

To create the n-tier solution and DataEntityTier class library

1. In Visual Studio, on the **File** menu, select **New > Project**.
2. Expand either **Visual C#** or **Visual Basic** in the left-hand pane, then select **Windows Desktop**.
3. In the middle pane, select the **Class Library** project type.
4. Name the project **DataEntityTier**.
5. Name the solution **NTierWalkthrough**, and then choose **OK**.

An NTierWalkthrough solution that contains the DataEntityTier project is created and added to **Solution Explorer**.

Create the class library to hold the TableAdapters (DataAccessTier)

The next step after you create the DataEntityTier project is to create another class library project. This project holds the generated TableAdapters and is called the *data access tier* of the application. The data access tier contains the information that is required to connect to the database and is typically located in the middle tier.

To create a separate class library for the TableAdapters

1. Right-click on the solution in **Solution Explorer** and choose **Add > New Project**.
2. In the **New Project** dialog box, in the middle pane, select **Class Library**.
3. Name the project **DataAccessTier** and choose **OK**.

The DataAccessTier project is created and added to the NTierWalkthrough solution.

Create the Dataset

The next step is to create a typed dataset. Typed datasets are created with both the dataset class (including `DataTables` classes) and the `TableAdapter` classes in a single project. (All classes are generated into a single file.) When you separate the dataset and TableAdapters into different projects, it is the dataset class that is moved to the other project, leaving the `TableAdapter` classes in the original project. Therefore, create the dataset in the project that will ultimately contain the TableAdapters (the `DataAccessTier` project). You create the dataset by using the **Data Source Configuration Wizard**.

NOTE

You must have access to the Northwind sample database to create the connection. For information about how to set up the Northwind sample database, see [How to: Install sample databases](#).

To create the dataset

1. Select the `DataAccessTier` in **Solution Explorer**.
2. On the **Data** menu, select **Show Data Sources**.

The **Data Sources** window opens.

3. In the **Data Sources** window, select **Add New Data Source** to start the **Data Source Configuration Wizard**.
4. On the **Choose a Data Source Type** page, select **Database** and then select **Next**.
5. On the **Choose Your Data Connection** page, perform one of the following actions:

If a data connection to the Northwind sample database is available in the drop-down list, select it.

-or-

Select **New Connection** to open the **Add Connection** dialog box.

6. If the database requires a password, select the option to include sensitive data, and then choose **Next**.

NOTE

If you selected a local database file (instead of connecting to SQL Server) you might be asked if you want to add the file to the project. Choose **Yes** to add the database file to the project.

7. Select **Next** on the **Save the Connection String to the Application Configuration File** page.
8. Expand the **Tables** node on the **Choose Your Database Objects** page.
9. Select the check boxes for the **Customers** and **Orders** tables, and then choose **Finish**.

`NorthwindDataSet` is added to the `DataAccessTier` project and appears in the **Data Sources** window.

Separate the TableAdapters from the Dataset

After you create the dataset, separate the generated dataset class from the TableAdapters. You do this by setting the **DataSet Project** property to the name of the project in which to store the separated out dataset class.

To separate the TableAdapters from the Dataset

1. Double-click `NorthwindDataSet.xsd` in **Solution Explorer** to open the dataset in the **Dataset Designer**.
2. Select an empty area on the designer.

3. Locate the **DataSet Project** node in the **Properties** window.
4. In the **DataSet Project** list, select **DataEntityTier**.
5. On the **Build** menu, select **Build Solution**.

The dataset and TableAdapters are separated into the two class library projects. The project that originally contained the whole dataset (`DataAccessTier`) now contains only the TableAdapters. The project designated in the **DataSet Project** property (`DataEntityTier`) contains the typed dataset: *NorthwindDataSet.Dataset.Designer.vb* (or *NorthwindDataSet.Dataset.Designer.cs*).

NOTE

When you separate datasets and TableAdapters (by setting the **DataSet Project** property), existing partial dataset classes in the project will not be moved automatically. Existing dataset partial classes must be manually moved to the dataset project.

Create a New Service Application

This walkthrough demonstrates how to access the data access tier by using a WCF service, so let's create a new WCF service application.

To create a new WCF Service application

1. Right-click on the solution in **Solution Explorer** and choose **Add > New Project**.
2. In the **New Project** dialog box, in the left-hand pane, select **WCF**. In the middle pane, select **WCF Service Library**.
3. Name the project **DataService** and select **OK**.

The DataService project is created and added to the NTierWalkthrough solution.

Create methods in the data access tier to return the customers and orders data

The data service has to call two methods in the data access tier: `GetCustomers` and `GetOrders`. These methods return the Northwind `Customers` and `Orders` tables. Create the `GetCustomers` and `GetOrders` methods in the `DataAccessTier` project.

To create a method in the data access tier that returns the Customers table

1. In **Solution Explorer**, double-click **NorthwindDataset.xsd** to open the dataset.
2. Right-click **CustomersTableAdapter** and click **Add Query**.
3. On the **Choose a Command Type** page, leave the default value of **Use SQL statements** and click **Next**.
4. On the **Choose a Query Type** page, leave the default value of **SELECT which returns rows** and click **Next**.
5. On the **Specify a SQL SELECT statement** page, leave the default query and click **Next**.
6. On the **Choose Methods to Generate** page, type `GetCustomers` for the **Method name** in the **Return a DataTable** section.
7. Click **Finish**.

To create a method in the data access tier that returns the Orders table

1. Right-click **OrdersTableAdapter** and click **Add Query**.
2. On the **Choose a Command Type** page, leave the default value of **Use SQL statements** and click **Next**.
3. On the **Choose a Query Type** page, leave the default value of **SELECT which returns rows** and click **Next**.
4. On the **Specify a SQL SELECT statement** page, leave the default query and click **Next**.
5. On the **Choose Methods to Generate** page, type **GetOrders** for the **Method name** in the **Return a DataTable** section.
6. Click **Finish**.
7. On the **Build** menu, click **Build Solution**.

Add a reference to the data entity and data access tiers to the data service

Because the data service requires information from the dataset and TableAdapters, add references to the **DataEntityTier** and **DataAccessTier** projects.

To add references to the data service

1. Right-click **DataService** in **Solution Explorer** and click **Add Reference**.
2. Click the **Projects** tab in the **Add Reference** dialog box.
3. Select both the **DataAccessTier** and **DataEntityTier** projects.
4. Click **OK**.

Add functions to the service to call the GetCustomers and GetOrders methods in the data access tier

Now that the data access tier contains the methods to return data, create methods in the data service to call the methods in the data access tier.

NOTE

For C# projects, you must add a reference to the `System.Data.DataSetExtensions` assembly for the following code to compile.

To create the GetCustomers and GetOrders functions in the data service

1. In the **DataService** project, double-click **IService1.vb** or **IService1.cs**.
2. Add the following code under the **Add your service operations here** comment:

```
<OperationContract> _  
Function GetCustomers() As DataEntityTier.NorthwindDataSet.CustomersDataTable  
  
<OperationContract> _  
Function GetOrders() As DataEntityTier.NorthwindDataSet.OrdersDataTable
```



```
[OperationContract]
DataEntityTier.NorthwindDataSet.CustomersDataTable GetCustomers();

[OperationContract]
DataEntityTier.NorthwindDataSet.OrdersDataTable GetOrders();
```

3. In the DataService project, double-click **Service1.vb** (or **Service1.cs**).
4. Add the following code to the **Service1** class:

```
Public Function GetCustomers() As DataEntityTier.NorthwindDataSet.CustomersDataTable Implements
IService1.GetCustomers
    Dim CustomersTableAdapter1 As New
DataAccessTier.NorthwindDataSetTableAdapters.CustomersTableAdapter
    Return CustomersTableAdapter1.GetCustomers()
End Function

Public Function GetOrders() As DataEntityTier.NorthwindDataSet.OrdersDataTable Implements
IService1.GetOrders
    Dim OrdersTableAdapter1 As New DataAccessTier.NorthwindDataSetTableAdapters.OrdersTableAdapter
    Return OrdersTableAdapter1.GetOrders()
End Function
```

```
public DataEntityTier.NorthwindDataSet.CustomersDataTable GetCustomers()
{
    DataAccessTier.NorthwindDataSetTableAdapters.CustomersTableAdapter
CustomersTableAdapter1
    = new DataAccessTier.NorthwindDataSetTableAdapters.CustomersTableAdapter();
    return CustomersTableAdapter1.GetCustomers();
}
public DataEntityTier.NorthwindDataSet.OrdersDataTable GetOrders()
{
    DataAccessTier.NorthwindDataSetTableAdapters.OrdersTableAdapter
OrdersTableAdapter1
    = new DataAccessTier.NorthwindDataSetTableAdapters.OrdersTableAdapter();
    return OrdersTableAdapter1.GetOrders();
}
```

5. On the **Build** menu, click **Build Solution**.

Create a presentation tier to display data from the data service

Now that the solution contains the data service that has methods, which call into the data access tier, create another project that calls into the data service and present the data to users. For this walkthrough, create a Windows Forms application; this is the presentation tier of the n-tier application.

To create the presentation tier project

1. Right-click on the solution in **Solution Explorer** and choose **Add > New Project**.
2. In the **New Project** dialog box, in the left-hand pane, select **Windows Desktop**. In the middle pane, select **Windows Forms App**.
3. Name the project **PresentationTier** and click **OK**.

The PresentationTier project is created and added to the NTierWalkthrough solution.

Set the PresentationTier project as the startup project

We'll set the **PresentationTier** project to be the startup project for the solution, because it's the actual client

application that presents and interacts with the data.

To set the new presentation tier project as the startup project

- In **Solution Explorer**, right-click **PresentationTier** and click **Set as Startup Project**.

Add References to the Presentation Tier

The client application, **PresentationTier** requires a service reference to the data service in order to access the methods in the service. In addition, a reference to the dataset is required to enable type sharing by the WCF service. Until you enable type sharing through the data service, code added to the partial dataset class is not available to the presentation tier. Because you typically add code, such as validation code to the row and column changing events of a data table, it's likely that you'll want to access this code from the client.

To add a reference to the presentation tier

1. In **Solution Explorer**, right-click **PresentationTier** and select **Add Reference**.
2. In the **Add Reference** dialog box, select the **Projects** tab.
3. Select **DataEntityTier** and choose **OK**.

To add a service reference to the presentation tier

1. In **Solution Explorer**, right-click **PresentationTier** and select **Add Service Reference**.
2. In the **Add Service Reference** dialog box, select **Discover**.
3. Select **Service1** and choose **OK**.

NOTE

If you have multiple services on the current computer, select the service that you created previously in this walkthrough (the service that contains the `GetCustomers` and `GetOrders` methods).

Add DataGridViews to the form to display the data returned by the data service

After you add the service reference to the data service, the **Data Sources** window is automatically populated with the data that is returned by the service.

To add two data bound DataGridViews to the form

1. In **Solution Explorer**, select the **PresentationTier** project.
2. In the **Data Sources** window, expand **NorthwindDataSet** and locate the **Customers** node.
3. Drag the **Customers** node onto **Form1**.
4. In the **Data Sources** window, expand the **Customers** node and locate the related **Orders** node (the **Orders** node nested in the **Customers** node).
5. Drag the related **Orders** node onto **Form1**.
6. Create a `Form1_Load` event handler by double-clicking an empty area of the form.
7. Add the following code to the `Form1_Load` event handler.

```
Dim DataSvc As New ServiceReference1.Service1Client
NorthwindDataSet.Customers.Merge(DataSvc.GetCustomers)
NorthwindDataSet.Orders.Merge(DataSvc.GetOrders)
```

```
ServiceReference1.Service1Client DataSvc =
    new ServiceReference1.Service1Client();
northwindDataSet.Customers.Merge(DataSvc.GetCustomers());
northwindDataSet.Orders.Merge(DataSvc.GetOrders());
```

Increase the maximum message size allowed by the service

The default value for `maxReceivedMessageSize` is not large enough to hold the data retrieved from the `Customers` and `Orders` tables. In the following steps, you'll increase the value to 6553600. You change the value on the client, which automatically updates the service reference.

NOTE

The lower default size is intended to limit exposure to denial of service (DoS) attacks. For more information, see [MaxReceivedMessageSize](#).

To increase the `maxReceivedMessageSize` value

1. In **Solution Explorer**, double-click the **app.config** file in the **PresentationTier** project.
2. Locate the `maxReceivedMessage` size attribute and change the value to `6553600`.

Test the application

Run the application by pressing **F5**. The data from the `Customers` and `Orders` tables is retrieved from the data service and displayed on the form.

Next steps

Depending on your application requirements, there are several steps that you may want to perform after you save related data in the Windows-based application. For example, you could make the following enhancements to this application:

- Add validation to the dataset.
- Add additional methods to the service for updating data back to the database.

See also

- [Work with datasets in n-tier applications](#)
- [Hierarchical update](#)
- [Accessing data in Visual Studio](#)

Compatible database systems for Visual Studio

8/5/2021 • 3 minutes to read • [Edit Online](#)

To develop a data-connected application in Visual Studio, you typically install the database system on your local development machine, and then deploy the application and database to a production environment when they are ready. Visual Studio installs SQL Server Express LocalDB on your machine as part of the **Data storage and processing** workload. This LocalDB instance is useful for developing data-connected applications quickly and easily.

For a database system to be accessible from .NET applications and to be visible in Visual Studio data tools windows, it must have an ADO.NET data provider. A provider must specifically support Entity Framework if you plan to use Entity data models in your .NET application. Many providers are offered through the NuGet Package Manager or through the Visual Studio Marketplace.

If you are using Azure storage APIs, install the Azure storage emulators on your local machine during development in order to avoid charges until you are ready to deploy to production. For more information, see [Use the Azure Storage Emulator for development and testing](#).

The following list includes some of the more popular database systems that can be used in Visual Studio projects. The list is not exhaustive. For a list of third-party vendors that offer ADO.NET data providers that enable deep integration with Visual Studio tooling, see [ADO.NET Data Providers](#).

Microsoft SQL Server

SQL Server is the Microsoft flagship database offering. SQL Server 2016 delivers breakthrough performance, advanced security, and rich, integrated reporting and analytics. It ships in various editions that are designed for different uses: from highly scalable, high-performance business analytics, to use on a single computer. SQL Server Express is a full-featured edition of SQL Server that is tailored for redistribution and embedding. LocalDB is a simplified edition of SQL Server Express that requires no configuration and runs in your application's process. You can download either or both products from the [SQL Server Express download page](#). Many of the SQL examples in this section use SQL Server LocalDB. SQL Server Management Studio (SSMS) is a stand-alone database management application that has more functionality than what is provided in Visual Studio SQL Server Object Explorer. You can get SSMS from the previous link.

Oracle

You can download a paid or free edition of the Oracle database from the [Oracle technology network](#) page. For design-time support for Entity Framework and TableAdapters, you will need the [Oracle Developer tools for Visual Studio](#). Other official Oracle products, including the Oracle Instant Client, are available through the NuGet Package Manager. You can download Oracle sample schemas by following the instructions in the [Oracle online documentation](#).

MySQL

MySQL is a popular open-source database system that is widely used in enterprises and websites. Downloads for MySQL, MySQL for Visual Studio, and related products are at [MySQL on Windows](#). Third parties offer various Visual Studio extensions and stand-alone management applications for MySQL. You can browse the offerings in the NuGet Package Manager (**Tools > NuGet Package Manager > Manage NuGet Packages for Solution**).

PostgreSQL

PostgreSQL is a free, open-source object relational database system. To install it on Windows, you can download it from the [PostgreSQL download page](#). You can also build PostgreSQL from the source code. The PostgreSQL core system includes a C language interface. Many third parties provide NuGet packages for using PostgreSQL from .NET applications. You can browse the offerings in the NuGet Package Manager (**Tools > NuGet Package Manager > Manage NuGet Packages for Solution**). Perhaps, the most popular package is provided by npgsql.org.

SQLite

SQLite is an embedded SQL database engine that runs in the application's own process. You can download it from the [SQLite download page](#). Many third-party NuGet packages for SQLite are also available. You can browse the offerings in the NuGet Package Manager (**Tools > NuGet Package Manager > Manage NuGet Packages for Solution**).

Firebird

Firebird is an open-source SQL database system. You can download it from the [Firebird download page](#). An ADO.NET data provider is available through the NuGet Package Manager.

See also

- [Accessing data in Visual Studio](#)
- [How to determine the version and edition of SQL Server and its components](#)

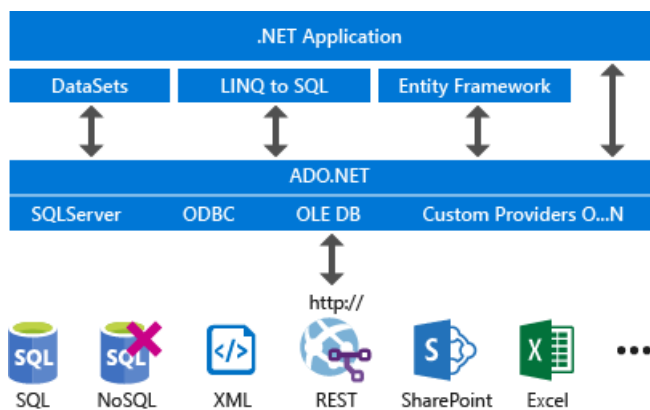
Visual Studio data tools for .NET

8/5/2021 • 2 minutes to read • [Edit Online](#)

Visual Studio and .NET together provide extensive API and tooling support for connecting to databases, modeling data in memory, and displaying the data in the user interface. The .NET classes that provide data-access functionality are known as [ADO.NET](#). ADO.NET, along with the data tooling in Visual Studio, was designed primarily to support relational databases and XML. These days, many NoSQL database vendors, or third parties, offer ADO.NET providers.

[.NET Core](#) supports ADO.NET, except for datasets and their related types. If you're targeting .NET Core and require an object-relational mapping (ORM) layer, use [Entity Framework Core](#).

The following diagram shows a simplified view of the basic architecture:



Typical workflow

The typical workflow is this:

1. Install a development or test database on your local machine. See [Installing database systems, tools, and samples](#). If you are using an Azure data service, this step is not necessary.
2. Test the connection to the database (or service or local file) in Visual Studio. See [Add new connections](#).
3. (Optional) Use the tools to generate and configure a new model. Models based on Entity Framework are the default recommendation for new applications. The model, whichever one you use, is the data source with which the application interacts. The model sits logically between the database or service and the application. See [Add new data sources](#).
4. Drag the data source from the **Data Sources** window onto a Windows Forms, ASP.NET, or Windows Presentation Foundation design surface to generate the data-binding code that will display the data to the user in the way that you specify. See [Bind controls to data in Visual Studio](#).
5. Add custom code for things like business rules, search, and data validation, or to take advantage of custom functionality that the underlying database exposes.

You can skip step 3 and program a .NET application to issue commands directly to a database, rather than using a model. In this case, you will find the relevant documentation here: [ADO.NET](#). Note that you still can use the **Data Source Configuration Wizard** and designers to generate data-binding code when you populate your own objects in memory and then data-bind UI controls to those objects.

See also

- [Access data in Visual Studio](#)

Entity Framework Tools in Visual Studio

8/5/2021 • 2 minutes to read • [Edit Online](#)

Entity Framework is an object-relational mapping technology that enables .NET developers to work with relational data by using domain-specific objects. It eliminates the need for most of the data-access code that developers usually need to write. Entity Framework is the recommended object-relational mapping (ORM) modeling technology for new .NET applications.

Entity Framework Tools are designed to help you build Entity Framework (EF) applications. The complete documentation for Entity Framework is here: [Overview - EF 6](#).

NOTE

The Entity Framework Tools described on this page are used to generate *.edmx* files, which are not supported in EF Core. To generate an EF Core model from an existing database, see [Reverse Engineering - EF Core](#). For more information on the differences between EF 6 and EF Core, see [Compare EF 6 and EF Core](#).

With Entity Framework Tools, you can create a *conceptual model* from an existing database and then graphically visualize and edit your conceptual model. Or, you can graphically create a conceptual model first, and then generate a database that supports your model. In either case, you can automatically update your model when the underlying database changes and automatically generate object-layer code for your application. Database generation and object-layer code generation are customizable.

The Entity Framework tools are installed as part of the **Data storage and processing** workload in the Visual Studio Installer. You can also install them as an individual component under the **SDKs, libraries, and frameworks** category.

These are the specific tools that make up Entity Framework tools in Visual Studio:

- You can use the **ADO.NET Entity Data Model Designer (Entity Designer)** to visually create and modify entities, associations, mappings, and inheritance relationships. The **Entity Designer** also generates C# or Visual Basic object-layer code.
- You can use the **Entity Data Model Wizard** to generate a conceptual model from an existing database and add database connection information to your application.
- You can use the **Create Database Wizard** to create a conceptual model first and then create a database that supports the model.
- You can use the **Update Model Wizard** to update your conceptual model, storage model, and mappings when changes have been made to the underlying database.

NOTE

Starting with Visual Studio 2010, Entity Framework tools do not support SQL Server 2000.

The tools generate or modify an *.edmx* file. This *.edmx* file contains information that describes the conceptual model, the storage model, and the mappings between them. For more information, see [EDMX](#).

Entity Framework Power Tools help you build applications that use the Entity Data Model. The power tools can generate a conceptual model, validate an existing model, produce source-code files that contain object classes based on the conceptual model, and produce source-code files that contain views that the model generates. For

detailed information, see [Pre-Generated Mapping Views](#).

Related topics

| TITLE | DESCRIPTION |
|---|---|
| ADO.NET Entity Framework | Describes how to use Entity Data Model Tools, which Entity Framework provides, to create applications. |
| Entity Data Model | Provides links and information for working with data that is used by applications built on Entity Framework. |
| Entity Framework (EF) Documentation | Provides an index of videos, tutorials, and advanced documentation to help you make the most of Entity Framework. |
| ASP.NET 5 Application to New Database | Describes how to create a new ASP.NET 5 application by using Entity Framework 7. |

See also

- [Visual Studio data tools for .NET](#)

Dataset tools in Visual Studio

8/5/2021 • 2 minutes to read • [Edit Online](#)

NOTE

Datasets and related classes are legacy .NET technologies from the early 2000s that enable applications to work with data in memory while the applications are disconnected from the database. They are especially useful for applications that enable users to modify data and persist the changes back to the database. Although datasets have proven to be a very successful technology, we recommend that new .NET applications use Entity Framework. Entity Framework provides a more natural way to work with tabular data as object models, and it has a simpler programming interface.

A `DataSet` object is an in-memory object that is essentially a mini-database. It contains `DataTable`, `DataColumn`, and `DataRow` objects in which you can store and modify data from one or more databases without having to maintain an open connection. The dataset maintains information about changes to its data, so updates can be tracked and sent back to the database when your application becomes reconnected.

Datasets and related classes are defined in the `System.Data` namespace in the .NET API. You can create and modify datasets dynamically in code using ADO.NET. The documentation in this section shows how to work with datasets by using Visual Studio designers. Datasets that are created through designers use `TableAdapter` objects to interact with the database. Datasets that are created programmatically use `DataAdapter` objects. For information about creating datasets programmatically, see [DataAdapters and DataReaders](#).

If your application needs to only read data from a database, and not perform updates, adds, or deletes, you can usually get better performance by using a `DataReader` object to retrieve data into a generic `List` object or another collection object. If you are displaying the data, you can data-bind the user interface to the collection.

Dataset workflow

Visual Studio provides tooling to simplify working with datasets. The basic end-to-end workflow is:

- Use the [Data Sources window](#) to create a new dataset from one or more data sources. Use the **Dataset Designer** to configure the dataset and set its properties. For example, you need to specify which tables from the data source to include, and which columns from each table. Choose carefully to conserve the amount of memory that the dataset requires. For more information, see [Create and configure datasets](#).
- Specify the relationships between the tables so that foreign keys are handled correctly. For more information, see [Fill datasets by using TableAdapters](#).
- Use the **TableAdapter Configuration Wizard** to specify the query or stored procedure that populates the dataset, and what database operations (update, delete, and so on) to implement. For more information, see these topics:
 - [Fill datasets by using TableAdapters](#)
 - [Edit data in datasets](#)
 - [Validate data in datasets](#)
 - [Save data back to the database](#)
- Query and search the data in the dataset. For more information, see [Query datasets](#). LINQ to DataSet enables [LINQ \(Language-Integrated Query\)](#) over data in a `DataSet` object. For more information, see [LINQ to DataSet](#).

- Use the **Data Sources** window to bind user-interface controls to the dataset or its individual columns, and to specify which columns are user-editable. For more information, see [Bind controls to data in Visual Studio](#).

Datasets and N-tier architecture

For information about datasets in N-tier applications, see [Work with datasets in n-tier applications](#).

Datasets and XML

For information about converting datasets to and from XML, see [Read XML data into a dataset](#) and [Save a dataset as XML](#).

See also

- [Visual Studio data tools for .NET](#)

Typed vs. untyped datasets

8/5/2021 • 2 minutes to read • [Edit Online](#)

A typed dataset is a dataset that is first derived from the base [DataSet](#) class and then uses information from the **Dataset Designer**, which is stored in an .xsd file, to generate a new, strongly typed dataset class. Information from the schema (tables, columns, and so on) is generated and compiled into this new dataset class as a set of first-class objects and properties. Because a typed dataset inherits from the base [DataSet](#) class, the typed class assumes all of the functionality of the [DataSet](#) class and can be used with methods that take an instance of a [DataSet](#) class as a parameter.

An untyped dataset, in contrast, has no corresponding built-in schema. As in a typed dataset, an untyped dataset contains tables, columns, and so on—but those are exposed only as collections. (However, after you manually create the tables and other data elements in an untyped dataset, you can export the dataset's structure as a schema by using the dataset's [WriteXmlSchema](#) method.)

Contrast data access in typed and untyped datasets

The class for a typed dataset has an object model in which its properties take on the actual names of the tables and columns. For example, if you are working with a typed dataset, you can reference a column by using code such as the following:

```
// This accesses the CustomerID column in the first row of the Customers table.
string customerIDValue = northwindDataSet.Customers[0].CustomerID;
```

```
' This accesses the CustomerID column in the first row of the Customers table.
Dim customerIDValue As String = NorthwindDataSet.Customers(0).CustomerID
```

In contrast, if you are working with an untyped dataset, the equivalent code is:

```
string customerIDValue = (string)
    dataset1.Tables["Customers"].Rows[0]["CustomerID"];
```

```
Dim customerIDValue As String =
    CType(dataset1.Tables("Customers").Rows(0).Item("CustomerID"), String)
```

Typed access is not only easier to read, but also fully supported by IntelliSense in the Visual Studio **Code Editor**. In addition to being easier to work with, the syntax for the typed dataset provides type checking at compile time, greatly reducing the possibility of errors in assigning values to dataset members. If you change the name of a column in your [DataSet](#) class and then compile your application, you receive a build error. By double-clicking the build error in the **Task List**, you can go directly to the line or lines of code that reference the old column name. Access to tables and columns in a typed dataset is also slightly faster at run time because access is determined at compile time, not through collections at run time.

Even though typed datasets have many advantages, an untyped dataset is useful in a variety of circumstances. The most obvious scenario is when no schema is available for the dataset. This might occur, for example, if your application is interacting with a component that returns a dataset, but you do not know in advance what its structure is. Similarly, there are times when you are working with data that does not have a static, predictable structure. In that case, it is impractical to use a typed dataset, because you would have to regenerate the typed

dataset class with each change in the data structure.

More generally, there are many times when you might create a dataset dynamically without having a schema available. In that case, the dataset is simply a convenient structure in which you can keep information, as long as the data can be represented in a relational way. At the same time, you can take advantage of the dataset's capabilities, such as the ability to serialize the information to pass to another process, or to write out an XML file.

See also

- [Dataset tools](#)

Fill datasets by using TableAdapters

8/5/2021 • 9 minutes to read • [Edit Online](#)

A TableAdapter component fills a dataset with data from the database, based on one or more queries or stored procedures that you specify. TableAdapters can also perform adds, updates, and deletes on the database to persist changes that you make to the dataset. You can also issue global commands that are unrelated to any specific table.

NOTE

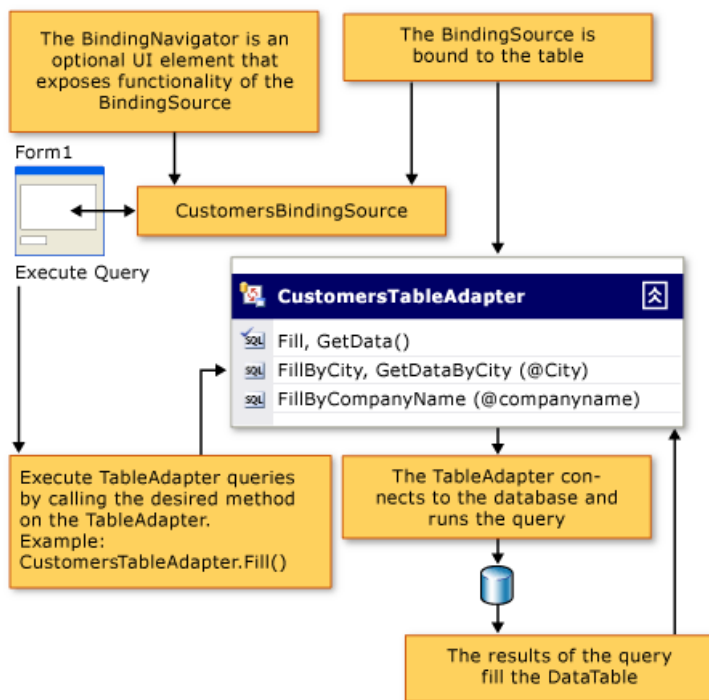
TableAdapters are generated by Visual Studio designers. If you are creating datasets programmatically, then use DataAdapter, which is a .NET class.

For detailed information about TableAdapter operations, you can skip directly to one of these topics:

| TOPIC | DESCRIPTION |
|---|---|
| Create and configure TableAdapters | How to use the designers to create and configure TableAdapters |
| Create parameterized TableAdapter queries | How to enable users to supply arguments to TableAdapter procedures or queries |
| Directly access the database with a TableAdapter | How to use the Dbdirect methods of TableAdapters |
| Turn off constraints while filling a dataset | How to work with foreign-key constraints when updating data |
| How to extend the functionality of a TableAdapter | How to add custom code to TableAdapters |
| Read XML data into a dataset | How to work with XML |

TableAdapter overview

TableAdapters are designer-generated components that connect to a database, run queries or stored procedures, and fill their DataTable with the returned data. TableAdapters also send updated data from your application back to the database. You can run as many queries as you want on a TableAdapter as long as they return data that conforms to the schema of the table with which the TableAdapter is associated. The following diagram shows how TableAdapters interact with databases and other objects in memory:



While TableAdapters are designed with the **Dataset Designer**, the TableAdapter classes are not generated as nested classes of **DataSet**. They are located in separate namespaces that are specific to each dataset. For example, if you have a dataset named `NorthwindDataSet`, the TableAdapters that are associated with **DataTables** in the `NorthwindDataSet` would be in the `NorthwindDataSetTableAdapters` namespace. To access a particular TableAdapter programmatically, you must declare a new instance of the TableAdapter. For example:

```
NorthwindDataSet northwindDataSet = new NorthwindDataSet();

NorthwindDataSetTableAdapters.CustomersTableAdapter customersTableAdapter =
    new NorthwindDataSetTableAdapters.CustomersTableAdapter();

customersTableAdapter.Fill(northwindDataSet.Customers);
```

```
Dim northwindDataSet As New NorthwindDataSet()
Dim customersTableAdapter As New NorthwindDataSetTableAdapters.CustomersTableAdapter()

customersTableAdapter.Fill(northwindDataSet.Customers)
```

Associated DataTable schema

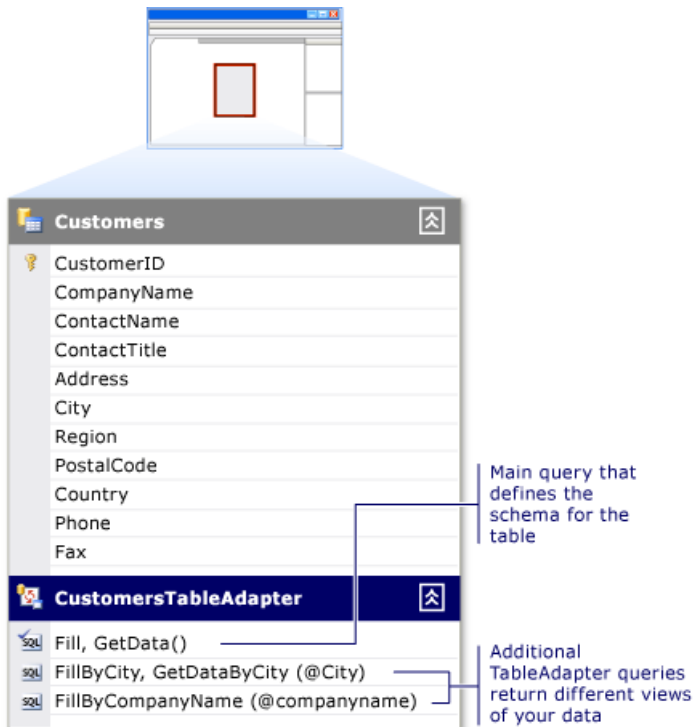
When you create a TableAdapter, you use the initial query or stored procedure to define the schema of the TableAdapter's associated **DataTable**. You run this initial query or stored procedure by calling the TableAdapter's **Fill** method (which fills the TableAdapter's associated **DataTable**). Any changes that are made to the TableAdapter's main query are reflected in the schema of the associated data table. For example, removing a column from the main query also removes the column from the associated data table. If any additional queries on the TableAdapter use SQL statements that return columns that are not in the main query, the designer attempts to synchronize the column changes between the main query and the additional queries.

TableAdapter update commands

The update functionality of a TableAdapter is dependent on how much information is available in the main query in the **TableAdapter Wizard**. For example, TableAdapters that are configured to fetch values from multiple tables (using a `JOIN`), scalar values, views, or the results of aggregate functions are not initially created with the ability to send updates back to the underlying database. However, you can configure the **INSERT**,

`UPDATE` , and `DELETE` commands manually in the **Properties** window.

TableAdapter queries



TableAdapters can contain multiple queries to fill their associated data tables. You can define as many queries for a TableAdapter as your application requires, as long as each query returns data that conforms to the same schema as its associated data table. This capability enables a TableAdapter to load different results based on differing criteria.

For example, if your application contains a table with customer names, you can create a query that fills the table with every customer name that begins with a certain letter, and another that fills the table with all customers that are located in the same state. To fill a `Customers` table with customers in a given state, you can create a `FillByState` query that takes a parameter for the state value as follows:

`SELECT * FROM Customers WHERE State = @State` . You run the query by calling the `FillByState` method and passing in the parameter value like this: `CustomerTableAdapter.FillByState("WA")` .

In addition to adding queries that return data of the same schema as the TableAdapter's data table, you can add queries that return scalar (single) values. For example, a query that returns a count of customers (`SELECT Count(*) From Customers`) is valid for a `CustomersTableAdapter`, even though the data that's returned doesn't conform to the table's schema.

ClearBeforeFill property

By default, every time you run a query to fill a TableAdapter's data table, the existing data is cleared, and only the results of the query are loaded into the table. Set the TableAdapter's `ClearBeforeFill` property to `false` if you want to add or merge the data that's returned from a query to the existing data in a data table. Regardless of whether you clear the data, you need to explicitly send updates back to the database, if you want to persist them. So remember to save any changes to the data in the table before running another query that fills the table. For more information, see [Update data by using a TableAdapter](#).

TableAdapter inheritance

TableAdapters extend the functionality of standard data adapters by encapsulating a configured [DataAdapter](#) class. By default, the TableAdapter inherits from the [Component](#) class and can't be cast to the [DataAdapter](#) class.

Casting a TableAdapter to the [DataAdapter](#) class results in an [InvalidCastException](#) error. To change the base class of a TableAdapter, you can specify a class that derives from [Component](#) in the **Base Class** property of the TableAdapter in the **Dataset Designer**.

TableAdapter methods and properties

The TableAdapter class is not a .NET type. This means you can't look it up in the documentation or the **Object Browser**. It's created at design time when you use one of the wizards mentioned earlier. The name that's assigned to a TableAdapter when you create it is based on the name of the table you are working with. For example, when you create a TableAdapter based on a table in a database named `Orders`, the TableAdapter is named `OrdersTableAdapter`. The class name of the TableAdapter can be changed using the **Name** property in the **Dataset Designer**.

Following are the commonly used methods and properties of TableAdapters:

| MEMBER | DESCRIPTION |
|---|--|
| <code>TableAdapter.Fill</code> | Populates the TableAdapter's associated data table with the results of the TableAdapter's <code>SELECT</code> command. |
| <code>TableAdapter.Update</code> | Sends changes back to the database and returns an integer that represents the number of rows affected by the update. For more information, see Update data by using a TableAdapter . |
| <code>TableAdapter.GetData</code> | Returns a new DataTable that's filled with data. |
| <code>TableAdapter.Insert</code> | Creates a new row in the data table. For more information, see Insert new records into a database . |
| <code>TableAdapter.ClearBeforeFill</code> | Determines whether a data table is emptied before you call one of the <code>Fill</code> methods. |

TableAdapter update method

TableAdapters use data commands to read to and write from the database. Use the TableAdapter's initial `Fill` (main) query as the basis for creating the schema of the associated data table, as well as the `InsertCommand`, `UpdateCommand`, and `DeleteCommand` commands that are associated with the `TableAdapter.Update` method. Calling a TableAdapter's `Update` method runs the statements that were created when the TableAdapter was originally configured, not one of the additional queries that you added with the **TableAdapter Query Configuration Wizard**.

When you use a TableAdapter, it effectively performs the same operations with the commands that you would typically perform. For example, when you call the adapter's `Fill` method, the adapter runs the data command in its `SelectCommand` property and uses a data reader (for example, [SqlDataReader](#)) to load the result set into the data table. Similarly, when you call the adapter's `Update` method, it runs the appropriate command (in the `UpdateCommand`, `InsertCommand`, and `DeleteCommand` properties) for each changed record in the data table.

NOTE

If there is enough information in the main query, the `InsertCommand`, `UpdateCommand`, and `DeleteCommand` commands are created by default when the `TableAdapter` is generated. If the `TableAdapter`'s main query is more than a single table `SELECT` statement, it's possible the designer won't be able to generate `InsertCommand`, `UpdateCommand`, and `DeleteCommand`. If these commands aren't generated, you might receive an error when running the `TableAdapter.Update` method.

TableAdapter GenerateDbDirectMethods

In addition to `InsertCommand`, `UpdateCommand`, and `DeleteCommand`, `TableAdapters` are created with methods that you can run directly against the database. You can call these methods (`TableAdapter.Insert`, `TableAdapter.Update`, and `TableAdapter.Delete`) directly to manipulate data in the database. This means you can call these individual methods from your code instead of calling `TableAdapter.Update` to handle the inserts, updates, and deletes that are pending for the associated data table.

If you don't want to create these direct methods, set the `TableAdapter`'s **GenerateDbDirectMethods** property to `false` (in the **Properties** window). Additional queries that are added to the `TableAdapter` are standalone queries — they don't generate these methods.

TableAdapter support for nullable types

`TableAdapters` support nullable types `Nullable(Of T)` and `T?`. For more information about nullable types in Visual Basic, see [Nullable Value Types](#). For more information about nullable types in C#, see [Use nullable types](#).

TableAdapterManager reference

By default, a `TableAdapterManager` class generates when you create a dataset that contains related tables. To prevent the class from being generated, change the value of the `Hierarchical Update` property of the dataset to false. When you drag a table that has a relation onto the design surface of a Windows Form or WPF page, Visual Studio declares a member variable of the class. If you don't use databinding, you have to manually declare the variable.

The `TableAdapterManager` class is not a .NET type. Therefore, you cannot look it up in the documentation. It's created at design time as part of the dataset creation process.

The following are the frequently used methods and properties of the `TableAdapterManager` class:

| MEMBER | DESCRIPTION |
|---|--|
| <code>UpdateAll</code> method | Saves all data from all data tables. |
| <code>BackUpDataSetBeforeUpdate</code> property | Determines whether to create a backup copy of the dataset before executing the <code>TableAdapterManager.UpdateAll</code> method. Boolean. |
| <code>tableName</code> <code>TableAdapter</code> property | Represents a <code>TableAdapter</code> . The generated <code>TableAdapterManager</code> contains a property for each <code>TableAdapter</code> it manages. For example, a dataset with a <code>Customers</code> and <code>Orders</code> table generates with a <code>TableAdapterManager</code> that contains <code>CustomersTableAdapter</code> and <code>OrdersTableAdapter</code> properties. |

| MEMBER | DESCRIPTION |
|-----------------------------------|---|
| <code>UpdateOrder</code> property | <p>Controls the order of the individual insert, update, and delete commands. Set this to one of the values in the <code>TableAdapterManager.UpdateOrderOption</code> enumeration.</p> <p>By default, the <code>UpdateOrder</code> is set to InsertUpdateDelete. This means that inserts, then updates, and then deletes are performed for all tables in the dataset.</p> |

Security

When you use data commands with a `CommandType` property set to [Text](#), carefully check information that is sent from a client before passing it to your database. Malicious users might try to send (inject) modified or additional SQL statements in an effort to gain unauthorized access or damage the database. Before you transfer user input to a database, always verify that the information is valid. A best practice is to always use parameterized queries or stored procedures when possible.

See also

- [Dataset tools](#)

Work with datasets in n-tier applications

8/5/2021 • 2 minutes to read • [Edit Online](#)

N-tier data applications are data-centric applications that are separated into multiple logical layers (or *tiers*). In other words, an n-tier data application is an application that is separated into multiple projects, with the data access tier, the business logic tier, and the presentation tier each in its own project. For more information, see [N-Tier data applications overview](#).

Typed datasets have been enhanced so that the TableAdapters and dataset classes can be generated into discrete projects. This provides the ability to quickly separate application layers and generate n-tier data applications.

N-tier support in typed datasets enables iterative development of the application architecture to an n-tier design. It also removes the requirement to manually separate the code into more than one project. Start out designing the data layer by using the **Dataset Designer**. When you're ready to take the application architecture to an n-tiered design, set the **DataSet Project** property of a dataset to generate the dataset class into a separate project.

Reference

- [DataSet](#)
- [TypedTableBase<T>](#)

See also

- [N-Tier data applications overview](#)
- [Walkthrough: Creating an n-tier Data Application](#)
- [Add code to TableAdapters in n-tier applications](#)
- [Add code to datasets in n-tier applications](#)
- [Add validation to an n-tier dataset](#)
- [Separate datasets and TableAdapters into different projects](#)
- [Hierarchical update](#)
- [Dataset tools in Visual Studio](#)
- [Accessing data in Visual Studio](#)
- [Create and configure TableAdapters](#)
- [N-Tier and remote applications with LINQ to SQL](#)

Database projects and data-tier applications

8/5/2021 • 2 minutes to read • [Edit Online](#)

You can use database projects to create new databases, new data-tier applications (DACs), and to update existing databases and data-tier applications. Both database projects and DAC projects enable you to apply version control and project management techniques to your database development efforts in much the same way that you apply those techniques to managed or native code. You can help your development team manage changes to databases and database servers by creating a DAC project, database project, or a server project and putting it under version control. Members of your team can then check out files to make, build, and test changes in an isolated development environment, or sandbox, before sharing them with the team. To help ensure code quality, your team can finish and test all changes for a particular release of the database in a staging environment before you deploy the changes into production.

For a list of the database features that are supported by data-tier applications, see [DAC support for SQL Server objects](#). If you use features in your database that are not supported by data-tier applications, you should instead use a database project to manage changes to your database.

Common high-level tasks

| HIGH-LEVEL TASK | SUPPORTING CONTENT |
|---|--|
| Start development of a data-tier application: The concept of a data-tier application (DAC) was introduced with SQL Server 2008. A DAC contains the definition for a SQL Server database and the supporting instance objects that are used by a client-server or 3-tier application. A DAC includes database objects, such as tables and views, together with instance entities such as logins. You can use Visual Studio to create a DAC project, build a DAC package file, and send the DAC package file to a database administrator for deployment onto an instance of the SQL Server database engine. | <ul style="list-style-type: none">- Data-tier applications- SQL Server Management Studio |
| Performing iterative database development: Developers can check out parts of the project and update them in an isolated development environment. By using this type of environment, you can test your changes without affecting other members of the team. After the changes are complete, you check the files back into version control, where other team members can obtain your changes and build and deploy them to a test server. | <ul style="list-style-type: none">- Project-oriented offline database development (SQL Server Data Tools)- Transact-SQL debugger (SQL Server Management Studio) |
| Prototyping, verifying test results, and modifying database scripts and objects: You can use the Transact-SQL editor to perform any one of these common tasks. | <ul style="list-style-type: none">- Query and text editors (SQL Server Management Studio) |

See also

- [Visual Studio data tools for .NET](#)

N-tier data applications overview

8/5/2021 • 3 minutes to read • [Edit Online](#)

N-tier data applications are data applications that are separated into multiple *tiers*. Also called "distributed applications" and "multitier applications", n-tier applications separate processing into discrete tiers that are distributed between the client and the server. When you develop applications that access data, you should have a clear separation between the various tiers that make up the application.

A typical n-tier application includes a presentation tier, a middle tier, and a data tier. The easiest way to separate the various tiers in an n-tier application is to create discrete projects for each tier that you want to include in your application. For example, the presentation tier might be a Windows Forms application, whereas the data access logic might be a class library located in the middle tier. Additionally, the presentation layer might communicate with the data access logic in the middle tier through a service such as a web service. Separating application components into separate tiers increases the maintainability and scalability of the application. It does this by enabling easier adoption of new technologies that can be applied to a single tier without the requirement to redesign the whole solution. In addition, n-tier applications typically store sensitive information in the middle-tier, which maintains isolation from the presentation tier.

Visual Studio contains several features to help developers create n-tier applications:

- The dataset provides a **DataSet Project** property that enables you to separate the dataset (data entity layer) and TableAdapters (data access layer) into discrete projects.
- The [LINQ to SQL tools in Visual Studio](#) provides settings to generate the DataContext and data classes into separate namespaces. This enables logical separation of the data access and data entity tiers.
- [LINQ to SQL](#) provides the [Attach](#) method that enables you to bring together the DataContext from different tiers in an application. For more information, see [N-Tier and remote applications with LINQ to SQL](#).

Presentation tier

The *presentation tier* is the tier in which users interact with an application. It often contains additional application logic also. Typical presentation tier components include the following:

- Data binding components, such as the [BindingSource](#) and [BindingNavigator](#).
- Object representations of data, such as [LINQ to SQL](#) entity classes for use in the presentation tier.

The presentation tier typically accesses the middle tier by using a service reference (for example, a [Windows Communication Foundation Services and WCF Data Services in Visual Studio](#) application). The presentation tier does not directly access the data tier. The presentation tier communicates with the data tier by way of the data access component in the middle tier.

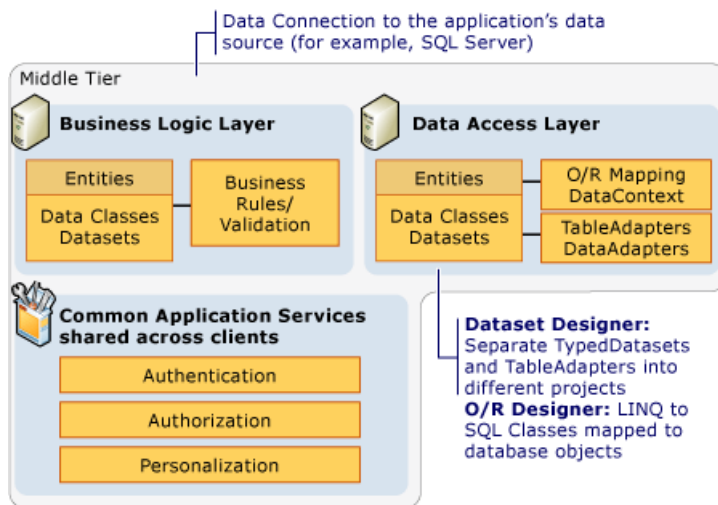
Middle tier

The *middle tier* is the layer that the presentation tier and the data tier use to communicate with each other. Typical middle tier components include the following:

- Business logic, such as business rules and data validation.
- Data access components and logic, such as the following:
 - [TableAdapters](#) and [DataAdapters and DataReaders](#).

- Object representations of data, such as [LINQ to SQL](#) entity classes.
- Common application services, such as authentication, authorization, and personalization.

The following illustration shows features and technologies that are available in Visual Studio and where they might fit in to the middle tier of an n-tier application.



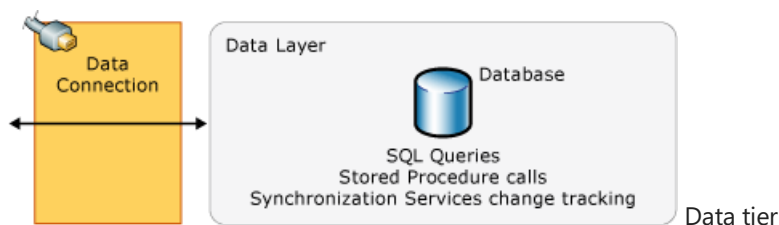
Middle tier

The middle tier typically connects to the data tier by using a data connection. This data connection is typically stored in the data access component.

Data tier

The *data tier* is basically the server that stores an application's data (for example, a server running SQL Server).

The following illustration shows features and technologies that are available in Visual Studio and where they might fit in to the data tier of an n-tier application.



Data tier

The data tier cannot be accessed directly from the client in the presentation tier. Instead, the data access component in the middle tier is used for communication between the presentation and data tiers.

Help for n-tier development

The following topics provide information about working with n-tier applications:

[Separate datasets and TableAdapters into different projects](#)

[Walkthrough: Creating an n-tier data application](#)

[N-tier and remote applications with LINQ to SQL](#)

See also

- [Walkthrough: Creating an n-tier data application](#)
- [Hierarchical update](#)
- [Dataset tools in Visual Studio](#)

- [Accessing data in Visual Studio](#)

Create a database and add tables in Visual Studio

8/5/2021 • 5 minutes to read • [Edit Online](#)

You can use Visual Studio to create and update a local database file in SQL Server Express LocalDB. You can also create a database by executing Transact-SQL statements in the **SQL Server Object Explorer** tool window in Visual Studio. In this topic, we'll create an *.mdf* file and add tables and keys by using the Table Designer.

Prerequisites

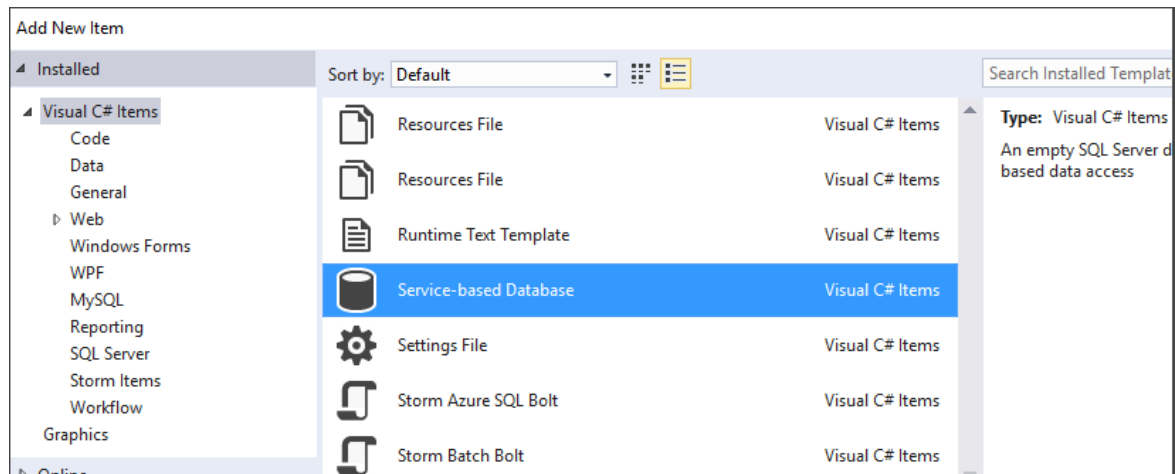
To complete this walkthrough, you'll need the **.NET desktop development** and **Data storage and processing** workloads installed in Visual Studio. To install them, open **Visual Studio Installer** and choose **Modify** (or **More > Modify**) next to the version of Visual Studio you want to modify.

NOTE

The procedures in this article apply only to .NET Framework Windows Forms projects, not to .NET Core Windows Forms projects.

Create a project and a local database file

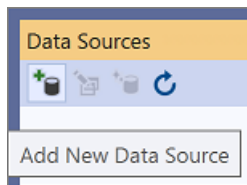
1. Create a new **Windows Forms App (.NET Framework)** project and name it **SampleDatabaseWalkthrough**.
2. On the menu bar, select **Project > Add New Item**.
3. In the list of item templates, scroll down and select **Service-based Database**.



4. Name the database **SampleDatabase**, and then click **Add**.

Add a data source

1. If the **Data Sources** window isn't open, open it by pressing **Shift+Alt+D** or selecting **View > Other Windows > Data Sources** on the menu bar.
2. In the **Data Sources** window, select **Add New Data Source**.



The **Data Source Configuration Wizard** opens.

3. On the **Choose a Data Source Type** page, choose **Database** and then choose **Next**.
4. On the **Choose a Database Model** page, choose **Next** to accept the default (Dataset).
5. On the **Choose Your Data Connection** page, select the **SampleDatabase.mdf** file in the drop-down list, and then choose **Next**.
6. On the **Save the Connection String to the Application Configuration File** page, choose **Next**.
7. On the **Choose your Database Objects** page, you'll see a message that says the database doesn't contain any objects. Choose **Finish**.

View properties of the data connection

You can view the connection string for the *SampleDatabase.mdf* file by opening the Properties window of the data connection:

- Select **View** > **SQL Server Object Explorer** to open the **SQL Server Object Explorer** window. Expand **(localdb)\MSSQLLocalDB > Databases**, and then right-click on *SampleDatabase.mdf* and select **Properties**.
- Alternatively, you can select **View** > **Server Explorer**, if that window isn't already open. Open the Properties window by expanding the **Data Connections** node, right-clicking on *SampleDatabase.mdf*, and then selecting **Properties**.

TIP

If you can't expand the Data Connections node, or the *SampleDatabase.mdf* connection is not listed, select the **Connect to Database** button in the Server Explorer toolbar. In the **Add Connection** dialog box, make sure that **Microsoft SQL Server Database File** is selected under **Data source**, and then browse to and select the *SampleDatabase.mdf* file. Finish adding the connection by selecting **OK**.

Create tables and keys by using Table Designer

In this section, you'll create two tables, a primary key in each table, and a few rows of sample data. You'll also create a foreign key to specify how records in one table correspond to records in the other table.

Create the Customers table

1. In **Server Explorer**, expand the **Data Connections** node, and then expand the **SampleDatabase.mdf** node.

If you can't expand the Data Connections node, or the *SampleDatabase.mdf* connection is not listed, select the **Connect to Database** button in the Server Explorer toolbar. In the **Add Connection** dialog box, make sure that **Microsoft SQL Server Database File** is selected under **Data source**, and then browse to and select the *SampleDatabase.mdf* file. Finish adding the connection by selecting **OK**.

2. Right-click on **Tables** and select **Add New Table**.

The Table Designer opens and shows a grid with one default row, which represents a single column in the table that you're creating. By adding rows to the grid, you'll add columns in the table.

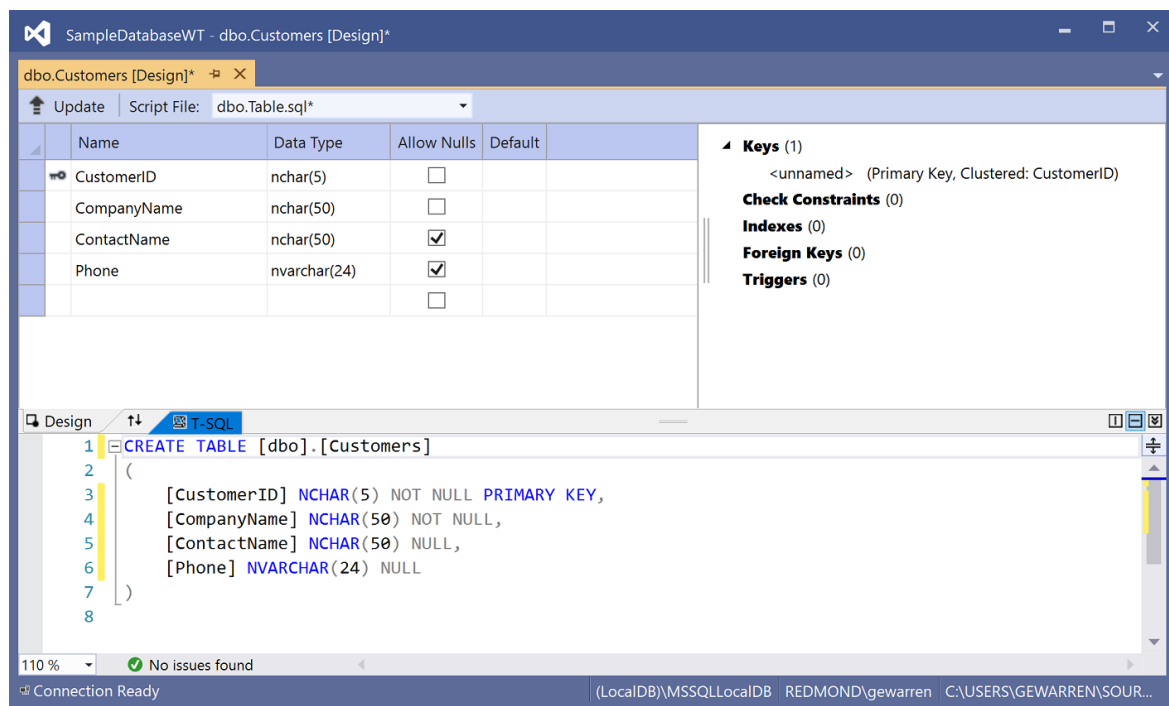
- In the grid, add a row for each of the following entries:

| COLUMN NAME | DATA TYPE | ALLOW NULLS |
|-------------|---------------|-----------------|
| CustomerID | nchar(5) | False (cleared) |
| CompanyName | nvarchar(50) | False (cleared) |
| ContactName | nvarchar (50) | True (selected) |
| Phone | nvarchar (24) | True (selected) |

- Right-click on the **CustomerID** row, and then select **Set Primary Key**.
- Right-click on the default row (**Id**), and then select **Delete**.
- Name the Customers table by updating the first line in the script pane to match the following sample:

```
CREATE TABLE [dbo].[Customers]
```

You should see something like this:



- In the upper-left corner of **Table Designer**, select **Update**.
- In the **Preview Database Updates** dialog box, select **Update Database**.

The Customers table is created in the local database file.

Create the Orders table

- Add another table, and then add a row for each entry in the following table:

| COLUMN NAME | DATA TYPE | ALLOW NULLS |
|-------------|-----------|-----------------|
| OrderID | int | False (cleared) |

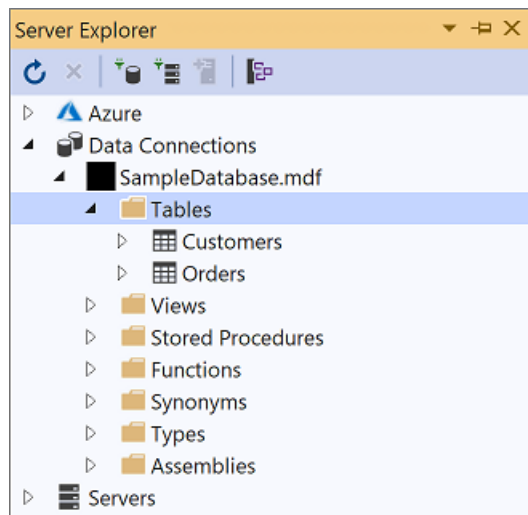
| COLUMN NAME | DATA TYPE | ALLOW NULLS |
|---------------|-----------|-----------------|
| CustomerID | nchar(5) | False (cleared) |
| OrderDate | datetime | True (selected) |
| OrderQuantity | int | True (selected) |

- Set **OrderID** as the primary key, and then delete the default row.
- Name the Orders table by updating the first line in the script pane to match the following sample:

```
CREATE TABLE [dbo].[Orders]
```

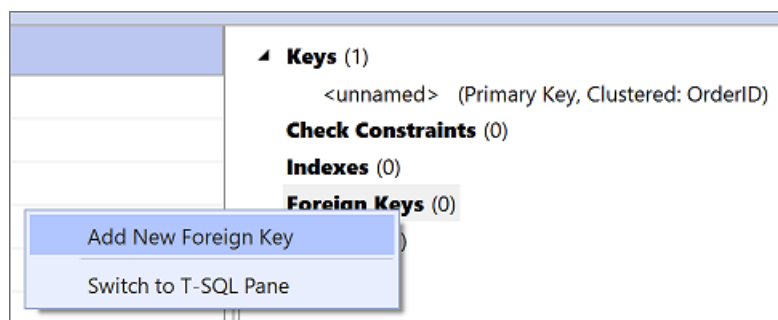
- In the upper-left corner of the **Table Designer**, select **Update**.
- In the **Preview Database Updates** dialog box, select **Update Database**.

The Orders table is created in the local database file. If you expand the **Tables** node in Server Explorer, you see the two tables:



Create a foreign key

- In the context pane on the right side of the Table Designer grid for the Orders table, right-click on **Foreign Keys** and select **Add New Foreign Key**.



- In the text box that appears, replace the text **ToTable** with **Customers**.
- In the T-SQL pane, update the last line to match the following sample:

```
CONSTRAINT [FK_Orders_Customers] FOREIGN KEY ([CustomerID]) REFERENCES [Customers]([CustomerID])
```

4. In the upper-left corner of the **Table Designer**, select **Update**.
5. In the **Preview Database Updates** dialog box, select **Update Database**.

The foreign key is created.

Populate the tables with data

1. In **Server Explorer** or **SQL Server Object Explorer**, expand the node for the sample database.
2. Open the shortcut menu for the **Tables** node, select **Refresh**, and then expand the **Tables** node.
3. Open the shortcut menu for the Customers table, and then select **Show Table Data**.
4. Add whatever data you want for some customers.

You can specify any five characters you want as the customer IDs, but choose at least one that you can remember for use later in this procedure.

5. Open the shortcut menu for the Orders table, and then select **Show Table Data**.
6. Add data for some orders.

IMPORTANT

Make sure that all order IDs and order quantities are integers and that each customer ID matches a value that you specified in the **CustomerID** column of the Customers table.

7. On the menu bar, select **File > Save All**.

See also

- [Accessing data in Visual Studio](#)

Add new connections

8/5/2021 • 2 minutes to read • [Edit Online](#)

You can test your connection to a database or service, and explore database contents and schemas, by using **Server Explorer**, **Cloud Explorer**, or **SQL Server Object Explorer**. The functionality of these windows overlaps to some extent. The basic differences are:

- **Server Explorer**

Installed by default in Visual Studio. Can be used to test connections and view SQL Server databases, any other databases that have an ADO.NET provider installed, and some Azure services. Also shows low-level objects such as system performance counters, event logs, and message queues. If a data source has no ADO.NET provider, it won't show up here, but you can still use it from Visual Studio by connecting programmatically.

- **Cloud Explorer**

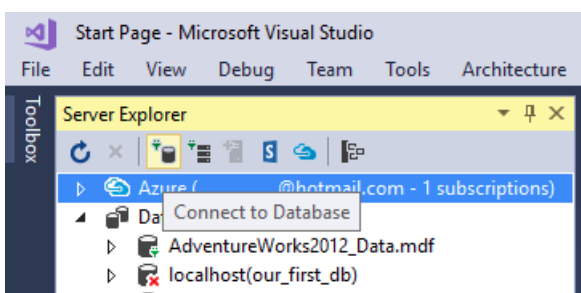
Install this window manually as a Visual Studio extension from [Visual Studio Marketplace](#). Provides specialized functionality for exploring and connecting to Azure services.

- **SQL Server Object Explorer**

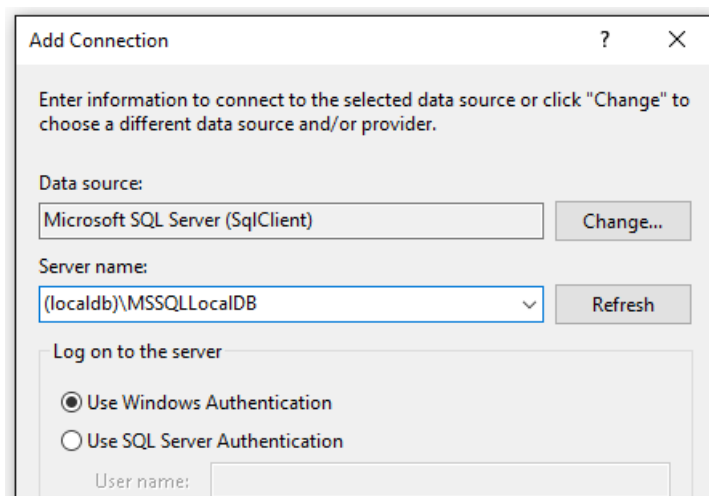
Installed with SQL Server Data Tools and visible under the **View** menu. If you don't see it there, go to **Programs and Features** in Control Panel, find Visual Studio, and then select **Change** to re-run the installer after selecting the check box for SQL Server Data Tools. Use **SQL Server Object Explorer** to view SQL databases (if they have an ADO.NET provider), create new databases, modify schemas, create stored procedures, retrieve connection strings, view the data, and more. SQL databases that have no ADO.NET provider installed won't show up here, but you can still connect to them programmatically.

Add a connection in Server Explorer

To create a connection to the database, click the **Add Connection** icon in **Server Explorer**, or right-click in **Server Explorer** on the **Data Connections** node and select **Add Connection**. From here, you can also connect to a database on another server, a SharePoint service, or an Azure service.



This brings up the **Add Connection** dialog box. Here, we have entered the name of the SQL Server LocalDB instance.



Add Connection ? X

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:

Server name:

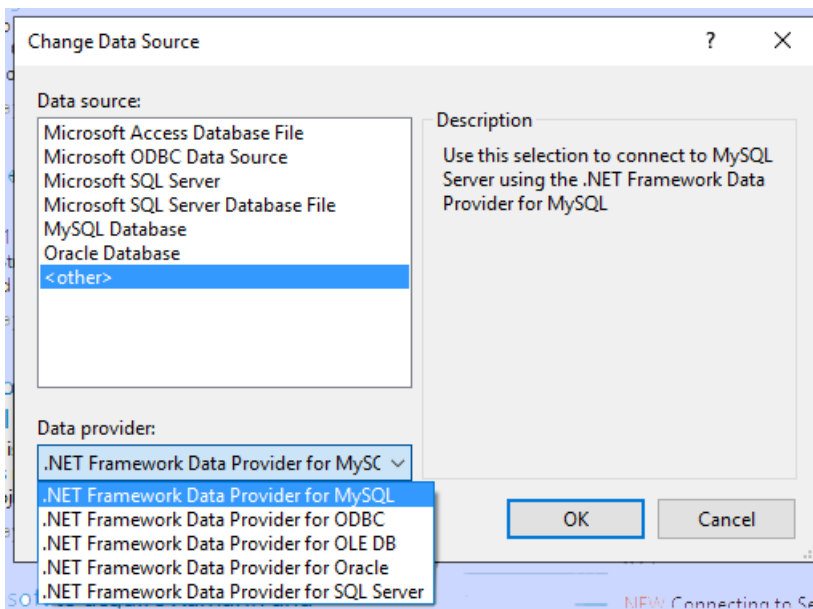
Log on to the server

☒ Use Windows Authentication
☐ Use SQL Server Authentication

User name:

Change the provider

If the data source is not what you want, click the **Change** button to choose a new data source and/or a new ADO.NET data provider. The new provider might ask for your credentials, depending on how you configured it.



Change Data Source ? X

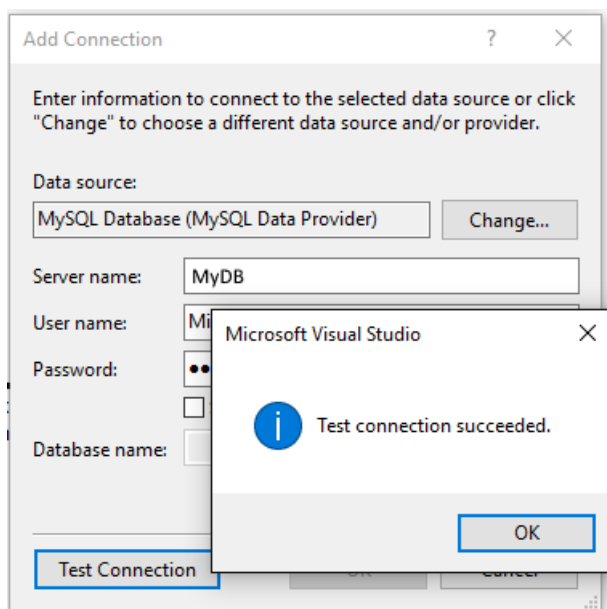
| Data source: | Description |
|------------------------------------|--|
| Microsoft Access Database File | Use this selection to connect to MySQL Server using the .NET Framework Data Provider for MySQL |
| Microsoft ODBC Data Source | |
| Microsoft SQL Server | |
| Microsoft SQL Server Database File | |
| MySQL Database | |
| Oracle Database | |
| <other> | |

Data provider:

- .NET Framework Data Provider for MySQL
- .NET Framework Data Provider for ODBC
- .NET Framework Data Provider for OLE DB
- .NET Framework Data Provider for Oracle
- .NET Framework Data Provider for SQL Server

Test the connection

After you have chosen the data source, click **Test Connection**. If it doesn't succeed, you will need to troubleshoot based on the vendor's documentation.



If the test succeeds, you are ready to create a *data source*, which is a Visual Studio term that really means a *data model* that is based on the underlying database or service.

See also

- [Visual Studio data tools for .NET](#)

How to: Save and edit connection strings

8/5/2021 • 2 minutes to read • [Edit Online](#)

Connection strings in Visual Studio applications are saved in the application configuration file (also referred to as application settings), or hard-coded directly in your application. Saving connection strings in the application configuration file simplifies the task of maintaining your application. If the connection string needs to be changed, you can update it in the application settings file (as opposed to having to change it in the source code and recompile the application).

Storing sensitive information (such as the password) within the connection string can affect the security of your application. Connection strings saved to the application configuration file are not encrypted or obfuscated, so it may be possible for someone to access the file and view its contents. Using Windows integrated security is a more secure way to control access to a database.

If you do not choose to use Windows integrated security and your database requires a user name and password, you can omit them from the connection string, but your application will need to provide this information to successfully connect to the database. For example, you can create a dialog box that prompts the user for this information and dynamically builds the connection string at run time. Security can still be an issue if the information is intercepted on the way to the database. For more information, see [Protecting connection information](#).

To save a connection string from within the Data Source Configuration Wizard

In the **Data Source Configuration Wizard**, select the option to save the connection on the **Save the Connection String to the Application Configuration File** page.

To save a connection string directly into application settings

1. In **Solution Explorer**, double-click the **My Project** icon (Visual Basic) or **Properties** icon (C#) to open the **Project Designer**.
2. Select the **Settings** tab.
3. Enter a **Name** for the connection string. Refer to this name when accessing the connection string in code.
4. Set the **Type** to **(Connection string)**.
5. Leave the **Scope** set to **Application**.
6. Type your connection string into the **Value** field, or click the **ellipsis (...)** button in the **Value** field to open the **Connection Properties** dialog box to build your connection string.

Edit connection strings stored in application settings

You can modify connection information that is saved in application settings by using the **Project Designer**.

To edit a connection string stored in application settings

1. In **Solution Explorer**, double-click the **My Project** icon (Visual Basic) or **Properties** icon (C#) to open the **Project Designer**.
2. Select the **Settings** tab.
3. Locate the connection you want to edit and select the text in the **Value** field.
4. Edit the connection string in the **Value** field, or click the **ellipsis (...)** button in the **Value** field to edit your connection with the **Connection Properties** dialog box.

Edit connection strings for datasets

You can modify connection information for each `TableAdapter` in a dataset.

To edit a connection string for a `TableAdapter` in a dataset

1. In **Solution Explorer**, double-click the dataset (.xsd file) that has the connection you want to edit.
2. Select the **TableAdapter** or query that has the connection you want to edit.
3. In the **Properties** window, expand the **Connection node**.
4. To quickly modify the connection string, edit the **ConnectionString** property, or click the down arrow on the **Connection** property and choose **New Connection**.

Security

Storing sensitive information (such as a password) within the connection string can affect the security of your application. Using Windows integrated security is a more secure way to control access to a database. For more information, see [Protecting connection information](#).

See also

- [Adding connections](#)

Connect to data in an Access database

8/5/2021 • 3 minutes to read • [Edit Online](#)

You can connect to an Access database (either an *.mdb* file or an *.accdb* file) by using Visual Studio. After you define the connection, the data appears in the **Data Sources** window. From there, you can drag tables or views onto your design surface.

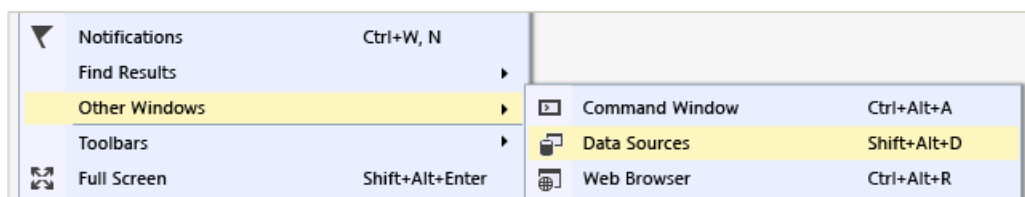
Prerequisites

To use these procedures, you need a Windows Forms or WPF project and either an Access database (*.accdb* file) or an Access 2000-2003 database (*.mdb* file). Follow the procedure that corresponds to your file type.

Create a dataset for an *.accdb* file

Connect to databases created with Microsoft 365, Access 2013, Access 2010, or Access 2007 by using the following procedure.

1. Open a Windows Forms or WPF application project in Visual Studio.
2. To open the **Data Sources** window, on the **View** menu, select **Other Windows > Data Sources**.



3. In the **Data Sources** window, click **Add New Data Source**.

The **Data Source Configuration Wizard** opens.

4. Select **Database** on the **Choose a Data Source Type** page, and then select **Next**.
5. Select **Dataset** on the **Choose a Database Model** page, and then select **Next**.
6. On the **Choose your Data Connection** page, select **New Connection** to configure a new data connection.

The **Add Connection** dialog box opens.

7. If **Data source** is not set to **Microsoft Access Database File**, select the **Change** button.

The **Change Data Source** dialog box opens. In the list of data sources, choose **Microsoft Access Database File**. In the **Data provider** drop-down, select **.NET Framework Data Provider for OLE DB**, and then choose **OK**.

8. Choose **Browse** next to **Database file name**, and then navigate to your *.accdb* file and choose **Open**.
9. Enter a user name and password (if necessary), and then choose **OK**.
10. Select **Next** on the **Choose your Data Connection** page.

You may get a dialog box telling you the data file is not in your current project. Select **Yes** or **No**.

11. Select **Next** on the **Save connection string to the Application Configuration file** page.

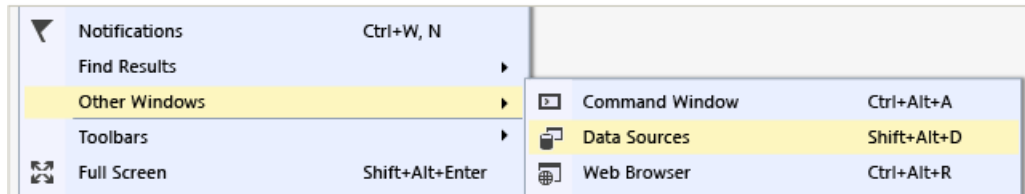
12. Expand the **Tables** node on the **Choose your Database Objects** page.
13. Select the tables or views you want to include in your dataset, and then select **Finish**.

The dataset is added to your project, and the tables and views appear in the **Data Sources** window.

Create a dataset for an .mdb file

Connect to databases created with Access 2000-2003 by using the following procedure.

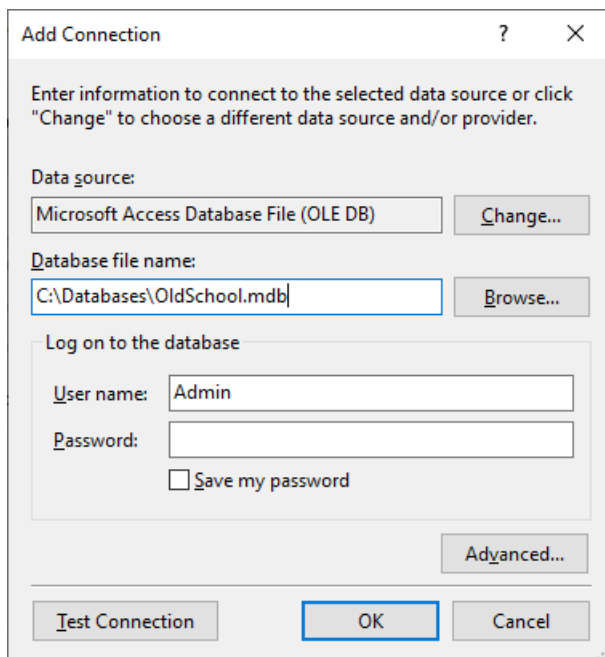
1. Open a Windows Forms or WPF application project in Visual Studio.
2. On the **View** menu, select **Other Windows > Data Sources**.



3. In the **Data Sources** window, click **Add New Data Source**.

The **Data Source Configuration Wizard** opens.

4. Select **Database** on the **Choose a Data Source Type** page, and then select **Next**.
5. Select **Dataset** on the **Choose a Database Model** page, and then select **Next**.
6. On the **Choose your Data Connection** page, select **New Connection** to configure a new data connection.
7. If the data source is not **Microsoft Access Database File (OLE DB)**, select **Change** to open the **Change Data Source** dialog box and select **Microsoft Access Database File**, and then select **OK**.
8. In the **Database file name**, specify the path and name of the **.mdb** file you want to connect to, and then select **OK**.



9. Select **Next** on the **Choose your Data Connection** page.
10. Select **Next** on the **Save connection string to the Application Configuration file** page.

11. Expand the **Tables** node on the **Choose your Database Objects** page.
12. Select whatever tables or views you want in your dataset, and then select **Finish**.

The dataset is added to your project, and the tables and views appear in the **Data Sources** window.

Next steps

The dataset that you just created is available in the **Data Sources** window. You can now perform any of the following tasks:

- Select items in the **Data Sources** window and drag them onto your form or design surface (see [Bind Windows Forms controls to data in Visual Studio](#) or [WPF data binding overview](#)).
- Open the data source in the **Dataset Designer** to add or edit the objects that make up the dataset.
- Add validation logic to the [ColumnChanging](#) or [RowChanging](#) event of the data tables in the dataset (see [Validate data in datasets](#)).

See also

- [Add connections](#)
- [WPF data binding overview](#)
- [Windows Forms data binding](#)

Add new data sources

8/5/2021 • 5 minutes to read • [Edit Online](#)

NOTE

The features described in this article apply to .NET Framework Windows Forms and WPF development. The features are not supported for .NET Core development, for both WPF and Windows Forms.

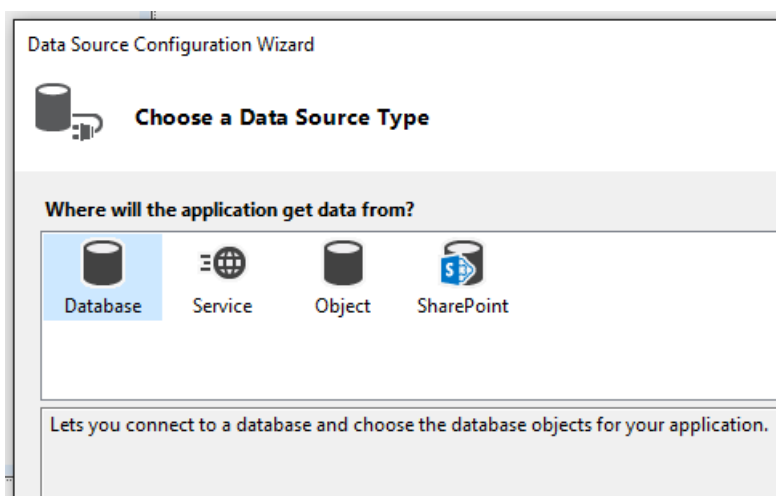
In the context of .NET data tools in Visual Studio, the term *data source* refers to .NET objects that connect to a data store and make the data available to a .NET application. The Visual Studio designers can consume the output of the data source to generate the boilerplate code that binds the data to forms when you drag and drop database objects from the **Data Sources** window. This kind of data source can be:

- A class in an Entity Framework model that is associated with some kind of database.
- A dataset that is associated with some kind of database.
- A class that represents a network service such as a Windows Communication Foundation (WCF) data service or a REST service.
- A class that represents a SharePoint service.
- A class or collection in your solution.

NOTE

If you're not using data-binding features, datasets, Entity Framework, LINQ to SQL, WCF, or SharePoint, the concept of a "data source" does not apply. Just connect directly to the database by using the `SqlCommand` objects and communicate directly with the database.

You create and edit data sources by using the **Data Source Configuration Wizard** in a Windows Forms or Windows Presentation Foundation application. For Entity Framework, first create your entity classes, and then start the wizard by selecting **Project > Add New Data Source** (described in more detail later in this article).



Data Sources window

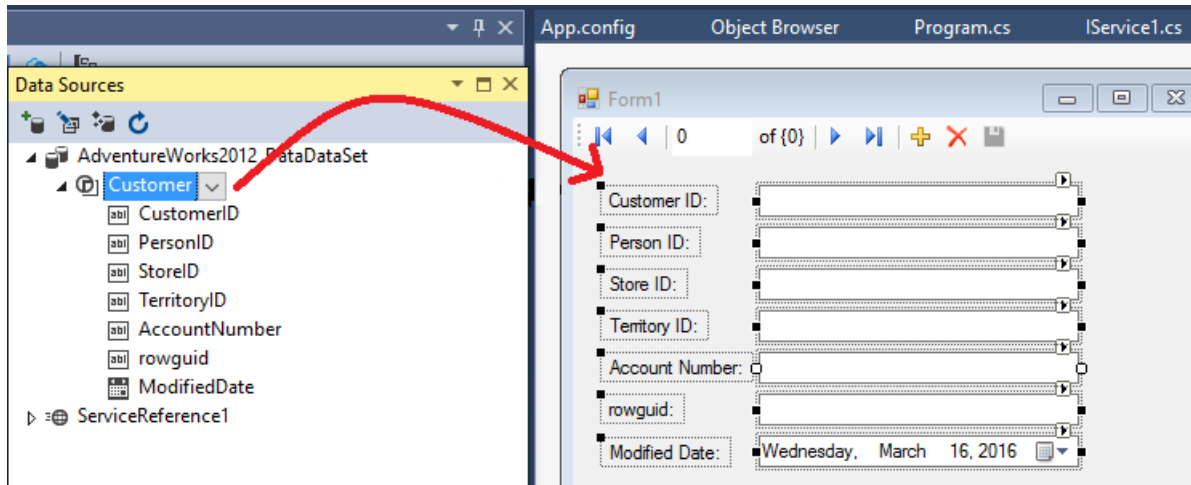
After you create a data source, it appears in the **Data Sources** tool window.

TIP

To open the **Data Sources** window, make sure your project is open, and then press **Shift+Alt+D** or choose **View > Other Windows > Data Sources**.

You can drag a data source from the **Data Sources** window onto a form design surface or control. This causes boilerplate code to be generated that displays the data from the data store.

The following illustration shows a dataset that has been dropped onto a Windows form. If you select **F5** on the application, the data from the underlying database appears in the form's controls.



Data source for a database or a database file

You can create a dataset or an Entity Framework model to use as a data source for a database or database file.

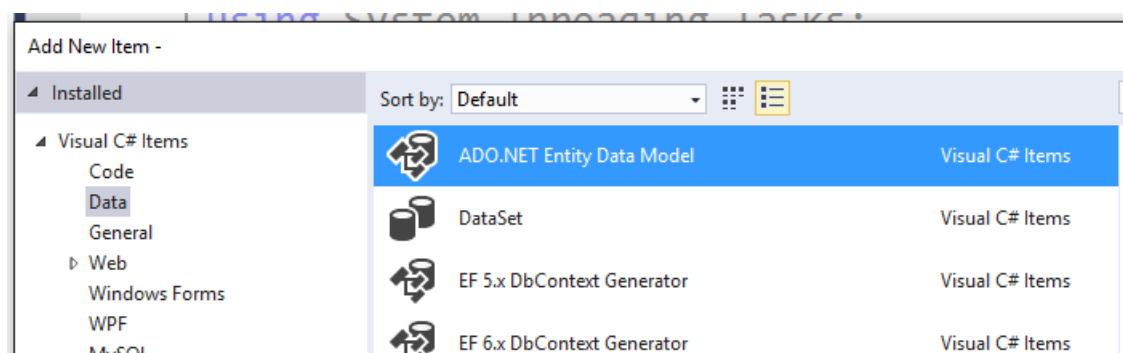
Dataset

To create a dataset as a data source, run the **Data Source Configuration Wizard** by selecting **Project > Add New Data Source**. Choose the **Database** data-source type, and follow the prompts to specify either a new or existing database connection, or a database file.

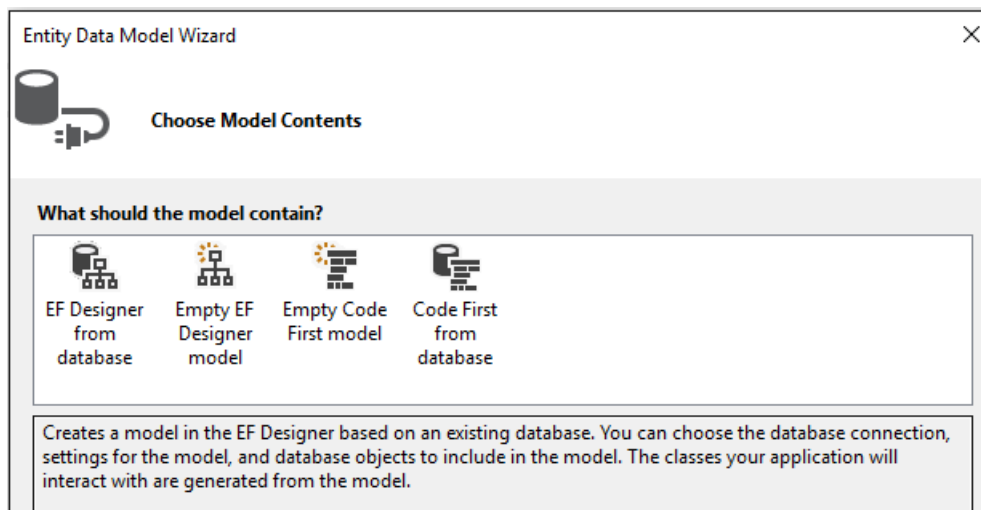
Entity classes

To create an Entity Framework model as a data source:

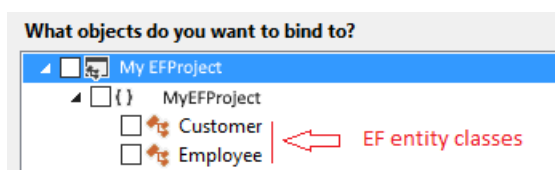
1. Run the **Entity Data Model Wizard** to create the entity classes. Select **Project > Add New Item > ADO.NET Entity Data Model**.



2. Choose the method you want to generate the model by.



3. Add the model as a data source. The generated classes appear in the **Data Source Configuration Wizard** when you choose the **Objects** category.



Data source for a service

To create a data source from a service, run the **Data Source Configuration Wizard** and choose the **Service** data-source type. This is just a shortcut to the **Add Service Reference** dialog box, which you can also access by right-clicking the project in **Solution Explorer** and selecting **Add service reference**.

When you create a data source from a service, Visual Studio adds a service reference to your project. Visual Studio also creates proxy objects that correspond to the objects that the service returns. For example, a service that returns a dataset is represented in your project as a dataset; a service that returns a specific type is represented in your project as the type returned.

You can create a data source from the following types of services:

- [WCF Data Services](#)
- [WCF services](#)
- Web services

NOTE

The items that appear in the **Data Sources** window are dependent on the data that the service returns. Some services might not provide enough information for the **Data Source Configuration Wizard** to create bindable objects. For example, if the service returns an untyped dataset, no items appear in the **Data Sources** window when you complete the wizard. This is because untyped datasets do not provide a schema, and therefore the wizard does not have enough information to create the data source.

Data source for an object

You can create a data source from any object that exposes one or more public properties by running the **Data Source Configuration Wizard** and then selecting the **Object** data-source type. All public properties of an object are displayed in the **Data Sources** window. If you are using Entity Framework and have generated a model, this is where you find the entity classes that are the data sources for your application.

On the **Select the Data Objects** page, expand the nodes in the tree view to locate the objects that you want to bind to. The tree view contains nodes for your project and for assemblies and other projects that are referenced by your project.

If you want to bind to an object in an assembly or project that does not appear in the tree view, click **Add Reference** and use the **Add Reference Dialog Box** to add a reference to the assembly or project. After you add the reference, the assembly or project is added to the tree view.

NOTE

You may need to build the project that contains your objects before the objects appear in the tree view.

NOTE

To support drag-and-drop data binding, objects that implement the [ITypeedList](#) or [IListSource](#) interface must have a default constructor. Otherwise, Visual Studio cannot instantiate the data-source object, and it displays an error when you drag the item to the design surface.

Data source for a SharePoint list

You can create a data source from a SharePoint list by running the **Data Source Configuration Wizard** and selecting the **SharePoint** data-source type. SharePoint exposes data through WCF Data Services, so creating a SharePoint data source is the same as creating a data source from a service. Selecting the **SharePoint** item in the **Data Source Configuration Wizard** opens the **Add Service Reference** dialog box, where you connect to the SharePoint data service by pointing to the SharePoint server. This requires the SharePoint SDK.

See also

- [Visual Studio data tools for .NET](#)

LINQ to SQL tools in Visual Studio

8/5/2021 • 5 minutes to read • [Edit Online](#)

LINQ to SQL was the first object-relational mapping technology released by Microsoft. It works well in basic scenarios and continues to be supported in Visual Studio, but it's no longer under active development. Use LINQ to SQL when maintaining a legacy application that's already using it, or in simple applications that use SQL Server and do not require multi-table mapping. In general, new applications should use the Entity Framework when an object-relational mapper layer is required.

Install the LINQ to SQL tools

In Visual Studio, you create LINQ to SQL classes that represent SQL tables by using the **Object Relational Designer (O/R Designer)**. The O/R designer is the UI for editing .dbml files. Editing .dbml files with a designer surface requires the LINQ to SQL tools which are not installed by default as part of any of the workloads of Visual Studio.

To install the LINQ to SQL tools, start the Visual Studio installer, choose **Modify**, then select the **Individual Components** tab, and then select **LINQ to SQL tools** under the **Code Tools** category.

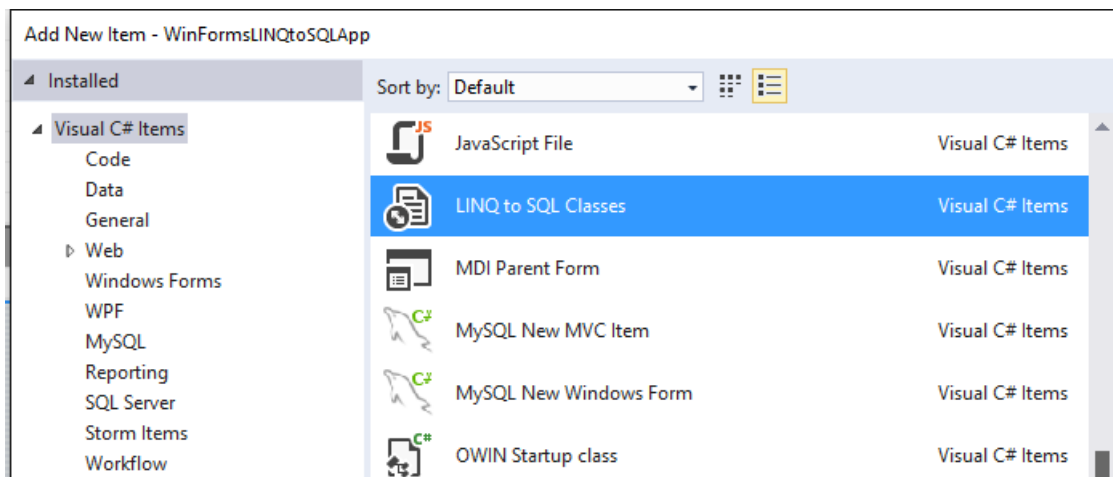
What is the O/R Designer

The **O/R Designer** has two distinct areas on its design surface: the entities pane on the left, and the methods pane on the right. The entities pane is the main design surface that displays the entity classes, associations, and inheritance hierarchies. The methods pane is the design surface that displays the [DataContext](#) methods that are mapped to stored procedures and functions.

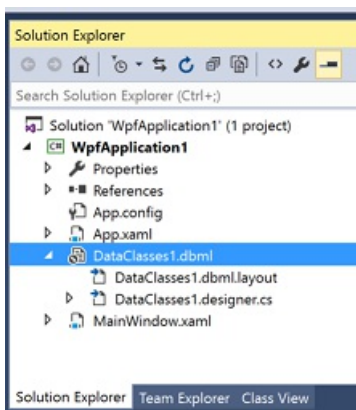
The **O/R Designer** provides a visual design surface for creating [LINQ to SQL](#) entity classes and associations (relationships) that are based on objects in a database. In other words, the **O/R Designer** creates an object model in an application that maps to objects in a database. It also generates a strongly-typed [DataContext](#) that sends and receives data between the entity classes and the database. The **O/R Designer** also provides functionality to map stored procedures and functions to [DataContext](#) methods for returning data and populating entity classes. Finally, the **O/R Designer** provides the ability to design inheritance relationships between entity classes.

Open the O/R designer

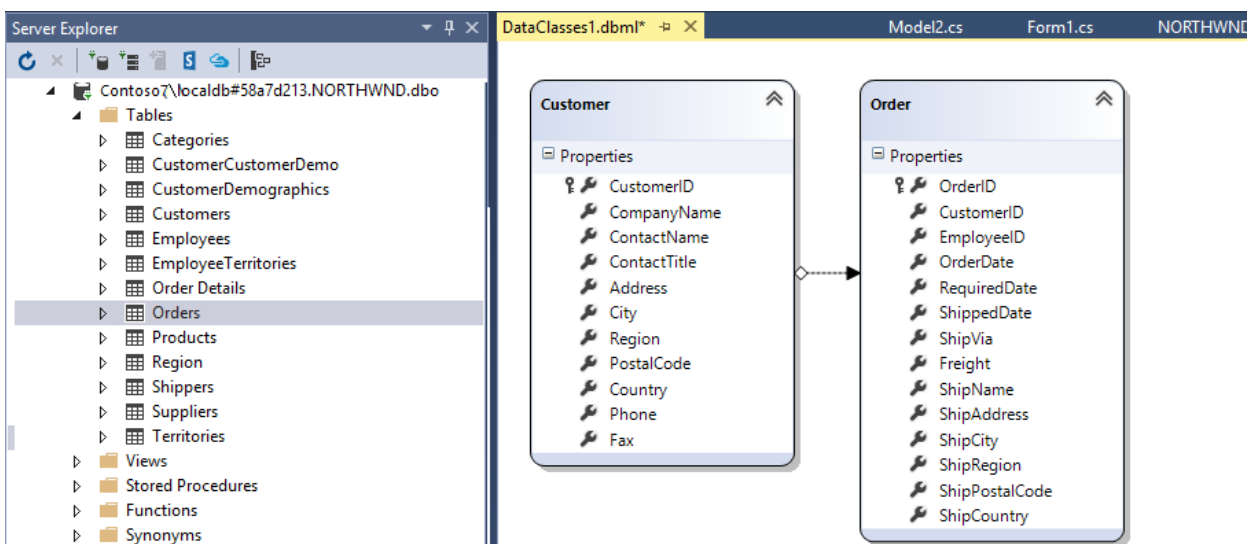
To add a LINQ to SQL entity model to your project, choose **Project > Add New Item**, and then select **LINQ to SQL Classes** from the list of project items:



Visual Studio creates a *.dbml* file and adds it to your solution. This is the XML mapping file and its related code files.



When you select the *.dbml* file, Visual Studio shows the **O/R Designer** surface that enables you to visually create the model. The following illustration shows the designer after the Northwind **Customers** and **Orders** tables have been dragged from **Server Explorer**. Note the relationship between the tables.



IMPORTANT

The **O/R Designer** is a simple object relational mapper because it supports only 1:1 mapping relationships. In other words, an entity class can have only a 1:1 mapping relationship with a database table or view. Complex mapping, such as mapping an entity class to a joined table, is not supported; use the Entity Framework for complex mapping. Additionally, the designer is a one-way code generator. This means that only changes that you make to the designer surface are reflected in the code file. Manual changes to the code file are not reflected in the **O/R Designer**. Any changes that you make manually in the code file are overwritten when the designer is saved and code is regenerated. For information about how to add user code and extend the classes generated by the **O/R Designer**, see [How to: Extend code generated by the O/R Designer](#).

Create and configure the DataContext

After you add a **LINQ to SQL Classes** item to a project and open the **O/R Designer**, the empty design surface represents an empty **DataContext** ready to be configured. the **DataContext** is configured with connection information provided by the first item that is dragged onto the design surface. Therefore, the **DataContext** is configured by using connection information from the first item dropped onto the design surface. For more information about the **DataContext** class see, [DataContext methods \(O/R Designer\)](#).

Create entity classes that map to database tables and views

You can create entity classes mapped to tables and views by dragging database tables and views from **Server Explorer** or **Database Explorer** onto the **O/R Designer**. As indicated in the previous section, the **DataContext** is configured with connection information provided by the first item that is dragged onto the design surface. If a subsequent item that uses a different connection is added to the **O/R Designer**, you can change the connection for the **DataContext**. For more information, see [How to: Create LINQ to SQL classes mapped to tables and views \(O/R Designer\)](#).

Create DataContext methods that call stored procedures and functions

You can create **DataContext** methods that call (are mapped to) stored procedures and functions by dragging them from **Server Explorer** or **Database Explorer** onto the **O/R Designer**. Stored procedures and functions are added to the **O/R Designer** as methods of the **DataContext**.

NOTE

When you drag stored procedures and functions from **Server Explorer** or **Database Explorer** onto the **O/R Designer**, the return type of the generated **DataContext** method differs depending on where you drop the item. For more information, see [DataContext methods \(O/R Designer\)](#).

Configure a DataContext to use stored procedures to save data between entity classes and a database

As stated earlier, you can create **DataContext** methods that call stored procedures and functions. Additionally, you can also assign stored procedures that are used for the default LINQ to SQL run-time behavior, which performs inserts, updates, and deletes. For more information, see [How to: Assign stored procedures to perform updates, inserts, and deletes \(O/R Designer\)](#).

Inheritance and the O/R designer

Like other objects, LINQ to SQL classes can use inheritance and be derived from other classes. In a database,

inheritance relationships are created in several ways. The **O/R Designer** supports the concept of single-table inheritance as it is often implemented in relational systems. For more information, see [How to: Configure inheritance by using the O/R Designer](#).

LINQ to SQL queries

The entity classes created by the **O/R Designer** are designed for use with [Language-Integrated query \(LINQ\)](#). For more information, see [How to: Query for information](#).

Separate the generated DataContext and entity class code into different namespaces

The **O/R Designer** provides the **Context Namespace** and **Entity Namespace** properties on the [DataContext](#). These properties determine what namespace the [DataContext](#) and entity class code is generated into. By default, these properties are empty and the [DataContext](#) and entity classes are generated into the application's namespace. To generate the code into a namespace other than the application's namespace, enter a value into the **Context Namespace** and/or **Entity Namespace** properties.

Reference content

- [System.Linq](#)
- [System.Data.Linq](#)

See also

- [LINQ to SQL \(.NET Framework\)](#)
- [Frequently asked questions \(.NET Framework\)](#)

How to: Change the return type of a DataContext method (O/R Designer)

8/5/2021 • 2 minutes to read • [Edit Online](#)

The return type of a [DataContext](#) method (created based on a stored procedure or function) differs depending on where you drop the stored procedure or function in the **O/R Designer**. If you drop an item directly onto an existing entity class, a [DataContext](#) method that has the return type of the entity class is created (if the schema of the data returned by the stored procedure or function matches the shape of the entity class). If you drop an item onto an empty area of the **O/R Designer**, a [DataContext](#) method that returns an automatically generated type is created. You can change the return type of a [DataContext](#) method after you add it to the methods pane. To inspect or change the return type of a [DataContext](#) method, select it and click the **Return Type** property in the **Properties** window.

NOTE

You cannot revert [DataContext](#) methods that have a return type set to an entity class to return the auto-generated type by using the **Properties** window. To revert a [DataContext](#) method to return an auto-generated type, you must drag the original database object onto the **O/R Designer** again.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To change the return type of a DataContext method from the auto-generated type to an entity class

1. Select the [DataContext](#) method in the methods pane.
2. Select **Return Type** in the **Properties** window and then select an available entity class in the **Return Type** list. If the desired entity class is not in the list, add it to or create it in the **O/R Designer** to add it to the list.
3. Save the *.dbml* file.

To change the return type of a DataContext method from an entity class back to the auto-generated type

1. Select the [DataContext](#) method in the **Methods** pane and delete it.
2. Drag the database object from **Server Explorer** or **Database Explorer** onto an empty area of the **O/R Designer**.
3. Save the *.dbml* file.

See also

- [LINQ to SQL tools in Visual Studio](#)

- [LINQ to SQL](#)
- [DataContext methods \(O/R Designer\)](#)
- [How to: Create DataContext methods mapped to stored procedures and functions \(O/R Designer\)](#)

How to: Create DataContext methods mapped to stored procedures and functions (O/R Designer)

8/5/2021 • 2 minutes to read • [Edit Online](#)

You can add stored procedures and functions to the **O/R Designer** as [DataContext](#) methods. Calling the method and passing in the required parameters runs the stored procedure or function on the database and returns the data in the return type of the [DataContext](#) method. For detailed information about [DataContext](#) methods, see [DataContext methods \(O/R Designer\)](#).

NOTE

You can also use stored procedures to override the default LINQ to SQL run-time behavior that performs Inserts, Updates, and Deletes when changes are saved from entity classes to a database. For more information, see [How to: Assign stored procedures to perform updates, inserts, and deletes \(O/R Designer\)](#).

Create DataContext methods

You can create [DataContext](#) methods by dragging stored procedures or functions from **Server Explorer** or **Database Explorer** onto the **O/R Designer**.

NOTE

The return type of the generated [DataContext](#) method differs depending on where you drop the stored procedure or function on the **O/R Designer**. Dropping items directly onto an existing entity class creates a [DataContext](#) method with the return type of the entity class. Dropping items onto an empty area of the **O/R Designer** creates a [DataContext](#) method that returns an automatically generated type. You can change the return type of a [DataContext](#) method after adding it to the **Methods** pane. To inspect or change the return type of a [DataContext](#) method, select it and inspect the **Return Type** property in the **Properties** window. For more information, see [How to: Change the return type of a DataContext method \(O/R Designer\)](#).

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To create DataContext methods that return automatically generated types

1. In **Server Explorer** or **Database Explorer**, expand the **Stored Procedures** node of the database with which you are working.
2. Locate the desired stored procedure and drag it onto an empty area of the **O/R Designer**.

The [DataContext](#) method is created with an automatically generated return type and appears in the **Methods** pane.

To create DataContext methods that have the return type of an entity class

1. In **Server Explorer** or **Database Explorer**, expand the **Stored Procedures** node of the database with which you are working.

2. Locate the desired stored procedure and drag it onto an existing entity class in the **O/R Designer**.

The [DataContext](#) method is created with the return type of the selected entity class and appears in the **Methods** pane.

NOTE

For information about changing the return type of existing [DataContext](#) methods, see [How to: Change the return type of a DataContext method \(O/R Designer\)](#).

See also

- [LINQ to SQL tools in Visual Studio](#)
- [DataContext methods \(O/R Designer\)](#)
- [Walkthrough: Creating LINQ to SQL classes](#)
- [LINQ to SQL](#)
- [Introduction to LINQ in Visual Basic](#)
- [LINQ in C#](#)

How to: Configure inheritance by using the O/R Designer

8/5/2021 • 3 minutes to read • [Edit Online](#)

The **Object Relational Designer (O/R Designer)** supports the concept of single-table inheritance as it is often implemented in relational systems. In single-table inheritance, there is a single database table that contains fields for both parent information and child information. With relational data, a discriminator column contains the value that determines which class any record belongs to.

For example, consider a `Persons` table that contains everyone employed by a company. Some people are employees and some people are managers. The `Persons` table contains a column named `EmployeeType` that has a value of 1 for managers and a value of 2 for employees; this is the discriminator column. In this scenario, you can create a subclass of employees and populate the class with only records that have an `EmployeeType` value of 2. You can also remove columns that do not apply from each of the classes.

Creating an object model that uses inheritance (and corresponds to relational data) can be somewhat confusing. The following procedure outlines the steps required for configuring inheritance with the **O/R Designer**. Following generic steps without referring to an existing table and columns might be difficult, so a walkthrough that uses data is provided. For detailed step-by-step directions for configuring inheritance by using the **O/R Designer**, see [Walkthrough: Creating LINQ to SQL classes by using single-table inheritance \(O/R Designer\)](#).

To create inherited data classes

1. Open the **O/R Designer** by adding a **LINQ to SQL Classes** item to an existing Visual Basic or C# project.
2. Drag the table you want to use as the base class onto the **O/R Designer**.
3. Drag a second copy of the table onto the **O/R Designer** and rename it. This is the derived class, or subclass.
4. Click **Inheritance** in the **Object Relational Designer** tab of the **Toolbox**, and then click the subclass (the table you renamed) and connect it to the base class.

NOTE

Click the **Inheritance** item in the **Toolbox** and release the mouse button, click the second copy of the class you created in step 3, and then click the first class you created in step 2. The arrow on the inheritance line points to the first class.

5. In each class, delete any object properties that you do not want to appear and that are not used for associations. You receive an error if you attempt to delete object properties used for associations: [The property <property name> cannot be deleted because it is participating in the association <association name>](#).

NOTE

Because a derived class inherits the properties defined in its base class, the same columns cannot be defined in each class. (Columns are implemented as properties.) You can enable the creation of columns in the derived class by setting the inheritance modifier on the property in the base class. For more information, see [Inheritance basics \(Visual Basic\)](#).

6. Select the inheritance line in the **O/R Designer**.
7. In the **Properties** window, set the **Discriminator Property** to the column name that distinguishes the records in your classes.
8. Set the **Derived Class Discriminator Value** property to the value in the database that designates the record as the inherited type. (This is the value that is stored in the discriminator column and is used to designate the inherited class.)
9. Set the **Base Class Discriminator Value** property to the value that designates the record as a base type. (This is the value that is stored in the discriminator column and is used to designate the base class.)
10. Optionally, you can also set the **Inheritance Default** property to designate a type in an inheritance hierarchy that is used when loading rows that do not match any defined inheritance code. In other words, if a record has a value in its discriminator column that does not match the value in either the **Derived Class Discriminator Value** or **Base Class Discriminator Value** properties, the record loads into the type designated as the **Inheritance Default**.

See also

- [LINQ to SQL tools in Visual Studio](#)
- [Walkthrough: Creating LINQ to SQL classes \(O-R Designer\)](#)
- [Accessing data in Visual Studio](#)
- [LINQ to SQL](#)
- [Walkthrough: Creating LINQ to SQL classes by using single-table inheritance \(O/R Designer\)](#)
- [Inheritance basics \(Visual Basic\)](#)
- [Inheritance](#)

How to: Create LINQ to SQL classes mapped to tables and views (O/R Designer)

8/5/2021 • 3 minutes to read • [Edit Online](#)

LINQ to SQL classes that are mapped to database tables and views are called *entity classes*. The entity class maps to a record, whereas the individual properties of an entity class map to the individual columns that make up a record. Create entity classes that are based on database tables or views by dragging tables or views from **Server Explorer** or **Database Explorer** onto the [LINQ to SQL tools in Visual Studio](#). The **O/R Designer** generates the classes and applies the specific LINQ to SQL attributes to enable LINQ to SQL functionality (the data communication and editing capabilities of the [DataContext](#)). For detailed information about LINQ to SQL classes, see [The LINQ to SQL object model](#).

NOTE

The **O/R Designer** is a simple object relational mapper because it supports only 1:1 mapping relationships. In other words, an entity class can have only a 1:1 mapping relationship with a database table or view. Complex mapping, such as mapping an entity class to multiple tables, is not supported. However, you can map an entity class to a view that joins multiple related tables.

Create LINQ to SQL classes that are mapped to database tables or views

Dragging tables or views from **Server Explorer** or **Database Explorer** onto the **O/R Designer** creates entity classes in addition to the [DataContext](#) methods that are used for performing updates.

By default, the LINQ to SQL runtime creates logic to save changes from an updatable entity class back to the database. This logic is based on the schema of the table (the column definitions and primary key information). If you do not want this behavior, you can configure an entity class to use stored procedures to perform inserts, updates, and deletes instead of using the default LINQ to SQL run-time behavior. For more information, see [How to: Assign stored procedures to perform updates, inserts, and deletes \(O/R Designer\)](#).

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To create LINQ to SQL classes that are mapped to database tables or views

1. In **Server** or **Database Explorer**, expand **Tables** or **Views** and locate the database table or view that you want to use in your application.
2. Drag the table or view onto the **O/R Designer**.

An entity class is created and appears on the design surface. The entity class has properties that map to the columns in the selected table or view.

Create an object data source and display the data on a form

After you create entity classes by using the **O/R Designer**, you can create an object data source and populate

the [Data Sources window](#) with the entity classes.

To create an object data source based on LINQ to SQL entity classes

1. On the **Build** menu, click **Build Solution** to build your project.
2. To open the **Data Sources** window, on the **Data** menu, click **Show Data Sources**.
3. In the **Data Sources** window, click **Add New Data Source**.
4. Click **Object** on the **Choose a Data Source Type** page and then click **Next**.
5. Expand the nodes and locate and select your class.

NOTE

If the **Customer** class is not available, cancel out of the wizard, build the project, and run the wizard again.

6. Click **Finish** to create the data source and add the **Customer** entity class to the **Data Sources** window.
7. Drag items from the **Data Sources** window onto a form.

See also

- [LINQ to SQL tools in Visual Studio](#)
- [Walkthrough: Creating LINQ to SQL classes \(O/R Designer\)](#)
- [DataContext methods \(O/R Designer\)](#)
- [How to: Create DataContext methods mapped to stored procedures and functions \(O/R Designer\)](#)
- [The LINQ to SQL object model](#)
- [Walkthrough: Customizing the insert, update, and delete behavior of entity classes](#)
- [How to: Create an association \(relationship\) between LINQ to SQL classes \(O/R Designer\)](#)

How to: Extend code generated by the O/R Designer

8/5/2021 • 2 minutes to read • [Edit Online](#)

Code generated by the **O/R Designer** is regenerated when changes are made to the entity classes and other objects on the designer surface. Because of this code regeneration, any code that you add to the generated code is typically overwritten when the designer regenerates code. The **O/R Designer** provides the ability to generate partial class files in which you can add code that is not overwritten. One example of adding your own code to the code generated by the **O/R Designer** is adding data validation to LINQ to SQL (entity) classes. For more information, see [How to: Add validation to entity classes](#).

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

Add code to an entity class

To create a partial class and add code to an entity class

1. Open or create a new LINQ to SQL Classes file (.dbml file) in the **O/R Designer**. (Double-click the .dbml file in **Solution Explorer** or **Database Explorer**.)
2. In the **O/R Designer**, right-click the class for which you want to add validation and then click **View Code**.

The Code Editor opens with a partial class for the selected entity class.

3. Add your code in the partial class declaration for the entity class.

Add code to a DataContext

To create a partial class and add code to a DataContext

1. Open or create a new LINQ to SQL Classes file (.dbml file) in the **O/R Designer**. (Double-click the .dbml file in **Solution Explorer** or **Database Explorer**.)
2. In the **O/R Designer**, right-click an empty area on the designer and then click **View Code**.

The Code Editor opens with a partial class for the DataContext.

3. Add your code in the partial class declaration for the DataContext.

See also

- [LINQ to SQL tools in Visual Studio](#)
- [Walkthrough: Creating LINQ to SQL classes \(O-R Designer\)](#)
- [LINQ to SQL](#)

How to: Create an association between LINQ to SQL classes (O/R Designer)

8/5/2021 • 2 minutes to read • [Edit Online](#)

Associations between entity classes in LINQ to SQL are analogous to relationships between tables in a database. You can create associations between entity classes by using the **Association Editor** dialog box.

You must select a parent class and child class when you use the **Association Editor** dialog box to create an association. The parent class is the entity class that contains the primary key; the child class is the entity class that contains the foreign-key. For example, if entity classes were created that map to the `Northwind Customers` and `Orders` tables, the `Customer` class would be the parent class and the `Order` class would be the child class.

NOTE

When you drag tables from **Server Explorer** or **Database Explorer** onto the **Object Relational Designer (O/R Designer)**, associations are automatically created based on the existing foreign-key relationships in the database.

Association properties

After you create an association, when you select the association in the **O/R Designer**, there are some configurable properties in the **Properties** window. (The association is the line between the related classes.) The following table provides descriptions for the properties of an association.

| PROPERTY | DESCRIPTION |
|---------------------------------|--|
| Cardinality | Controls whether the association is one-to-many or one-to-one. |
| Child Property | Specifies whether to create a property on the parent that is a collection or reference to the child records on the foreign-key side of the association. For example, in the association between <code>Customer</code> and <code>Order</code> , if the Child Property is set to True , a property named <code>Orders</code> is created on the parent class. |
| Parent Property | The property on the child class that references the associated parent class. For example, in the association between <code>Customer</code> and <code>Order</code> , a property named <code>Customer</code> that references the associated customer for an order is created on the <code>Order</code> class. |
| Participating Properties | Displays the association properties and provides an ellipsis button (...) that re-opens the Association Editor dialog box. |
| Unique | Specifies whether the foreign target columns have a uniqueness constraint. |

To create an association between entity classes

1. Right-click the entity class that represents the parent class in the association, point to **Add**, and then click **Association**.
2. Verify that the correct **Parent Class** is selected in the **Association Editor** dialog box.
3. Select the **Child Class** in the combo box.
4. Select the **Association Properties** that relate the classes. Typically, this maps to the foreign-key relationship defined in the database. For example, in the `Customers` and `Orders` association, the **Association Properties** are the `CustomerID` for each class.
5. Click **OK** to create the association.

See also

- [LINQ to SQL tools in Visual Studio](#)
- [Walkthrough: Creating LINQ to SQL classes](#)
- [LINQ to SQL](#)
- [DataContext methods \(O/R Designer\)](#)
- [How to: Represent primary keys](#)

How to: Add validation to entity classes

8/5/2021 • 4 minutes to read • [Edit Online](#)

Validating entity classes is the process of confirming that the values entered into data objects comply with the constraints in an object's schema, and also to the rules established for the application. Validating data before you send updates to the underlying database is a good practice that reduces errors. It also reduces the potential number of round trips between an application and the database.

The [LINQ to SQL tools in Visual Studio](#) provides partial methods that enable users to extend the designer-generated code that runs during Inserts, Updates, and Deletes of complete entities, and also during and after individual column changes.

NOTE

This topic provides the basic steps for adding validation to entity classes by using the **O/R Designer**. Because it might be difficult to follow these generic steps without referring to a specific entity class, a walkthrough that uses actual data is provided.

Add validation for changes to the value in a specific column

This procedure shows how to validate data when the value in a column changes. Because the validation is performed inside the class definition (instead of in the user interface), an exception is thrown if the value causes validation to fail. Implement error handling for the code in your application that attempts to change the column values.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To validate data during a column's value change

1. Open or create a new LINQ to SQL Classes file (.dbml file) in the **O/R Designer**. (Double-click the .dbml file in **Solution Explorer**.)
2. In the **O/R Designer**, right-click the class for which you want to add validation and then click **View Code**.

The Code Editor opens with a partial class for the selected entity class.

3. Place the cursor in the partial class.
4. For Visual Basic projects:
 - a. Expand the **Method Name** list.
 - b. Locate the **OnCOLUMNNAMEChanging** method for the column you want to add validation to.
 - c. An `OnCOLUMNNAMEChanging` method is added to the partial class.
 - d. Add the following code to first verify that a value has been entered and then to ensure that the value entered for the column is acceptable for your application. The `value` argument contains the

proposed value, so add logic to confirm that it is a valid value:

```
If value.HasValue Then
    ' Add code to ensure that the value is acceptable.
    ' If value < 1 Then
    '     Throw New Exception("Invalid data!")
    ' End If
End If
```

For C# projects:

Because C# projects do not automatically generate event handlers, you can use IntelliSense to create the column-changing partial methods. Type `partial` and then a space to access the list of available partial methods. Click the column-changing method for the column you want to add validation for. The following code resembles the code that is generated when you select a column-changing partial method:

```
partial void OnCOLUMNNAMEChanging(COLUMNDATATYPE value)
{
    throw new System.NotImplementedException();
}
```

Add Validation for Updates to an Entity Class

In addition to checking values during changes, you can also validate data when an attempt is made to update a complete entity class. Validation during an attempted update enables you to compare values in multiple columns if business rules require this. The following procedure shows how to validate when an attempt is made to update a complete entity class.

NOTE

Validation code for updates to complete entity classes is executed in the partial [DataContext](#) class (instead of in the partial class of a specific entity class).

To validate data during an update to an entity class

1. Open or create a new LINQ to SQL Classes file (.dbml file) in the **O/R Designer**. (Double-click the .dbml file in **Solution Explorer**.)
2. Right-click an empty area on the **O/R Designer** and click **View Code**.

The Code Editor opens with a partial class for the `DataContext`.

3. Place the cursor in the partial class for the `DataContext`.
4. For Visual Basic projects:
 - a. Expand the **Method Name** list.
 - b. Click **UpdateENTITYCLASSNAME**.
 - c. An `UpdateENTITYCLASSNAME` method is added to the partial class.
 - d. Access individual column values by using the `instance` argument, as shown in the following code:

```
If (instance.COLUMNNAME = x) And (instance.COLUMNNAME = y) Then
    Dim ErrorMessage As String = "Invalid data!"
    Throw New Exception(ErrorMessage)
End If
```

For C# projects:

Because C# projects do not automatically generate event handlers, you can use IntelliSense to create the partial `UpdateCLASSNAME` method. Type `partial` and then a space to access the list of available partial methods. Click the update method for the class on which you want to add validation. The following code resembles the code that is generated when you select an `UpdateCLASSNAME` partial method:

```
partial void UpdateCLASSNAME(CLASSNAME instance)
{
    if ((instance.COLUMNNAME == x) && (instance.COLUMNNAME = y))
    {
        string ErrorMessage = "Invalid data!";
        throw new System.Exception(ErrorMessage);
    }
}
```

See also

- [LINQ to SQL tools in Visual Studio](#)
- [Validating data](#)
- [LINQ to SQL \(.NET Framework\)](#)

Walkthrough: Customize the insert, update, and delete behavior of entity classes

8/5/2021 • 9 minutes to read • [Edit Online](#)

The [LINQ to SQL tools in Visual Studio](#) provides a visual design surface for creating and editing LINQ to SQL classes (entity classes) that are based on objects in a database. By using [LINQ to SQL](#), you can use LINQ technology to access SQL databases. For more information, see [LINQ \(Language-Integrated query\)](#).

By default, the logic to perform updates is provided by the LINQ to SQL runtime. The runtime creates default `Insert`, `Update`, and `Delete` statements based on the schema of the table (the column definitions and primary key information). When you do not want to use the default behavior, you can configure the update behavior and designate specific stored procedures for performing the necessary inserts, updates, and deletes required to work with the data in the database. You can also do this when the default behavior is not generated, for example, when your entity classes map to views. Additionally, you can override the default update behavior when the database requires table access through stored procedures. For more information, see [Customizing operations by using stored procedures](#).

NOTE

This walkthrough requires the availability of the `InsertCustomer`, `UpdateCustomer`, and `DeleteCustomer` stored procedures for the Northwind database.

This walkthrough provides the steps that you must follow to override the default LINQ to SQL run-time behavior for saving data back to a database by using stored procedures.

During this walkthrough, you learn how to perform the following tasks:

- Create a new Windows Forms application and add a LINQ to SQL file to it.
- Create an entity class that is mapped to the Northwind `Customers` table.
- Create an object data source that references the LINQ to SQL `Customer` class.
- Create a Windows Form that contains a [DataGridView](#) that is bound to the `Customer` class.
- Implement save functionality for the form.
- Create [DataContext](#) methods by adding stored procedures to the **O/R Designer**.
- Configure the `Customer` class to use stored procedures to perform inserts, updates, and deletes.

Prerequisites

This walkthrough uses SQL Server Express LocalDB and the Northwind sample database.

1. If you don't have SQL Server Express LocalDB, install it either from the [SQL Server Express download page](#), or through the **Visual Studio Installer**. In the **Visual Studio Installer**, you can install SQL Server Express LocalDB as part of the **Data storage and processing** workload, or as an individual component.
2. Install the Northwind sample database by following these steps:
 - a. In Visual Studio, open the **SQL Server Object Explorer** window. (**SQL Server Object Explorer**

installs as part of the **Data storage and processing** workload in the **Visual Studio Installer**.)
Expand the **SQL Server** node. Right-click on your LocalDB instance and select **New Query**.

A query editor window opens.

- b. Copy the [Northwind Transact-SQL script](#) to your clipboard. This T-SQL script creates the Northwind database from scratch and populates it with data.
- c. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes running and the Northwind database is created.

Creating an application and adding LINQ to SQL classes

Because you are working with LINQ to SQL classes and displaying the data on a Windows Form, create a new Windows Forms application and add a LINQ to SQL Classes file.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To create a new Windows Forms Application project that contains LINQ to SQL classes

1. In Visual Studio, on the **File** menu, select **New > Project**.
2. Expand either **Visual C#** or **Visual Basic** in the left-hand pane, then select **Windows Desktop**.
3. In the middle pane, select the **Windows Forms App** project type.
4. Name the project **UpdatingWithSProcsWalkthrough**, and then choose **OK**.

The **UpdatingWithSProcsWalkthrough** project is created, and added to **Solution Explorer**.

5. On the **Project** menu, click **Add New Item**.
6. Click the **LINQ to SQL Classes** template and type **Northwind.dbml** in the **Name** box.
7. Click **Add**.

An empty LINQ to SQL Classes file (**Northwind.dbml**) is added to the project, and the **O/R Designer** opens.

Create the Customer entity class and object data source

Create LINQ to SQL classes that are mapped to database tables by dragging tables from **Server Explorer** or **Database Explorer** onto the **O/R Designer**. The result is LINQ to SQL entity classes that map to the tables in the database. After you create entity classes, they can be used as object data sources just like other classes that have public properties.

To create a Customer entity class and configure a data source with it

1. In **Server Explorer** or **Database Explorer**, locate the **Customer** table in the SQL Server version of the Northwind sample database.
2. Drag the **Customers** node from **Server Explorer** or **Database Explorer** onto the **O/R Designer* surface.

An entity class named **Customer** is created. It has properties that correspond to the columns in the Customers table. The entity class is named **Customer** (not **Customers**) because it represents a single

customer from the Customers table.

NOTE

This renaming behavior is called *pluralization*. It can be turned on or off in the [Options dialog box](#). For more information, see [How to: Turn pluralization on and off \(O/R Designer\)](#).

3. On the **Build** menu, click **Build UpdatingwithSProcsWalkthrough** to build the project.
4. To open the **Data Sources** window, on the **Data** menu, click **Show Data Sources**.
5. In the **Data Sources** window, click **Add New Data Source**.
6. Click **Object** on the **Choose a Data Source Type** page and then click **Next**.
7. Expand the **UpdatingwithSProcsWalkthrough** node and locate and select the **Customer** class.

NOTE

If the **Customer** class is not available, cancel out of the wizard, build the project, and run the wizard again.

8. Click **Finish** to create the data source and add the **Customer** entity class to the **Data Sources** window.

Create a DataGridView to display the customer data on a Windows Form

Create controls that are bound to entity classes by dragging LINQ to SQL data source items from the **Data Sources** window onto a Windows Form.

To add controls that are bound to the entity classes

1. Open **Form1** in Design view.
2. From the **Data Sources** window, drag the **Customer** node onto **Form1**.

NOTE

To display the **Data Sources** window, click **Show Data Sources** on the **Data** menu.

3. Open **Form1** in the Code Editor.
4. Add the following code to the form, global to the form, outside any specific method, but inside the **Form1** class:

```
Private NorthwindDataContext1 As New NorthwindDataContext
```

```
private NorthwindDataContext northwindDataContext1  
= new NorthwindDataContext();
```

5. Create an event handler for the **Form_Load** event and add the following code to the handler:

```
CustomerBindingSource.DataSource = NorthwindDataContext1.Customers
```

```
customerBindingSource.DataSource  
    = northwindDataContext1.Customers;
```

Implement save functionality

By default, the save button is not enabled and save functionality is not implemented. Also, code is not automatically added to save changed data to the database when data-bound controls are created for object data sources. This section explains how to enable the save button and implement save functionality for LINQ to SQL objects.

To implement save functionality

1. Open **Form1** in Design view.
2. Select the save button on the **CustomerBindingNavigator** (the button with the floppy disk icon).
3. In the **Properties** window, set the **Enabled** property to **True**.
4. Double-click the save button to create an event handler and switch to the Code Editor.
5. Add the following code into the save button event handler:

```
NorthwindDataContext1.SubmitChanges()
```

```
northwindDataContext1.SubmitChanges();
```

Override the default behavior for performing updates (inserts, updates, and deletes)

To override the default update behavior

1. Open the LINQ to SQL file in the **O/R Designer**. (Double-click the **Northwind.dbml** file in **Solution Explorer**.)
2. In **Server Explorer** or **Database Explorer**, expand the Northwind databases **Stored Procedures** node and locate the **InsertCustomers**, **UpdateCustomers**, and **DeleteCustomers** stored procedures.
3. Drag all three stored procedures onto the **O/R Designer**.

The stored procedures are added to the methods pane as [DataContext](#) methods. For more information, see [DataContext methods \(O/R Designer\)](#).

4. Select the **Customer** entity class in the **O/R Designer**.
5. In the **Properties** window, select the **Insert** property.
6. Click the ellipsis (...) next to **Use Runtime** to open the **Configure Behavior** dialog box.
7. Select **Customize**.
8. Select the **InsertCustomers** method in the **Customize** list.
9. Click **Apply** to save the configuration for the selected Class and Behavior.

NOTE

You can continue to configure the behavior for each class/behavior combination as long as you click **Apply** after you make each change. If you change the class or behavior before you click **Apply**, a warning dialog box providing an opportunity to apply any changes appears.

10. Select **Update** in the **Behavior** list.
11. Select **Customize**.
12. Select the **UpdateCustomers** method in the **Customize** list.

Inspect the list of **Method Arguments** and **Class Properties** and notice that there are two **Method Arguments** and two **Class Properties** for some columns in the table. This makes it easier to track changes and create statements that check for concurrency violations.

13. Map the **Original_CustomerID** method argument to the **CustomerID (Original)** class property.

NOTE

By default, method arguments will map to class properties when the names match. If property names are changed and no longer match between the table and the entity class, you might have to select the equivalent class property to map to if the **O/R Designer** cannot determine the correct mapping. Additionally, if method arguments do not have valid class properties to map to, you can set the **Class Properties** value to **(None)**.

14. Click **Apply** to save the configuration for the selected Class and Behavior.
15. Select **Delete** in the **Behavior** list.
16. Select **Customize**.
17. Select the **DeleteCustomers** method in the **Customize** list.
18. Map the **Original_CustomerID** method argument to the **CustomerID (Original)** class property.
19. Click **OK**.

NOTE

Although it is not an issue for this particular walkthrough, it is worth noting that LINQ to SQL handles database-generated values automatically for identity (auto-increment), rowguidcol (database-generated GUID), and timestamp columns during inserts and updates. Database-generated values in other column types will unexpectedly result in a null value. To return the database-generated values, you should manually set **IsDbGenerated** to `true` and **AutoSync** to one of the following: **AutoSync.Always**, **AutoSync.OnInsert**, or **AutoSync.OnUpdate**.

Test the application

Run the application again to verify that the **UpdateCustomers** stored procedure correctly updates the customer record in the database.

1. Press **F5**.
2. Modify a record in the grid to test the update behavior.
3. Add a new record to test the insert behavior.
4. Click the save button to save changes back to the database.

5. Close the form.
6. Press **F5** and verify that the updated record and the newly inserted record persisted.
7. Delete the new record you created in step 3 to test the delete behavior.
8. Click the save button to submit the changes and remove the deleted record from the database.
9. Close the form.
10. Press **F5** and verify that the deleted record was removed from the database.

NOTE

If your application uses SQL Server Express Edition, depending on the value of the **Copy to Output Directory** property of the database file, the changes may not appear when you press **F5** in step 10.

Next steps

Depending on your application requirements, there are several steps that you may want to perform after you create LINQ to SQL entity classes. Some enhancements you could make to this application include the following:

- Implement concurrency checking during updates. For information, see [Optimistic Concurrency: overview](#).
- Add LINQ queries to filter data. For information, see [Introduction to LINQ queries \(C#\)](#).

See also

- [LINQ to SQL tools in Visual Studio](#)
- [DataContext methods](#)
- [How to: Assign stored procedures to perform updates, inserts, and deletes](#)
- [LINQ to SQL](#)
- [LINQ to SQL queries](#)

How to: Assign stored procedures to perform updates, inserts, and deletes (O/R Designer)

8/5/2021 • 3 minutes to read • [Edit Online](#)

Stored procedures can be added to the **O/R Designer** and executed as typical [DataContext](#) methods. They can also be used to override the default LINQ to SQL run-time behavior that performs Inserts, Updates, and Deletes when changes are saved from entity classes to a database (for example, when calling the [SubmitChanges](#) method).

NOTE

If your stored procedure returns values that need to be sent back to the client (for example, values calculated in the stored procedure), create output parameters in your stored procedures. If you cannot use output parameters, write a partial method implementation instead of relying on overrides generated by the O/R Designer. Members mapped to database-generated values need to be set to appropriate values after successful completion of INSERT or UPDATE operations. For more information, see [Responsibilities of the Developer In Overriding Default Behavior](#).

NOTE

LINQ to SQL handles database-generated values automatically for identity (auto-increment), rowguidcol (database-generated GUID), and timestamp columns. Database-generated values in other column types will unexpectedly result in a null value. To return the database-generated values, you should manually set [IsDbGenerated](#) to **true** and [AutoSync](#) to one of the following: [AutoSync.Always](#), [AutoSync.OnInsert](#), or [AutoSync.OnUpdate](#).

Configure the Update Behavior of an Entity Class

By default, the logic to update a database (inserts, updates, and deletes) with changes that were made to the data in LINQ to SQL entity classes is provided by the LINQ to SQL runtime. The runtime creates default INSERT, UPDATE, and DELETE commands that are based on the schema of the table (the column and primary key information). When the default behavior is not desired, you can configure the update behavior by assigning specific stored procedures for performing the necessary inserts, updates, and deletes required to manipulate the data in your table. You can also do this when the default behavior is not generated, for example, when your entity classes map to views. Finally, you can override the default update behavior when the database requires table access through stored procedures.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To assign stored procedures to override the default behavior of an entity class

1. Open the **LINQ to SQL** file in the designer. (Double-click the **.dbml** file in **Solution Explorer**.)
2. In **Server Explorer** or **Database Explorer**, expand **Stored Procedures** and locate the stored procedures that you want to use for the Insert, Update, and/or Delete commands of the entity class.
3. Drag the stored procedure onto the **O/R Designer**.

The stored procedure is added to the methods pane as a [DataContext](#) method. For more information, see [DataContext Methods \(O/R Designer\)](#).

4. Select the entity class for which you want to use the stored procedure for performing updates.
5. In the **Properties** window, select the command to override (**Insert**, **Update**, or **Delete**).
6. Click the ellipsis (...) next to the words **Use Runtime** to open the **Configure Behavior** dialog box.
7. Select **Customize**.
8. Select the desired stored procedure in the **Customize** list.
9. Inspect the list of **Method Arguments** and **Class Properties** to verify that the **Method Arguments** map to the appropriate **Class Properties**. Map the original method arguments (`Original_<ArgumentName>`) to the original properties (`<PropertyName> (Original)`) for the `Update` and `Delete` commands.

NOTE

By default, method arguments map to class properties when the names match. If changed property names no longer match between the table and the entity class, you might have to select the equivalent class property to map to if the designer cannot determine the correct mapping.

10. Click **OK** or **Apply**.

NOTE

You can continue to configure the behavior for each class and behavior combination as long as you click **Apply** after you make each change. If you change the class or behavior before you click **Apply**, a warning dialog box appears and provides you an opportunity to apply your changes.

To revert to using the default runtime logic for updates, click the ellipsis next to the **Insert**, **Update**, or **Delete** command in the **Properties** window and then select **Use runtime** in the **Configure Behavior** dialog box.

See also

- [LINQ to SQL tools in Visual Studio](#)
- [DataContext methods](#)
- [LINQ to SQL \(.NET Framework\)](#)
- [Insert, update, and delete operations \(.NET Framework\)](#)

How to: Turn pluralization on and off (O/R Designer)

8/5/2021 • 2 minutes to read • [Edit Online](#)

By default, when you drag database objects that have names ending in `s` or `ies` from **Server Explorer** or **Database Explorer** onto the [LINQ to SQL tools in Visual Studio](#), the names of the generated entity classes are changed from plural to singular. This is done to more accurately represent the fact that the instantiated entity class maps to a single record of data. For example, adding a `Customers` table to the **O/R Designer** results in an entity class named `Customer` because the class will hold data for only a single customer.

NOTE

Pluralization is on by default only in the English-language version of Visual Studio.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To turn pluralization on and off

1. On the **Tools** menu, click **Options**.
2. In the **Options** dialog box, expand **Database Tools**.

NOTE

Select **Show all settings** if the **Database Tools** node is not visible.

3. Click **O/R Designer**.
4. Set **Pluralization of names** to **Enabled = False** to set the **O/R Designer** so that it does not change class names.
5. Set **Pluralization of names** to **Enabled = True** to apply pluralization rules to the class names of objects added to the **O/R Designer**.

See also

- [LINQ to SQL tools in Visual Studio](#)
- [LINQ to SQL](#)
- [Accessing data in Visual Studio](#)

How to: Create and configure datasets in Visual Studio

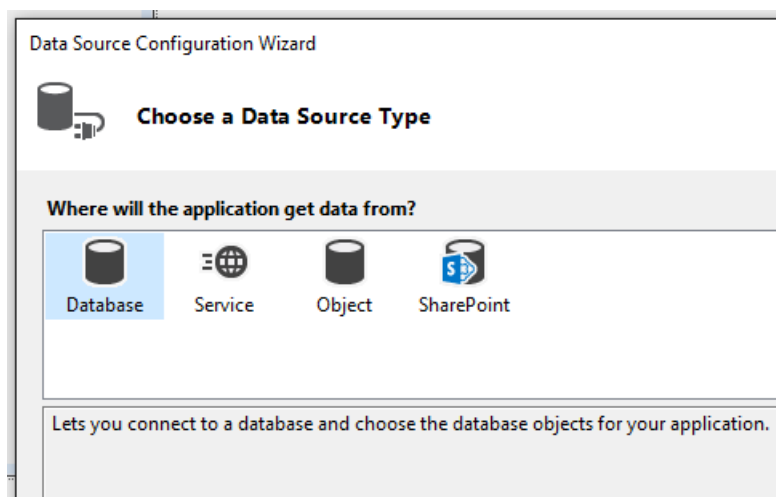
8/5/2021 • 3 minutes to read • [Edit Online](#)

A dataset is a set of objects that store data from a database in memory and support change tracking to enable create, read, update, and delete (CRUD) operations on that data without the need to be always connected to the database. Datasets were designed for simple *forms over data* business applications. For new applications, consider using Entity Framework to store and model data in memory. To work with datasets, you should have a basic knowledge of database concepts.

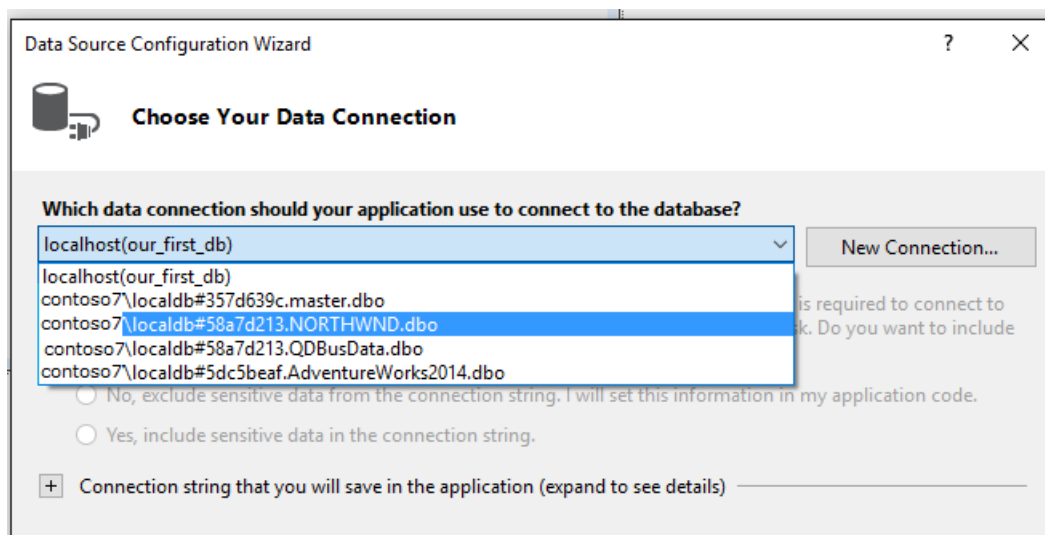
You can create a typed [DataSet](#) class in Visual Studio at design time by using the **Data Source Configuration Wizard**. For information on creating datasets programmatically, see [Creating a dataset \(ADO.NET\)](#).

Create a new dataset by using the Data Source Configuration Wizard

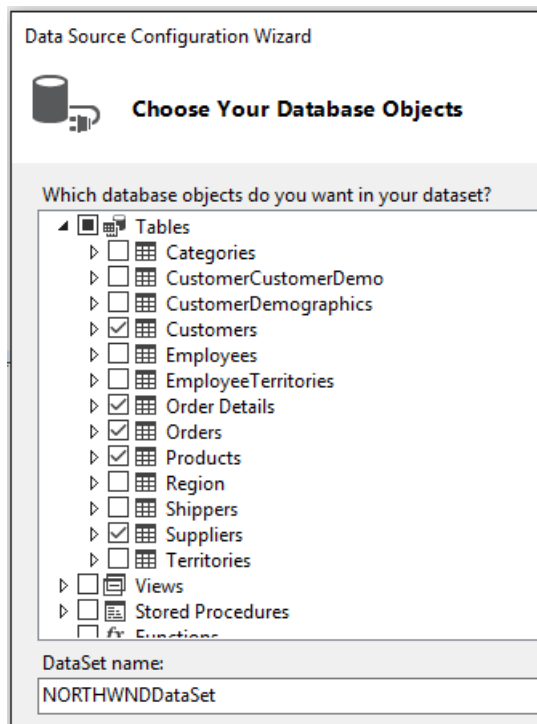
1. Open your project in Visual Studio, and then choose **Project > Add New Data Source** to start the **Data Source Configuration Wizard**.
2. Choose the type of data source to which you'll be connecting.



3. Choose the database or databases that will be the data source for your dataset.

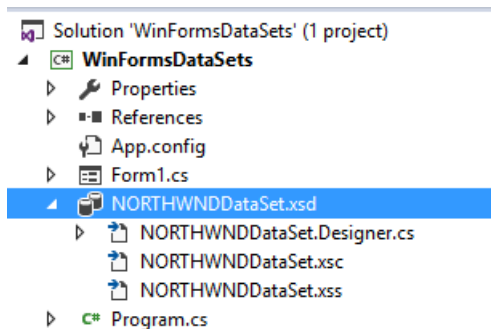


4. Choose the tables (or individual columns), stored procedures, functions, and views from the database that you want to be represented in the dataset.

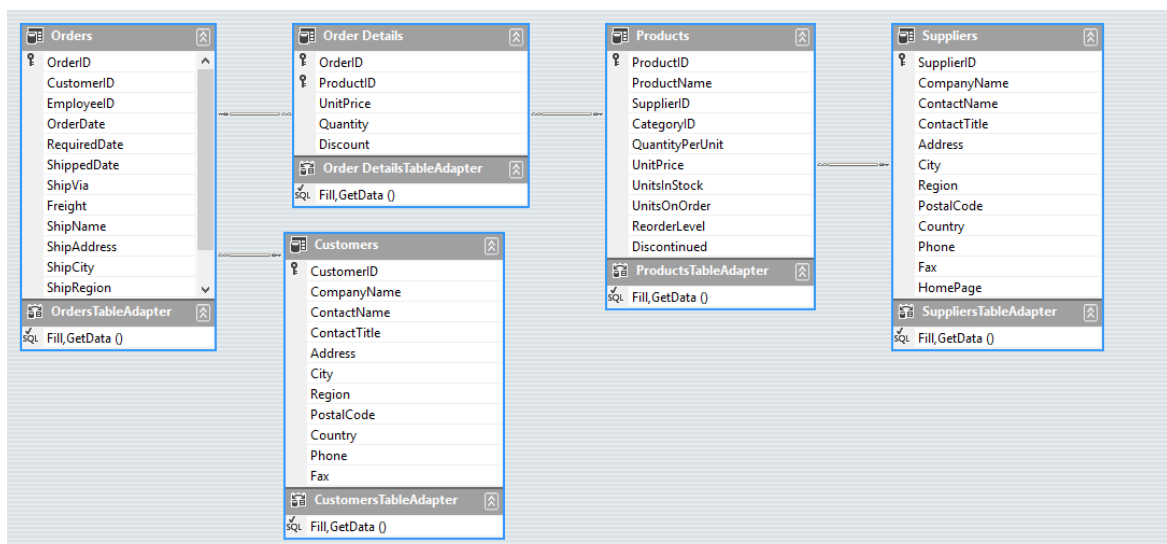


5. Click Finish.

The dataset appears as a node in Solution Explorer.



6. Click the dataset node in **Solution Explorer** to open the dataset in the **DataSet Designer**. Each table in the dataset has an associated **TableAdapter** object, which is represented at the bottom. The table adapter is used to populate the dataset and optionally to send commands to the database.



7. The relation lines that connect the tables represent table relationships, as defined in the database. By default, foreign-key constraints in a database are represented as a relation only, with the update and delete rules set to none. Typically, that is what you want. However, you can click the lines to bring up the **Relation** dialog, where you can change the behavior of hierarchical updates. For more information, see [Relationships in datasets](#) and [Hierarchical update](#).

Relation

Name: FK_Orders_Customers1

Specify the keys that relate tables in your dataset.

Parent Table: Orders Child Table: Customers

Columns:

| Key Columns | Foreign Key Columns |
|-------------|---------------------|
| OrderID | CustomerID |

Choose what to create

☐ Both Relation and Foreign Key Constraint

☒ Foreign Key Constraint Only

☐ Relation Only

Update Rule: Cascade

Delete Rule: Cascade

Accept/Reject Rule: None, Cascade, SetNull, SetDefault

☐ Nested Relation

OK Cancel

8. Click a table, table adapter, or column name in a table to see its properties in the **Properties** window. You can modify some of the values here. Just remember that you are modifying the dataset, not the source database.

Properties

OrderID DataColumn

Data

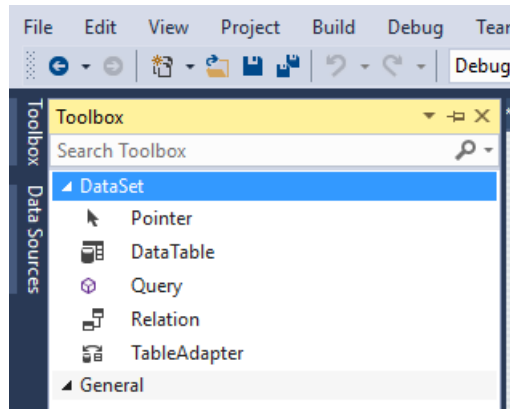
| | |
|-------------------|-------------------|
| AllowDBNull | False |
| AutoIncrement | True |
| AutoIncrementSeed | -1 |
| AutoIncrementStep | -1 |
| Caption | OrderID |
| DataType | System.Int32 |
| DateTimeMode | UnspecifiedLocal |
| DefaultValue | <DBNull> |
| Expression | |
| MaxLength | -1 |
| NullValue | (Throw exception) |
| ReadOnly | True |
| Source | OrderID |
| Unique | True |

Misc

| | |
|------|---------|
| Name | OrderID |
|------|---------|

9. You can add new tables or table adapters to the dataset, or add new queries for existing table adapters, or

specify new relations between tables by dragging those items from the **Toolbox** tab. This tab appears when the **DataSet Designer** is in focus.

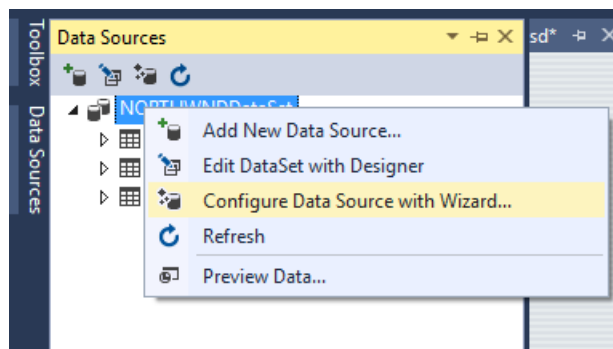


Next, you might want to specify how to populate the dataset with data. For that, you use the **TableAdapter Configuration Wizard**. For more information, see [Fill datasets by using TableAdapters](#).

Add a database table or other object to an existing dataset

This procedure shows how to add a table from the same database that you used to first create the dataset.

1. Click the dataset node in **Solution Explorer** to bring the **DataSet Designer** into focus.
2. Click the **Data Sources** tab in the left margin of Visual Studio, or type **data sources** in the search box.
3. Right-click the dataset node and select **Configure Data Source with Wizard**.



4. Use the wizard to specify which additional tables, stored procedures, or other database objects to add to the dataset.

Add a stand-alone data table to a dataset

1. Open your dataset in the **Dataset Designer**.
2. Drag a **DataTable** class from the **DataSet** tab of the **Toolbox** onto the **Dataset Designer**.
3. Add columns to define your data table. Right-click on the table and choose **Add > Column**. Use the **Properties** window to set the data type of the column and a key if necessary.

Stand-alone tables need to implement `Fill` logic in stand-alone tables so that you can fill them with data. For information on filling stand-alone data tables, see [Populating a DataSet from a DataAdapter](#).

See also

- [Dataset tools in Visual Studio](#)
- [Relationships in datasets](#)

- Hierarchical update
- Fill datasets by using TableAdapters

Create relationships between datasets

8/5/2021 • 5 minutes to read • [Edit Online](#)

Datasets that contain related data tables use [DataRelation](#) objects to represent a parent/child relationship between the tables and to return related records from one another. Adding related tables to datasets by using the **Data Source Configuration Wizard**, or the **Dataset Designer**, creates and configures the [DataRelation](#) object for you.

The [DataRelation](#) object performs two functions:

- It can make available the records related to a record you are working with. It provides child records if you are in a parent record ([GetChildRows](#)) and a parent record if you are working with a child record ([GetParentRow](#)).
- It can enforce constraints for referential integrity, such as deleting related child records when you delete a parent record.

It is important to understand the difference between a true join and the function of a [DataRelation](#) object. In a true join, records are taken from parent and child tables and put into a single, flat recordset. When you use a [DataRelation](#) object, no new recordset is created. Instead, the [DataRelation](#) tracks the relationship between tables and keeps parent and child records in sync.

DataRelation objects and constraints

A [DataRelation](#) object is also used to create and enforce the following constraints:

- A unique constraint, which guarantees that a column in the table contains no duplicates.
- A foreign-key constraint, which can be used to maintain referential integrity between a parent and child table in a dataset.

Constraints that you specify in a [DataRelation](#) object are implemented by automatically creating appropriate objects or setting properties. If you create a foreign-key constraint by using the [DataRelation](#) object, instances of the [ForeignKeyConstraint](#) class are added to the [DataRelation](#) object's [ChildKeyConstraint](#) property.

A unique constraint is implemented either by simply setting the [Unique](#) property of a data column to `true` or by adding an instance of the [UniqueConstraint](#) class to the [DataRelation](#) object's [ParentKeyConstraint](#) property. For information on suspending constraints in a dataset, see [Turn off constraints while filling a dataset](#).

Referential integrity rules

As part of the foreign-key constraint, you can specify referential integrity rules that are applied at three points:

- When a parent record is updated
- When a parent record is deleted
- When a change is accepted or rejected

The rules that you can make are specified in the [Rule](#) enumeration and are listed in the following table.

| FOREIGN-KEY CONSTRAINT RULE | ACTION |
|-----------------------------|---|
| Cascade | The change (update or delete) made to the parent record is also made in related records in the child table. |

| FOREIGN-KEY CONSTRAINT RULE | ACTION |
|-----------------------------|--|
| SetNull | Child records are not deleted, but the foreign key in the child records is set to DBNull . With this setting, child records can be left as "orphans"—that is, they have no relationship to parent records. Note: Using this rule can result in invalid data in the child table. |
| SetDefault | The foreign key in the related child records is set to its default value (as established by the column's DefaultValue property). |
| None | No change is made to related child records. With this setting, child records can contain references to invalid parent records. |

For more information about updates in dataset tables, see [Save data back to the database](#).

Constraint-only relations

When you create a [DataRelation](#) object, you have the option of specifying that the relation be used only to enforce constraints—that is, it will not also be used to access related records. You can use this option to generate a dataset that is slightly more efficient and that contains fewer methods than one with the related-records capability. However, you will not be able to access related records. For example, a constraint-only relation prevents you from deleting a parent record that still has child records, and you cannot access the child records through the parent.

Manually creating a data relation in the Dataset Designer

When you create data tables by using the data design tools in Visual Studio, relationships are created automatically if the information can be gathered from the source of your data. If you manually add data tables from the **DataSet** tab of the **Toolbox**, you may have to create the relationship manually. For information on creating [DataRelation](#) objects programmatically, see [Adding DataRelations](#).

Relationships between data tables appear as lines in the **Dataset Designer**, with a key and infinity glyph depicting the one-to-many aspect of the relationship. By default, the name of the relationship does not appear on the design surface.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To create a relationship between two data tables

1. Open your dataset in the **Dataset Designer**. For more information, see [Walkthrough: Creating a Dataset in the Dataset Designer](#).
2. Drag a **Relation** object from the **DataSet** toolbox onto the child data table in the relationship.

The **Relation** dialog box opens, populating the **Child Table** box with the table that you dragged the **Relation** object onto.
3. Select the parent table from the **Parent Table** box. The parent table contains records on the "one" side of a one-to-many relationship.
4. Verify that the correct child table is displayed in the **Child Table** box. The child table contains records on

the "many" side of a one-to-many relationship.

5. Type a name for the relationship in the **Name** box, or leave the default name based on the selected tables. This is the name of the actual [DataRelation](#) object in code.
6. Select the columns that join the tables in the **Key Columns** and **Foreign Key Columns** lists.
7. Select whether to create a relation, constraint, or both.
8. Select or clear the **Nested Relation** box. Selecting this option sets the [Nested](#) property to `true`, and it causes the child rows of the relation to be nested within the parent column when those rows are written as XML data or synchronized with [XmlDataDocument](#). For more information, see [Nesting DataRelations](#).
9. Set the rules to be enforced when you're making changes to records in these tables. For more information, see [Rule](#).
10. Click **OK** to create the relationship. A relation line appears on the designer between the two tables.

To display a relation name in the Dataset Designer

1. Open your dataset in the **Dataset Designer**. For more information, see [Walkthrough: Creating a Dataset in the Dataset Designer](#).
2. From the **Data** menu, select the **Show Relation Labels** command to display the relation name. Clear that command to hide the relation name.

See also

- [Create and configure datasets in Visual Studio](#)

Walkthrough: Create a Dataset with the Dataset Designer

8/5/2021 • 2 minutes to read • [Edit Online](#)

In this walkthrough you create a dataset using the **Dataset Designer**. The article takes you through the process of creating a new project and adding a new **DataSet** item to it. You'll learn how to create tables based on tables in a database without using a wizard.

Prerequisites

This walkthrough uses SQL Server Express LocalDB and the Northwind sample database.

1. If you don't have SQL Server Express LocalDB, install it either from the [SQL Server Express download page](#), or through the **Visual Studio Installer**. In the Visual Studio Installer, SQL Server Express LocalDB can be installed as part of the **Data storage and processing** workload, or as an individual component.
2. Install the Northwind sample database by following these steps:
 - a. In Visual Studio, open the **SQL Server Object Explorer** window. (SQL Server Object Explorer is installed as part of the **Data storage and processing** workload in the Visual Studio Installer.) Expand the **SQL Server** node. Right-click on your LocalDB instance and select **New Query**.

A query editor window opens.
 - b. Copy the [Northwind Transact-SQL script](#) to your clipboard. This T-SQL script creates the Northwind database from scratch and populates it with data.
 - c. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes executing and the Northwind database is created.

Create a New Windows Forms Application Project

1. In Visual Studio, on the **File** menu, select **New > Project**.
2. Expand either **Visual C#** or **Visual Basic** in the left-hand pane, then select **Windows Desktop**.
3. In the middle pane, select the **Windows Forms App** project type.
4. Name the project **DatasetDesignerWalkthrough**, and then choose **OK**.

Visual Studio adds the project to **Solution Explorer** and display a new form in the designer.

Add a New Dataset to the Application

1. On the **Project** menu, select **Add New Item**.

The **Add New Item** dialog box appears.

2. In the left-hand pane, select **Data**, then select **DataSet** in the middle pane.
3. Name the Dataset **NorthwindDataset**, and then choose **Add**.

Visual Studio adds a file called **NorthwindDataset.xsd** to the project and opens it in the **Dataset Designer**.

Create a Data Connection in Server Explorer

1. On the **View** menu, click **Server Explorer**.
2. In **Server Explorer**, click the **Connect to Database** button.
3. Create a connection to the Northwind sample database.

Create the Tables in the Dataset

This section explains how to add tables to the dataset.

To create the Customers table

1. Expand the data connection you created in **Server Explorer**, and then expand the **Tables** node.
2. Drag the **Customers** table from **Server Explorer** onto the **Dataset Designer**.

A **Customers** data table and **CustomersTableAdapter** are added to the dataset.

To create the Orders table

- Drag the **Orders** table from **Server Explorer** onto the **Dataset Designer**.

An **Orders** data table, **OrdersTableAdapter**, and data relation between the **Customers** and **Orders** tables are added to the dataset.

To create the OrderDetails table

- Drag the **Order Details** table from **Server Explorer** onto the **Dataset Designer**.

An **Order Details** data table, **OrderDetailsTableAdapter**, and a data relation between the **Orders** and **OrderDetails** tables are added to the dataset.

Next Steps

- Save the dataset.
- Select items in the **Data Sources** window and drag them onto a form. For more information, see [Bind Windows Forms controls to data in Visual Studio](#).
- Add more queries to the TableAdapters.
- Add validation logic to the [ColumnChanging](#) or [RowChanging](#) events of the data tables in the dataset. For more information, see [Validate data in datasets](#).

See also

- [Create and configure datasets in Visual Studio](#)
- [Bind Windows Forms controls to data in Visual Studio](#)
- [Bind controls to data in Visual Studio](#)
- [Validate data](#)

Walkthrough: Create a DataTable in the Dataset Designer

8/5/2021 • 2 minutes to read • [Edit Online](#)

This walkthrough explains how to create a [DataTable](#) (without a TableAdapter) using the **Dataset Designer**. For information on creating data tables that include TableAdapters, see [Create and configure TableAdapters](#).

Create a new Windows Forms application

1. In Visual Studio, on the **File** menu, select **New** > **Project**.
2. Expand either **Visual C#** or **Visual Basic** in the left-hand pane, then select **Windows Desktop**.
3. In the middle pane, select the **Windows Forms App** project type.
4. Name the project **DataTableWalkthrough**, and then choose **OK**.

The **DataTableWalkthrough** project is created and added to **Solution Explorer**.

Add a new Dataset to the application

1. On the **Project** menu, select **Add New Item**.

The **Add New Item** dialog box appears.

2. In the left-hand pane, select **Data**, then select **DataSet** in the middle pane.
3. Choose **Add**.

Visual Studio adds a file called **DataSet1.xsd** to the project and opens it in the **Dataset Designer**.

Add a new DataTable to the Dataset

1. Drag a **DataTable** from the **DataSet** tab of the **Toolbox** onto the **Dataset Designer**.

A table named **DataTable1** is added to the dataset.

2. Click the title bar of **DataTable1** and rename it **Music**.

Add columns to the DataTable

1. Right-click the **Music** table. Point to **Add**, and then click **Column**.
2. Name the column **SongID**.
3. In the **Properties** window, set the **DataType** property to **System.Int16**.
4. Repeat this process and add the following columns:

SongTitle : **System.String**

Artist : **System.String**

Genre : **System.String**

Set the Primary Key for the table

All data tables should have a primary key. A primary key uniquely identifies a specific record in a data table.

To set the primary key, right-click the **SongID** column, and then click **Set Primary Key**. A key icon appears next to the **SongID** column.

Save Your Project

To save the **DataTableWalkthrough** project, on the **File** menu, select **Save All**.

See also

- [Create and configure datasets in Visual Studio](#)
- [Bind controls to data in Visual Studio](#)
- [Validating data](#)

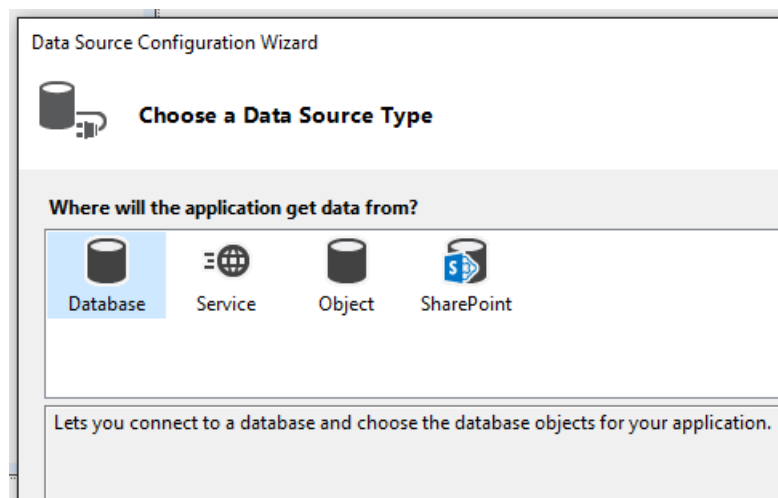
Create and configure TableAdapters

8/5/2021 • 6 minutes to read • [Edit Online](#)

TableAdapters provide communication between your application and a database. They connect to the database, run queries or stored procedures, and either return a new data table or fill an existing [DataTable](#) with the returned data. TableAdapters can also send updated data from your application back to the database.

TableAdapters are created for you when you perform one of the following actions:

- Drag database objects from **Server Explorer** into the **Dataset Designer**.
- Run the Data Source Configuration Wizard, and select either the **Database** or **Web Service** data source type.



You can also create a new TableAdapter and configure it with a data source by dragging a TableAdapter from the **Toolbox** to an empty region in the **Dataset Designer** surface.

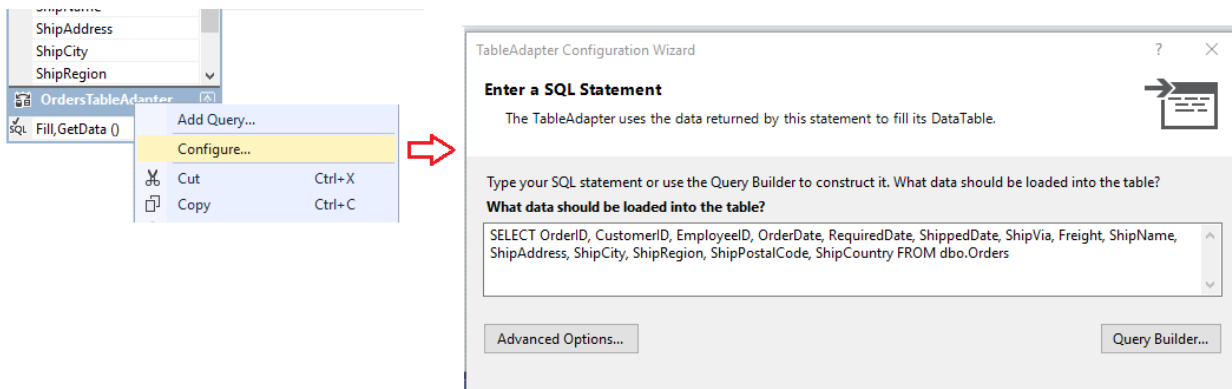
For an introduction to TableAdapters, see [Fill datasets by using TableAdapters](#).

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

Use the TableAdapter Configuration Wizard

Run the **TableAdapter Configuration Wizard** to create or edit TableAdapters and their associated DataTables. You can configure an existing TableAdapter by right-clicking on it in the **Dataset Designer**.



If you drag a new TableAdapter from the Toolbox when the **Dataset Designer** is in focus, the wizard starts and prompts you to specify which data source the TableAdapter should connect to. On the next page, the wizard asks what kind of commands it should use to communicate with the database, either SQL statements or stored procedures. (You won't see this if you are configuring a TableAdapter that is already associated with a data source.)

- You have the option to create a new stored procedure in the underlying database if you have the correct permissions for the database. If you don't have these permissions, this won't be an option.
- You can also choose to run existing stored procedures for the **SELECT**, **INSERT**, **UPDATE**, and **DELETE** commands of the TableAdapter. The stored procedure that's assigned to the **Update** command, for example, is run when the `TableAdapter.Update()` method is called.

Map parameters from the selected stored procedure to the corresponding columns in the data table. For example, if your stored procedure accepts a parameter named `@CompanyName` that it passes to the `CompanyName` column in the table, set the **Source Column** of the `@CompanyName` parameter to `CompanyName`.

NOTE

The stored procedure that's assigned to the **SELECT** command is run by calling the method of the TableAdapter that you name in the next step of the wizard. The default method is `Fill`, so the code that's typically used to run the **SELECT** procedure is `TableAdapter.Fill(tableName)`. If you change the default name from `Fill`, substitute `Fill` with the name you assign, and replace "TableAdapter" with the actual name of the TableAdapter (for example, `CustomersTableAdapter`).

- Selecting the **Create methods to send updates directly to the database** option is equivalent to setting the `GenerateDBDirectMethods` property to true. The option is unavailable when the original SQL statement does not provide enough information or the query is not an updateable query. This situation can occur, for example, in **JOIN** queries and queries that return a single (scalar) value.

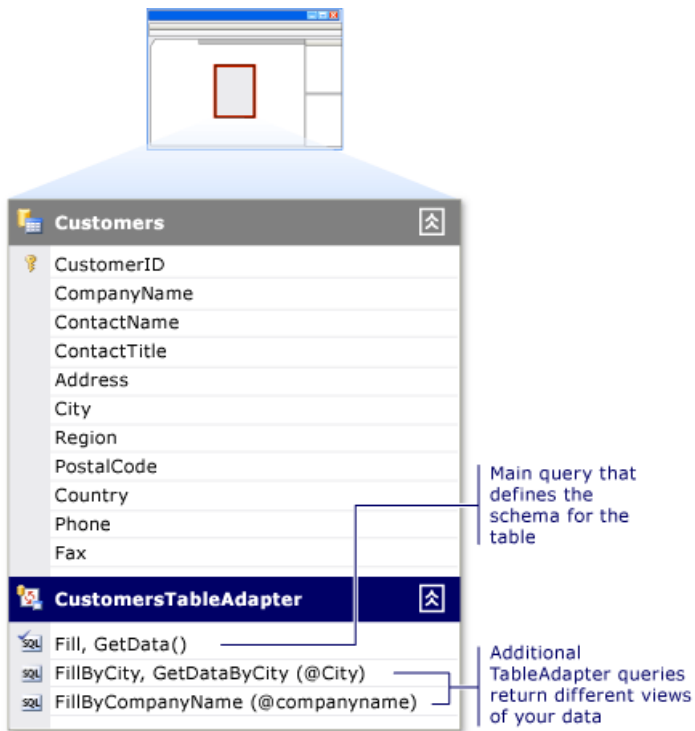
The **Advanced Options** in the wizard enable you to:

- Generate **INSERT**, **UPDATE**, and **DELETE** statements based on the **SELECT** statement that's defined on the **Generate SQL statements** page
- Use optimistic concurrency
- Specify whether to refresh the data table after **INSERT** and **UPDATE** statements are run

Configure a TableAdapter's Fill method

Sometimes you might want to change the schema of the TableAdapter's table. To do this, you modify the TableAdapter's primary `Fill` method. TableAdapters are created with a primary `Fill` method that defines the schema of the associated data table. The primary `Fill` method is based on the query or stored procedure you entered when you originally configured the TableAdapter. It's the first (topmost) method under the data table in

the DataSet Designer.



Any changes that you make to the TableAdapter's main `Fill` method are reflected in the schema of the associated data table. For example, removing a column from the query in the main `Fill` method also removes the column from the associated data table. Additionally, removing the column from the main `Fill` method removes the column from any additional queries for that TableAdapter.

You can use the TableAdapter Query Configuration Wizard to create and edit additional queries for the TableAdapter. These additional queries must conform to the table schema, unless they return a scalar value. Each additional query has a name that you specify.

The following example shows you how to call an additional query named `FillByCity`:

```
CustomersTableAdapter.FillByCity(NorthwindDataSet.Customers, "Seattle")
```

To start the TableAdapter Query Configuration Wizard with a new query

1. Open your dataset in the **Dataset Designer**.
2. If you are creating a new query, drag a **Query** object from the **DataSet** tab of the **Toolbox** onto a **DataTable**, or select **Add Query** from the TableAdapter's shortcut menu. You can also drag a **Query** object onto an empty area of the **Dataset Designer**, which creates a TableAdapter without an associated **DataTable**. These queries can only return single (scalar) values, or run UPDATE, INSERT, or DELETE commands against the database.
3. On the **Choose Your Data Connection** screen, select or create the connection that the query will use.

NOTE

This screen only appears when the designer can't determine the proper connection to use, or when no connections are available.

4. On the **Choose a Command Type** screen, select from the following methods of fetching data from the database:

- **Use SQL statements** enables you to type a SQL statement to select the data from your database.

- **Create new stored procedure** enables you to have the wizard create a new stored procedure (in the database) based on the specified SELECT statement.
- **Use existing stored procedures** enables you to run an existing stored procedure when running the query.

To start the TableAdapter Query Configuration wizard on an existing query

- If you are editing an existing TableAdapter query, right-click the query, and then choose **Configure** from the shortcut menu.

NOTE

Right-clicking the main query of a TableAdapter reconfigures the TableAdapter and [DataTable](#) schema. Right-clicking an additional query on a TableAdapter, however, configures the selected query only. The **TableAdapter Configuration Wizard** reconfigures the TableAdapter definition, while the **TableAdapter Query Configuration Wizard** reconfigures the selected query only.

To add a global query to a TableAdapter

- Global queries are SQL queries that return either a single (scalar) value or no value. Typically, global functions perform database operations such as inserts, updates, and deletes. They also aggregate information, such as a count of customers in a table or the total charges for all items in a particular order.

You add global queries by dragging a **Query** object from the **DataSet** tab of the **Toolbox** onto an empty area of the **Dataset Designer**.

- Provide a query that performs the desired task, for example,

```
SELECT COUNT(*) AS CustomerCount FROM Customers .
```

NOTE

Dragging a **Query** object directly onto the **Dataset Designer** creates a method that returns only a scalar (single) value. While the query or stored procedure you select might return more than a single value, the method that's created by the wizard only returns a single value. For example, the query might return the first column of the first row of the returned data.

See also

- [Fill datasets by using TableAdapters](#)

Create parameterized TableAdapter queries

8/5/2021 • 2 minutes to read • [Edit Online](#)

A parameterized query returns data that meets the conditions of a WHERE clause within the query. For example, you can parameterize a customer list to display only customers in a certain city by adding `WHERE City = @City` to the end of the SQL statement that returns a list of customers.

You create parameterized TableAdapter queries in the **Dataset Designer**. You can also create them in a Windows application with the **Parameterize Data Source** command on the **Data** menu. The **Parameterize Data Source** command creates controls on your form where you can input the parameter values and run the query.

NOTE

When constructing a parameterized query, use the parameter notation that's specific to the database you're coding against. For example, Access and OleDb data sources use the question mark '?' to denote parameters, so the WHERE clause would look like this: `WHERE City = ?`.

Create a parameterized TableAdapter query

To create a parameterized query in the Dataset Designer

- Create a new TableAdapter, adding a WHERE clause with the desired parameters to the SQL statement. For more information, see [Create and configure TableAdapters](#).

-or-

- Add a query to an existing TableAdapter, adding a WHERE clause with the desired parameters to the SQL statement.

To create a parameterized query while designing a data-bound form

1. Select a control on your form that is already bound to a dataset. For more information, see [Bind Windows Forms controls to data in Visual Studio](#).
2. On the **Data** menu, select **Add Query**.
3. Complete the **Search Criteria Builder** dialog box, adding a WHERE clause with the desired parameters to the SQL statement.

To add a query to an existing data-bound form

1. Open the form in the **Windows Forms Designer**.
2. On the **Data** menu, select **Add Query** or **Data Smart Tags**.

NOTE

If **Add Query** is not available on the **Data** menu, select a control on the form that displays the data source you want to add the parameterization to. For example, if the form displays data in a [DataGridView](#) control, select it. If the form displays data in individual controls, select any data-bound control.

3. In the **Select data source table** area, select the table to which you want to add parameterization.

4. Type a name in the **New query name** box if you are creating a new query.

-or-

Select a query in the **Existing query name** box.

5. In the **Query Text** box, type a query that takes parameters.

6. Select **OK**.

A control to input the parameter and a **Load** button are added to the form in a **ToolStrip** control.

Query for null values

TableAdapter parameters can be assigned null values when you want to query for records that have no current value. For example, consider the following query that has a `ShippedDate` parameter in its `WHERE` clause:

```
SELECT CustomerID, OrderDate, ShippedDate
FROM Orders
WHERE (ShippedDate = @ShippedDate) OR (ShippedDate IS NULL)
```

If this were a query on a TableAdapter, you could query for all orders that have not been shipped with the following code:

```
ordersTableAdapter.FillByShippedDate(northwindDataSet.Orders, null);
```

```
OrdersTableAdapter.FillByShippedDate(NorthwindDataSet.Orders, Nothing)
```

To enable a query to accept null values:

1. In the **Dataset Designer**, select the TableAdapter query that needs to accept null parameter values.
2. In the **Properties** window, select **Parameters**, then click the ellipsis (...) button to open the **Parameters Collection Editor**.
3. Select the parameter that allows null values and set the **AllowDBNull** property to `true`.

See also

- [Fill datasets by using TableAdapters](#)

Directly access the database with a TableAdapter

8/5/2021 • 2 minutes to read • [Edit Online](#)

In addition to the `InsertCommand`, `UpdateCommand`, and `DeleteCommand`, TableAdapters are created with methods that can be run directly against the database. You can call these methods (`TableAdapter.Insert`, `TableAdapter.Update`, and `TableAdapter.Delete`) to manipulate data directly in the database.

If you don't want to create these direct methods, set the TableAdapter's `GenerateDbDirectMethods` property to `false` in the **Properties** window. If any queries are added to a TableAdapter in addition to the TableAdapter's main query, they are standalone queries that don't generate these `DbDirect` methods.

Send commands directly to a database

Call the TableAdapter `DbDirect` method that performs the task you are trying to accomplish.

To insert new records directly into a database

- Call the TableAdapter's `Insert` method, passing in the values for each column as parameters. The following procedure uses the `Region` table in the Northwind database as an example.

NOTE

If you do not have an instance available, instantiate the TableAdapter that you want to use.

```
Dim regionTableAdapter As New NorthwindDataSetTableAdapters.RegionTableAdapter

regionTableAdapter.Insert(5, "NorthWestern")
```

```
NorthwindDataSetTableAdapters.RegionTableAdapter regionTableAdapter =
    new NorthwindDataSetTableAdapters.RegionTableAdapter();

regionTableAdapter.Insert(5, "NorthWestern");
```

To update records directly in a database

- Call the TableAdapter's `Update` method, passing in the new and original values for each column as parameters.

NOTE

If you do not have an instance available, instantiate the TableAdapter that you want to use.

```
Dim regionTableAdapter As New NorthwindDataSetTableAdapters.RegionTableAdapter

regionTableAdapter.Update(1, "East", 1, "Eastern")
```

```
NorthwindDataSetTableAdapters.RegionTableAdapter regionTableAdapter =  
    new NorthwindDataSetTableAdapters.RegionTableAdapter();  
  
regionTableAdapter.Update(1, "East", 1, "Eastern");
```

To delete records directly from a database

- Call the TableAdapter's `Delete` method, passing in the values for each column as parameters of the `Delete` method. The following procedure uses the `Region` table in the Northwind database as an example.

NOTE

If you do not have an instance available, instantiate the TableAdapter that you want to use.

```
Dim regionTableAdapter As New NorthwindDataSetTableAdapters.RegionTableAdapter  
  
regionTableAdapter.Delete(5, "NorthWestern")
```

```
NorthwindDataSetTableAdapters.RegionTableAdapter regionTableAdapter =  
    new NorthwindDataSetTableAdapters.RegionTableAdapter();  
  
regionTableAdapter.Delete(5, "NorthWestern");
```

See also

- [Fill datasets by using TableAdapters](#)

Turn off constraints while filling a dataset

8/5/2021 • 2 minutes to read • [Edit Online](#)

If a dataset contains constraints (such as foreign-key constraints), they can raise errors related to the order of operations that are performed against the dataset. For example, loading child records before loading related parent records can violate a constraint and cause an error. As soon as you load a child record, the constraint checks for the related parent record and raises an error.

If there were no mechanism to allow temporary constraint suspension, an error would be raised every time you tried to load a record into the child table. Another way to suspend all constraints in a dataset is with the [BeginEdit](#), and [EndEdit](#) properties.

NOTE

Validation events (for example, [ColumnChanging](#) and [RowChanging](#)) will not be raised when constraints are turned off.

To suspend update constraints programmatically

- The following example shows how to temporarily turn off constraint checking in a dataset:

```
dataSet1.EnforceConstraints = false;  
// Perform some operations on the dataset  
dataSet1.EnforceConstraints = true;
```

```
DataSet1.EnforceConstraints = False  
' Perform some operations on the dataset  
DataSet1.EnforceConstraints = True
```

To suspend update constraints using the Dataset Designer

- Open your dataset in the **Dataset Designer**. For more information, see [Walkthrough: Creating a dataset in the Dataset Designer](#).
- In the **Properties** window, set the [EnforceConstraints](#) property to `false`.

See also

- [Fill datasets by using TableAdapters](#)
- [Relationships in datasets](#)

Extend the functionality of a TableAdapter

8/5/2021 • 2 minutes to read • [Edit Online](#)

You can extend the functionality of a TableAdapter by adding code to the TableAdapter's partial class file.

The code that defines a TableAdapter is regenerated when any changes are made to the TableAdapter in the **Dataset Designer**, or when a wizard modifies the configuration of a TableAdapter. To prevent your code from being deleted during the regeneration of a TableAdapter, add code to the TableAdapter's partial class file.

Partial classes allow code for a specific class to be divided among multiple physical files. For more information, see [Partial](#) or [partial \(Type\)](#).

Locate TableAdapters in code

While TableAdapters are designed with the **Dataset Designer**, the TableAdapter classes that are generated are not nested classes of [DataSet](#). TableAdapters are located in a namespace based on the name of the TableAdapter's associated dataset. For example, if your application contains a dataset named `HRDataSet`, the TableAdapters would be located in the `HRDataSetTableAdapters` namespace. (The naming convention follows this pattern: *DataSetName* + `TableAdapters`).

The following example assumes a TableAdapter named `CustomersTableAdapter` is in a project with `NorthwindDataSet`.

To create a partial class for a TableAdapter

1. Add a new class to your project by going to the **Project** menu and selecting **Add Class**.
2. Name the class `CustomersTableAdapterExtended`.
3. Select **Add**.
4. Replace the code with the correct namespace and partial class name for your project as follows:

```
namespace NorthwindDataSetTableAdapters
{
    public partial class CustomersTableAdapter
    {
        // Add user code here. For example:
        public override string ToString()
        {
            return "Overridden in the partial class.";
        }
    }
}
```

```
Namespace NorthwindDataSetTableAdapters

    Partial Class CustomersTableAdapter

        ' Add user code here. For example:
        Public Overrides Function ToString() As String
            Return "Overridden in the partial class."
        End Function
    End Class
End Namespace
```

See also

- [Fill datasets by using TableAdapters](#)

Read XML data into a dataset

8/5/2021 • 4 minutes to read • [Edit Online](#)

ADO.NET provides simple methods for working with XML data. In this walkthrough, you create a Windows application that loads XML data into a dataset. The dataset is then displayed in a [DataGridView](#) control. Finally, an XML schema based on the contents of the XML file is displayed in a text box.

Create a new project

Create a new **Windows Forms App** project for either C# or Visual Basic. Name the project **ReadingXML**.

Generate the XML file to be read into the dataset

Because this walkthrough focuses on reading XML data into a dataset, the contents of an XML file is provided.

1. On the **Project** menu, select **Add New Item**.
2. Select **XML File**, name the file **authors.xml**, and then select **Add**.

The XML file loads into the designer and is ready for edit.

3. Paste the following XML data into the editor below the XML declaration:

```

<Authors_Table>
  <authors>
    <au_id>172-32-1176</au_id>
    <au_lname>White</au_lname>
    <au_fname>Johnson</au_fname>
    <phone>408 496-7223</phone>
    <address>10932 Bigge Rd.</address>
    <city>Menlo Park</city>
    <state>CA</state>
    <zip>94025</zip>
    <contract>>true</contract>
  </authors>
  <authors>
    <au_id>213-46-8915</au_id>
    <au_lname>Green</au_lname>
    <au_fname>Margie</au_fname>
    <phone>415 986-7020</phone>
    <address>309 63rd St. #411</address>
    <city>Oakland</city>
    <state>CA</state>
    <zip>94618</zip>
    <contract>>true</contract>
  </authors>
  <authors>
    <au_id>238-95-7766</au_id>
    <au_lname>Carson</au_lname>
    <au_fname>Cheryl</au_fname>
    <phone>415 548-7723</phone>
    <address>589 Darwin Ln.</address>
    <city>Berkeley</city>
    <state>CA</state>
    <zip>94705</zip>
    <contract>>true</contract>
  </authors>
  <authors>
    <au_id>267-41-2394</au_id>
    <au_lname>Hunter</au_lname>
    <au_fname>Anne</au_fname>
    <phone>408 286-2428</phone>
    <address>22 Cleveland Av. #14</address>
    <city>San Jose</city>
    <state>CA</state>
    <zip>95128</zip>
    <contract>>true</contract>
  </authors>
  <authors>
    <au_id>274-80-9391</au_id>
    <au_lname>Straight</au_lname>
    <au_fname>Dean</au_fname>
    <phone>415 834-2919</phone>
    <address>5420 College Av.</address>
    <city>Oakland</city>
    <state>CA</state>
    <zip>94609</zip>
    <contract>>true</contract>
  </authors>
</Authors_Table>

```

4. On the **File** menu, select **Save authors.xml**.

Create the user interface

The user interface for this application consists of the following:

- A [DataGridView](#) control that displays the contents of the XML file as data.

- A [TextBox](#) control that displays the XML schema for the XML file.
- Two [Button](#) controls.
 - One button reads the XML file into the dataset and displays it in the [DataGridView](#) control.
 - A second button extracts the schema from the dataset, and through a [StringWriter](#) displays it in the [TextBox](#) control.

To add controls to the form

1. Open `Form1` in design view.
2. From the **Toolbox**, drag the following controls onto the form:
 - One [DataGridView](#) control
 - One [TextBox](#) control
 - Two [Button](#) controls
3. Set the following properties:

| CONTROL | PROPERTY | SETTING |
|-----------------------|-------------------|-------------------------------|
| <code>TextBox1</code> | Multiline | <code>true</code> |
| | ScrollBars | Vertical |
| <code>Button1</code> | Name | <code>ReadXmlButton</code> |
| | Text | <code>Read XML</code> |
| <code>Button2</code> | Name | <code>ShowSchemaButton</code> |
| | Text | <code>Show Schema</code> |

Create the dataset that receives the XML data

In this step, you create a new dataset named `authors`. For more information about datasets, see [Dataset tools in Visual Studio](#).

1. In **Solution Explorer**, select the source file for **Form1**, and then select the **View Designer** button on the **Solution Explorer** toolbar.
2. From the **Toolbox**, **Data tab**, drag a **DataSet** onto **Form1**.
3. In the **Add Dataset** dialog box, select **Untyped dataset**, and then select **OK**.

DataSet1 is added to the component tray.

4. In the **Properties** window, set the **Name** and [DataSetName](#) properties for `AuthorsDataSet`.

Create the event handler to read the XML file into the dataset

The **Read XML** button reads the XML file into the dataset. It then sets properties on the [DataGridView](#) control that bind it to the dataset.

1. In **Solution Explorer**, select **Form1**, and then select the **View Designer** button on the **Solution**

Explorer toolbar.

2. Select the **Read XML** button.

The **Code Editor** opens at the `ReadXmlButton_Click` event handler.

3. Type the following code into the `ReadXmlButton_Click` event handler:

```
private void ReadXmlButton_Click(object sender, EventArgs e)
{
    string filePath = "Complete path where you saved the XML file";

    AuthorsDataSet.ReadXml(filePath);

    dataGridView1.DataSource = AuthorsDataSet;
    dataGridView1.DataMember = "authors";
}
```

```
Private Sub ReadXmlButton_Click() Handles ReadXmlButton.Click

    Dim filePath As String = "Complete path where you saved the XML file"

    AuthorsDataSet.ReadXml(filePath)

    DataGridView1.DataSource = AuthorsDataSet
    DataGridView1.DataMember = "authors"
End Sub
```

4. In the `ReadXMLButton_Click` event handler code, change the `filepath =` entry to the correct path.

Create the event handler to display the schema in the textbox

The **Show Schema** button creates a [StringWriter](#) object that's filled with the schema and is displayed in the [TextBox](#) control.

1. In **Solution Explorer**, select **Form1**, and then select the **View Designer** button.
2. Select the **Show Schema** button.

The **Code Editor** opens at the `ShowSchemaButton_Click` event handler.

3. Paste the following code into the `ShowSchemaButton_Click` event handler.

```
private void ShowSchemaButton_Click(object sender, EventArgs e)
{
    System.IO.StringWriter swXML = new System.IO.StringWriter();
    AuthorsDataSet.WriteXmlSchema(swXML);
    textBox1.Text = swXML.ToString();
}
```

```
Private Sub ShowSchemaButton_Click() Handles ShowSchemaButton.Click

    Dim swXML As New System.IO.StringWriter()
    AuthorsDataSet.WriteXmlSchema(swXML)
    TextBox1.Text = swXML.ToString
End Sub
```

Test the form

You can now test the form to make sure it behaves as expected.

1. Select **F5** to run the application.
2. Select the **Read XML** button.

The DataGridView displays the contents of the XML file.

3. Select the **Show Schema** button.

The text box displays the XML schema for the XML file.

Next steps

This walkthrough teaches you the basics of reading an XML file into a dataset, as well as creating a schema based on the contents of the XML file. Here are some tasks that you might do next:

- Edit the data in the dataset and write it back out as XML. For more information, see [WriteXml](#).
- Edit the data in the dataset and write it out to a database.

See also

- [Access data in Visual Studio](#)
- [XML tools in Visual Studio](#)

Edit data in datasets

8/5/2021 • 5 minutes to read • [Edit Online](#)

You edit data in data tables much like you edit the data in a table in any database. The process can include inserting, updating, and deleting records in the table. In a data-bound form, you can specify which fields are user-editable. In those cases, the data-binding infrastructure handles all the change tracking so that the changes can be sent back to the database later. If you programmatically make edits to data, and you intend to send those changes back to the database, you must use the objects and methods that do the change tracking for you.

In addition to changing the actual data, you can also query a [DataTable](#) to return specific rows of data. For example, you might query for individual rows, specific versions of rows (original and proposed), rows that have changed, or rows that have errors.

To edit rows in a dataset

To edit an existing row in a [DataTable](#), you need to locate the [DataRow](#) you want to edit, and then assign the updated values to the desired columns.

If you don't know the index of the row you want to edit, use the `FindBy` method to search by the primary key:

```
NorthwindDataSet.CustomersRow customersRow =  
    northwindDataSet1.Customers.FindByCustomerID("ALFKI");  
  
customersRow.CompanyName = "Updated Company Name";  
customersRow.City = "Seattle";;
```

```
Dim customersRow As NorthwindDataSet.CustomersRow  
customersRow = NorthwindDataSet1.Customers.FindByCustomerID("ALFKI")  
  
customersRow.CompanyName = "Updated Company Name"  
customersRow.City = "Seattle"
```

If you know the row index, you can access and edit rows as follows:

```
northwindDataSet1.Customers[4].CompanyName = "Updated Company Name";  
northwindDataSet1.Customers[4].City = "Seattle";
```

```
NorthwindDataSet1.Customers(4).CompanyName = "Updated Company Name"  
NorthwindDataSet1.Customers(4).City = "Seattle"
```

To insert new rows into a dataset

Applications that use data-bound controls typically add new records through the **Add New** button on a [BindingNavigator control](#).

To manually add new records to a dataset, create a new data row by calling the method on the [DataTable](#). Then, add the row to the [DataRow](#) collection ([Rows](#)) of the [DataTable](#):

```
NorthwindDataSet.CustomersRow newCustomersRow =  
    northwindDataSet1.Customers.NewCustomersRow();  
  
newCustomersRow.CustomerID = "ALFKI";  
newCustomersRow.CompanyName = "Alfreds Futterkiste";  
  
northwindDataSet1.Customers.Rows.Add(newCustomersRow);
```

```
Dim newCustomersRow As NorthwindDataSet.CustomersRow  
newCustomersRow = NorthwindDataSet1.Customers.NewCustomersRow()  
  
newCustomersRow.CustomerID = "ALFKI"  
newCustomersRow.CompanyName = "Alfreds Futterkiste"  
  
NorthwindDataSet1.Customers.Rows.Add(newCustomersRow)
```

In order to retain the information that the dataset needs to send updates to the data source, use the [Delete](#) method to remove rows in a data table. For example, if your application uses a [TableAdapter](#) (or [DataAdapter](#)), the [TableAdapter](#)'s `Update` method deletes rows in the database that have a [RowState](#) of [Deleted](#).

If your application does not need to send updates back to a data source, it's possible to remove records by directly accessing the data row collection ([Remove](#)).

To delete records from a data table

- Call the [Delete](#) method of a [DataRow](#).

This method doesn't physically remove the record. Instead, it marks the record for deletion.

NOTE

If you get the count property of a [DataRowCollection](#), the resulting count includes records that have been marked for deletion. To get an accurate count of records that aren't marked for deletion, you can loop through the collection looking at the [RowState](#) property of each record. (Records marked for deletion have a [RowState](#) of [Deleted](#).) Alternatively, you can create a data view of a dataset that filters based on row state and get the count property from there.

The following example shows how to call the [Delete](#) method to mark the first row in the `Customers` table as deleted:

```
northwindDataSet1.Customers.Rows[0].Delete();
```

```
NorthwindDataSet1.Customers.Rows(0).Delete()
```

Determine if there are changed rows

When changes are made to records in a dataset, information about those changes is stored until you commit them. You commit the changes when you call the `AcceptChanges` method of a dataset or data table, or when you call the `Update` method of a [TableAdapter](#) or data adapter.

Changes are tracked two ways in each data row:

- Each data row contains information related to its [RowState](#) (for example, [Added](#), [Modified](#), [Deleted](#), or [Unchanged](#)).

- Each changed data row contains multiple versions of that row ([DataRowVersion](#)), the original version (before changes) and the current version (after changes). During the period when a change is pending (the time when you can respond to the [RowChanging](#) event), a third version — the proposed version— is available as well.

The [HasChanges](#) method of a dataset returns `true` if changes have been made in the dataset. After determining that changed rows exist, you can call the [GetChanges](#) method of a [DataSet](#) or [DataTable](#) to return a set of changed rows.

To determine if changes have been made to any rows

- Call the [HasChanges](#) method of a dataset to check for changed rows.

The following example shows how to check the return value from the [HasChanges](#) method to detect whether there are any changed rows in a dataset named `NorthwindDataSet1`:

```
if (northwindDataSet1.HasChanges())
{
    // Changed rows were detected, add appropriate code.
}
else
{
    // No changed rows were detected, add appropriate code.
}
```

```
If NorthwindDataSet1.HasChanges() Then
    ' Changed rows were detected, add appropriate code.
Else
    ' No changed rows were detected, add appropriate code.
End If
```

Determine the type of changes

You can also check to see what type of changes were made in a dataset by passing a value from the [DataRowState](#) enumeration to the [HasChanges](#) method.

To determine what type of changes have been made to a row

- Pass a [DataRowState](#) value to the [HasChanges](#) method.

The following example shows how to check a dataset named `NorthwindDataSet1` to determine if any new rows have been added to it:

```
if (northwindDataSet1.HasChanges(DataRowState.Added))
{
    // New rows have been added to the dataset, add appropriate code.
}
else
{
    // No new rows have been added to the dataset, add appropriate code.
}
```

```

If NorthwindDataSet1.HasChanges(DataRowState.Added) Then

    ' New rows have been added to the dataset, add appropriate code.
Else
    ' No new rows have been added to the dataset, add appropriate code.
End If

```

To locate rows that have errors

When working with individual columns and rows of data, you might encounter errors. You can check the `HasErrors` property to determine if errors exist in a [DataSet](#), [DataTable](#), or [DataRow](#).

1. Check the `HasErrors` property to see if there are any errors in the dataset.
2. If the `HasErrors` property is `true`, iterate through the collections of tables, and then the through the rows, to find the row with the error.

```

private void FindErrors()
{
    if (dataSet1.HasErrors)
    {
        foreach (DataTable table in dataSet1.Tables)
        {
            if (table.HasErrors)
            {
                foreach (DataRow row in table.Rows)
                {
                    if (row.HasErrors)
                    {
                        // Process error here.
                    }
                }
            }
        }
    }
}

```

```

Private Sub FindErrors()
    Dim table As Data.DataTable
    Dim row As Data.DataRow

    If DataSet1.HasErrors Then

        For Each table In DataSet1.Tables
            If table.HasErrors Then

                For Each row In table.Rows
                    If row.HasErrors Then

                        ' Process error here.
                    End If
                Next
            End If
        Next
    End If
End Sub

```

See also

- [Dataset tools in Visual Studio](#)

Validate data in datasets

8/5/2021 • 10 minutes to read • [Edit Online](#)

Validating data is the process of confirming that the values being entered into data objects conform to the constraints within a dataset's schema. The validation process also confirms that these values are following the rules that have been established for your application. It's a good practice to validate data prior to sending updates to the underlying database. This reduces errors as well as the potential number of round trips between an application and the database.

You can confirm that data that's being written to a dataset is valid by building validation checks into the dataset itself. The dataset can check the data no matter how the update is being performed — whether directly by controls in a form, within a component, or in some other way. Because the dataset is part of your application (unlike the database backend), it's a logical place to build application-specific validation.

The best place to add validation to your application is in the dataset's partial class file. In Visual Basic or Visual C#, open the **Dataset Designer** and double-click the column or table for which you want to create validation. This action automatically creates an [ColumnChanging](#) or [RowChanging](#) event handler.

Validate data

Validation within a dataset is accomplished in the following ways:

- By creating your own application-specific validation that can check values in an individual data column during changes. For more information, see [How to: Validate data during column changes](#).
- By creating your own application-specific validation that can check data to values while an entire data row is changing. For more information, see [How to: Validate data during row changes](#).
- By creating keys, unique constraints, and so on as part of the actual schema definition of the dataset.
- By setting the properties of the [DataColumn](#) object's, such as [MaxLength](#), [AllowDBNull](#), and [Unique](#).

Several events are raised by the [DataTable](#) object when a change is occurring in a record:

- The [ColumnChanging](#) and [ColumnChanged](#) events are raised during and after each change to an individual column. The [ColumnChanging](#) event is useful when you want to validate changes in specific columns. Information about the proposed change is passed as an argument with the event.
- The [RowChanging](#) and [RowChanged](#) events are raised during and after any change in a row. The [RowChanging](#) event is more general. It indicates that a change is occurring somewhere in the row, but you don't know which column has changed.

By default, each change to a column therefore raises four events. The first is the [ColumnChanging](#) and [ColumnChanged](#) events for the specific column that's being changed. Next are the [RowChanging](#) and [RowChanged](#) events. If multiple changes are being made to the row, the events will be raised for each change.

NOTE

The data row's [BeginEdit](#) method turns off the [RowChanging](#) and [RowChanged](#) events after each individual column change. In that case, the event is not raised until the [EndEdit](#) method has been called, when the [RowChanging](#) and [RowChanged](#) events are raised just once. For more information, see [Turn off constraints while filling a dataset](#).

The event you choose depends on how granular you want the validation to be. If it's important that you catch an

error immediately when a column changes, build validation by using the [ColumnChanging](#) event. Otherwise, use the [RowChanging](#) event, which might result in catching several errors at once. Additionally, if your data is structured so that the value of one column is validated based on the contents of another column, perform your validation during the [RowChanging](#) event.

When records are updated, the [DataTable](#) object raises events that you can respond to as changes are occurring and after changes are made.

If your application uses a typed dataset, you can create strongly typed event handlers. This adds four additional typed events for which you can create handlers: `dataTableColumnNameRowChanging`, `dataTableColumnNameRowChanged`, `dataTableColumnNameRowDeleting`, and `dataTableColumnNameRowDeleted`. These typed event handlers pass an argument that includes the column names of your table that make code easier to write and read.

Data update events

| EVENT | DESCRIPTION |
|--------------------------------|---|
| ColumnChanging | The value in a column is being changed. The event passes the row and column to you, along with the proposed new value. |
| ColumnChanged | The value in a column has been changed. The event passes the row and column to you, along with the proposed value. |
| RowChanging | <p>The changes that were made to a DataRow object are about to be committed back into the dataset. If you have not called the BeginEdit method, the RowChanging event is raised for each change to a column immediately after the ColumnChanging event has been raised. If you called BeginEdit before making changes, the RowChanging event is raised only when you call the EndEdit method.</p> <p>The event passes the row to you, along with a value indicating what type of action (change, insert, and so on) is being performed.</p> |
| RowChanged | A row has been changed. The event passes the row to you, along with a value indicating what type of action (change, insert, and so on) is being performed. |
| RowDeleting | A row is being deleted. The event passes the row to you, along with a value indicating what type of action (delete) is being performed. |
| RowDeleted | A row has been deleted. The event passes the row to you, along with a value indicating what type of action (delete) is being performed. |

The [ColumnChanging](#), [RowChanging](#), and [RowDeleting](#) events are raised during the update process. You can use these events to validate data or perform other types of processing. Because the update is in process during these events, you can cancel it by throwing an exception, which prevents the update from finishing.

The [ColumnChanged](#), [RowChanged](#) and [RowDeleted](#) events are notification events that are raised when the update has finished successfully. These events are useful when you want to take further action based on a successful update.

Validate data during column changes

NOTE

The **Dataset Designer** creates a partial class in which validation logic can be added to a dataset. The designer-generated dataset doesn't delete or change any code in the partial class.

You can validate data when the value in a data column changes by responding to the [ColumnChanging](#) event. When raised, this event passes an event argument ([ProposedValue](#)) that contains the value that's being proposed for the current column. Based on the contents of `e.ProposedValue`, you can:

- Accept the proposed value by doing nothing.
- Reject the proposed value by setting the column error ([SetColumnError](#)) from within the column-changing event handler.
- Optionally use an [ErrorProvider](#) control to display an error message to the user. For more information, see [ErrorProvider component](#).

Validation can also be performed during the [RowChanging](#) event.

Validate data during row changes

You can write code to verify that each column you want to validate contains data that meets the requirements of your application. Do this by setting the column to indicate that it contains an error if a proposed value is unacceptable. The following examples set a column error when the `Quantity` column is 0 or less. The row-changing event handlers should resemble the following examples.

To validate data when a row changes (Visual Basic)

1. Open your dataset in the **Dataset Designer**. For more information, see [Walkthrough: Creating a Dataset in the Dataset Designer](#).
2. Double-click the title bar of the table you want to validate. This action automatically creates the [RowChanging](#) event handler of the [DataTable](#) in the dataset's partial-class file.

TIP

Double-click to the left of the table name to create the row-changing event handler. If you double-click the table name, you can edit it.

```
Private Sub Order_DetailsDataTable_Order_DetailsRowChanging(  
    ByVal sender As System.Object,  
    ByVal e As Order_DetailsRowChangeEvent  
) Handles Me.Order_DetailsRowChanging  
  
    If CType(e.Row.Quantity, Short) <= 0 Then  
        e.Row.SetColumnError("Quantity", "Quantity must be greater than 0")  
    Else  
        e.Row.SetColumnError("Quantity", "")  
    End If  
End Sub
```

To validate data when a row changes (C#)

1. Open your dataset in the **Dataset Designer**. For more information, see [Walkthrough: Creating a dataset in the Dataset Designer](#).
2. Double-click the title bar of the table you want to validate. This action creates a partial-class file for the

NOTE

The **Dataset Designer** does not automatically create an event handler for the [RowChanging](#) event. You have to create a method to handle the [RowChanging](#) event, and run code to hook up the event in the table's initialization method.

3. Copy the following code into the partial class:

```
public override void EndInit()
{
    base.EndInit();
    Order_DetailsRowChanging += TestRowChangeEvent;
}

public void TestRowChangeEvent(object sender, Order_DetailsRowChangeEvent e)
{
    if ((short)e.Row.Quantity <= 0)
    {
        e.Row.SetColumnError("Quantity", "Quantity must be greater than 0");
    }
    else
    {
        e.Row.SetColumnError("Quantity", "");
    }
}
```

To retrieve changed rows

Each row in a data table has a [RowState](#) property that keeps track of the current state of that row by using the values in the [DataRowState](#) enumeration. You can return changed rows from a dataset or data table by calling the `GetChanges` method of a [DataSet](#) or [DataTable](#). You can verify that changes exist prior to calling `GetChanges` by calling the [HasChanges](#) method of a dataset.

NOTE

After you commit changes to a dataset or data table (by calling the [AcceptChanges](#) method), the `GetChanges` method returns no data. If your application needs to process changed rows, you must process the changes before calling the `AcceptChanges` method.

Calling the [GetChanges](#) method of a dataset or data table returns a new dataset or data table that contains only records that have been changed. If you want to get specific records — for example, only new records or only modified records — you can pass a value from the [DataRowState](#) enumeration as a parameter to the `GetChanges` method.

Use the [DataRowVersion](#) enumeration to access the different versions of a row (for example, the original values that were in a row prior to processing it).

To get all changed records from a dataset

- Call the [GetChanges](#) method of a dataset.

The following example creates a new dataset called `changedRecords` and populates it with all the changed records from another dataset called `dataSet1`.


```
DataSet changedRecords = dataSet1.GetChanges();
```

```
Dim changedRecords As DataSet = DataSet1.GetChanges()
```

To get all changed records from a data table

- Call the [GetChanges](#) method of a DataTable.

The following example creates a new data table called `changedRecordsTable` and populates it with all the changed records from another data table called `dataTable1`.

```
DataTable changedRecordsTable = dataTable1.GetChanges();
```

```
Dim changedRecordsTable As DataTable = dataTable1.GetChanges()
```

To get all records that have a specific row state

- Call the `GetChanges` method of a dataset or data table and pass a [DataRowState](#) enumeration value as an argument.

The following example shows how to create a new dataset called `addedRecords` and populate it only with records that have been added to the `dataSet1` dataset.

```
DataSet addedRecords = dataSet1.GetChanges(DataRowState.Added);
```

```
Dim addedRecords As DataSet = DataSet1.GetChanges(DataRowState.Added)
```

The following example shows how to return all records that were recently added to the `Customers` table:

```
private NorthwindDataSet.CustomersDataTable GetNewRecords()
{
    return (NorthwindDataSet.CustomersDataTable)
        northwindDataSet1.Customers.GetChanges(DataRowState.Added);
}
```

```
Private Function GetNewRecords() As NorthwindDataSet.CustomersDataTable

    Return CType(NorthwindDataSet1.Customers.GetChanges(Data.DataRowState.Added),
        NorthwindDataSet.CustomersDataTable)
End Function
```

Access the original version of a DataRow

When changes are made to data rows, the dataset retains both the original ([Original](#)) and new ([Current](#)) versions of the row. For example, before calling the `AcceptChanges` method, your application can access the different versions of a record (as defined in the [DataRowVersion](#) enumeration) and process the changes accordingly.

NOTE

Different versions of a row exist only after it has been edited and before it the `AcceptChanges` method has been called. After the `AcceptChanges` method has been called, the current and original versions are the same.

Passing the `DataRowVersion` value along with the column index (or column name as a string) returns the value from that column's particular row version. The changed column is identified during the `ColumnChanging` and `ColumnChanged` events. This is a good time to inspect the different row versions for validation purposes. However, if you have temporarily suspended constraints, those events won't be raised, and you will need to programmatically identify which columns have changed. You can do this by iterating through the `Columns` collection and comparing the different `DataRowVersion` values.

To get the original version of a record

- Access the value of a column by passing in the `DataRowVersion` of the row you want to return.

The following example shows how to use a `DataRowVersion` value to get the original value of a

`CompanyName` field in a `DataRow`:

```
string originalCompanyName;  
originalCompanyName = northwindDataSet1.Customers[0]  
    ["CompanyName", DataRowVersion.Original].ToString();
```

```
Dim originalCompanyName = NorthwindDataSet1.Customers(0)(  
    "CompanyName", DataRowVersion.Original).ToString()
```

Access the current version of a DataRow

To get the current version of a record

- Access the value of a column, and then add a parameter to the index that indicates which version of a row you want to return.

The following example shows how to use a `DataRowVersion` value to get the current value of a

`CompanyName` field in a `DataRow`:

```
string currentCompanyName;  
currentCompanyName = northwindDataSet1.Customers[0]  
    ["CompanyName", DataRowVersion.Current].ToString();
```

```
Dim currentCompanyName = NorthwindDataSet1.Customers(0)(  
    "CompanyName", DataRowVersion.Current).ToString()
```

See also

- [Dataset tools in Visual Studio](#)
- [How to: Validate data in the Windows Forms DataGridView control](#)
- [How to: Display error icons for form validation with the Windows Forms ErrorProvider component](#)

Save data back to the database

8/5/2021 • 18 minutes to read • [Edit Online](#)

The dataset is an in-memory copy of data. If you modify that data, it's a good practice to save those changes back to the database. You do this in one of three ways:

- By calling one of the `Update` methods of a `TableAdapter`
- By calling one of `DBDirect` methods of the `TableAdapter`
- By calling the `UpdateAll` method on the `TableAdapterManager` that Visual Studio generates for you when the dataset contains tables that are related to other tables in the dataset

When you data bind dataset tables to controls on a Windows Form or XAML page, the data binding architecture does all the work for you.

If you're familiar with `TableAdapters`, you can jump directly to one of these topics:

| TOPIC | DESCRIPTION |
|---|---|
| Insert new records into a database | How to perform updates and inserts using <code>TableAdapters</code> or <code>Command</code> objects |
| Update data by using a <code>TableAdapter</code> | How to perform updates with <code>TableAdapters</code> |
| Hierarchical update | How to perform updates from a dataset with two or more related tables |
| Handle a concurrency exception | How to handle exceptions when two users attempt to change the same data in a database at the same time |
| How to: Save data by using a transaction | How to save data in a transaction using the <code>System.Transactions</code> namespace and a <code>TransactionScope</code> object |
| Save data in a transaction | Walkthrough that creates a Windows Forms application to demonstrate saving data to a database inside a transaction |
| Save data to a database (multiple tables) | How to edit records and save changes in multiple tables back to the database |
| Save data from an object to a database | How to pass data from an object that is not in a dataset to a database by using a <code>TableAdapter DbDirect</code> method |
| Save data with the <code>TableAdapter DbDirect</code> methods | How to use the <code>TableAdapter</code> to send SQL queries directly to the database |
| Save a dataset as XML | How to save a dataset to an XML document |

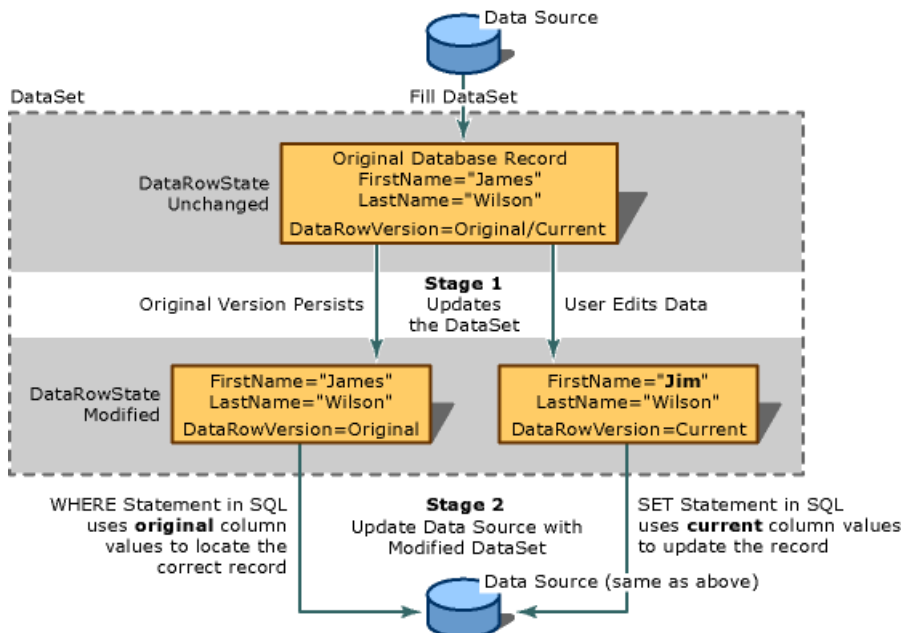
Two-stage updates

Updating a data source is a two-step process. The first step is to update the dataset with new records, changed records, or deleted records. If your application never sends those changes back to the data source, then you're

finished with the update.

If you do send the changes back to the database, a second step is required. If you aren't using data-bound controls, you have to manually call the `Update` method of the same `TableAdapter` (or data adapter) that you used to populate the dataset. However, you can also use different adapters, for example, to move data from one data source to another or to update multiple data sources. If you aren't using data binding, and are saving changes for related tables, you have to manually instantiate a variable of the auto-generated

`TableAdapterManager` class, and then call its `UpdateAll` method.



A dataset contains collections of tables, which contain a collections of rows. If you intend to update an underlying data source later, you must use the methods on the `DataTable.DataRowCollection` property when adding or removing rows. Those methods perform the change tracking that's needed for updating the data source. If you call the `RemoveAt` collection on the Rows property, the deletion won't be communicated back to the database.

Merge datasets

You can update the contents of a dataset by *merging* it with another dataset. This involves copying the contents of a *source* dataset into the calling dataset (referred to as the *target* dataset). When you merge datasets, new records in the source dataset are added to the target dataset. Additionally, extra columns in the source dataset are added to the target dataset. Merging datasets is useful when you have a local dataset and you get a second dataset from another application. It's also useful when you get a second dataset from a component such as an XML web service, or when you need to integrate data from multiple datasets.

When merging datasets, you can pass a Boolean argument (`preserveChanges`) that tells the `Merge` method whether to retain existing modifications in the target dataset. Because datasets maintain multiple versions of records, it's important to keep in mind that more than one version of the records is being merged. The following table shows how a record in two datasets is merged:

| Datarowversion | Target Dataset | Source Dataset |
|----------------|----------------|-----------------|
| Original | James Wilson | James C. Wilson |
| Current | Jim Wilson | James C. Wilson |

Calling the `Merge` method on the previous table with `preserveChanges=false targetDataset.Merge(sourceDataset)` results in the following data:

| DATAROWVERSION | TARGET DATASET | SOURCE DATASET |
|----------------|-----------------|-----------------|
| Original | James C. Wilson | James C. Wilson |
| Current | James C. Wilson | James C. Wilson |

Calling the [Merge](#) method with `preserveChanges = true` `targetDataset.Merge(sourceDataset, true)` results in the following data:

| DATAROWVERSION | TARGET DATASET | SOURCE DATASET |
|----------------|-----------------|-----------------|
| Original | James C. Wilson | James C. Wilson |
| Current | Jim Wilson | James C. Wilson |

Caution

In the `preserveChanges = true` scenario, if the [RejectChanges](#) method is called on a record in the target dataset, then it reverts to the original data from the *source* dataset. This means that if you try to update the original data source with the target dataset, it might not be able to find the original row to update. You can prevent a concurrency violation by filling another dataset with the updated records from the data source and then performing a merge to prevent a concurrency violation. (A concurrency violation occurs when another user modifies a record in the data source after the dataset has been filled.)

Update constraints

To make changes to an existing data row, add or update data in the individual columns. If the dataset contains constraints (such as foreign keys or non-nullable constraints), it's possible that the record can temporarily be in an error state as you update it. That is, it can be in an error state after you finish updating one column but before you get to the next one.

To prevent premature constraint violations you can temporarily suspend update constraints. This serves two purposes:

- It prevents an error from being thrown after you've finished updating one column but haven't started updating another.
- It prevents certain update events from being raised (events that are often used for validation).

NOTE

In Windows Forms, the data binding architecture that's built into the datagrid suspends constraint checking until focus moves out of a row, and you do not have to explicitly call the [BeginEdit](#), [EndEdit](#), or [CancelEdit](#) methods.

Constraints are automatically disabled when the [Merge](#) method is invoked on a dataset. When the merge is complete, if there are any constraints on the dataset that cannot be enabled, a [ConstraintException](#) is thrown. In this situation, the [EnforceConstraints](#) property is set to `false`, and all constraint violations must be resolved before resetting the [EnforceConstraints](#) property to `true`.

After you complete an update, you can re-enable constraint checking, which also re-enables update events and raises them.

For more information about suspending events, see [Turn off constraints while filling a dataset](#).

Dataset update errors

When you update a record in a dataset, there is the possibility of an error. For example, you might inadvertently write data of the wrong type to a column, or data that's too long, or data that has some other integrity problem. Or, you might have application-specific validation checks that can raise custom errors during any stage of an update event. For more information, see [Validate data in datasets](#).

Maintain information about changes

Information about the changes in a dataset is maintained in two ways: by flagging rows that indicate that they have changed ([RowState](#)), and by keeping multiple copies of a record ([DataRowVersion](#)). By using this information, processes can determine what has changed in the dataset and can send appropriate updates to the data source.

RowState property

The [RowState](#) property of a [DataRow](#) object is a value that provides information about the status of a particular row of data.

The following table details the possible values of the [DataRowState](#) enumeration:

| Datarowstate value | Description |
|---------------------------|---|
| Added | The row has been added as an item to a DataRowCollection . (A row in this state does not have a corresponding original version since it did not exist when the last AcceptChanges method was called). |
| Deleted | The row was deleted using the Delete of a DataRow object. |
| Detached | The row has been created but is not part of any DataRowCollection . A DataRow object is in this state immediately after it has been created, before it has been added to a collection, and after it has been removed from a collection. |
| Modified | A column value in the row has changed in some way. |
| Unchanged | The row has not changed since AcceptChanges was last called. |

DataRowVersion enumeration

Datasets maintain multiple versions of records. The [DataRowVersion](#) fields are used when retrieving the value found in a [DataRow](#) using the [Item\[\]](#) property or the [GetChildRows](#) method of the [DataRow](#) object.

The following table details the possible values of the [DataRowVersion](#) enumeration:

| Datarowversion value | Description |
|-------------------------|---|
| Current | The current version of a record contains all modifications that have been performed on the record since the last time AcceptChanges was called. If the row has been deleted, there is no current version. |
| Default | The default value of a record, as defined by the dataset schema or data source. |

| DataRowVersion Value | Description |
|----------------------|---|
| Original | The original version of a record is a copy of the record as it was the last time changes were committed in the dataset. In practical terms, this is typically the version of a record as read from a data source. |
| Proposed | The proposed version of a record that is available temporarily while you are in the middle of an update — that is, between the time you called the BeginEdit method and the EndEdit method. You typically access the proposed version of a record in a handler for an event such as RowChanging . Invoking the CancelEdit method reverses the changes and deletes the proposed version of the data row. |

The original and current versions are useful when update information is transmitted to a data source. Typically, when an update is sent to the data source, the new information for the database is in the current version of a record. Information from the original version is used to locate the record to update.

For example, in a case where the primary key of a record is changed, you need a way to locate the correct record in the data source in order to update the changes. If no original version existed, then the record would most likely be appended to the data source, resulting not only in an extra unwanted record, but in one record that is inaccurate and out of date. The two versions are also used in concurrency control. You can compare the original version against a record in the data source to determine if the record has changed since it was loaded into the dataset.

The proposed version is useful when you need to perform validation before actually committing the changes to the dataset.

Even if records have changed, there are not always original or current versions of that row. When you insert a new row into the table, there is no original version, only a current version. Similarly, if you delete a row by calling the table's `Delete` method, there is an original version, but no current version.

You can test to see if a specific version of a record exists by querying a data row's [HasVersion](#) method. You can access either version of a record by passing a [DataRowVersion](#) enumeration value as an optional argument when you request the value of a column.

Get changed records

It's a common practice not to update every record in a dataset. For example, a user might be working with a Windows Forms [DataGridView](#) control that displays many records. However, the user might update only a few records, delete one, and insert a new one. Datasets and data tables provide a method (`GetChanges`) for returning only the rows that have been modified.

You can create subsets of changed records using the `GetChanges` method of either the data table ([GetChanges](#)) or of the dataset ([GetChanges](#)) itself. If you call the method for the data table, it returns a copy of the table with only the changed records. Similarly, if you call the method on the dataset, you get a new dataset with only changed records in it.

`GetChanges` by itself returns all changed records. In contrast, by passing the desired [DataRowState](#) as a parameter to the `GetChanges` method, you can specify what subset of changed records you want: newly added records, records that are marked for deletion, detached records, or modified records.

Getting a subset of changed records is useful when you want to send records to another component for processing. Instead of sending the entire dataset, you can reduce the overhead of communicating with the other component by getting only the records that the component needs.

Commit changes in the dataset

As changes are made in the dataset, the [RowState](#) property of changed rows is set. The original and current versions of records are established, maintained, and made available to you by the [RowVersion](#) property. The metadata that's stored in the properties of these changed rows is necessary for sending the correct updates to the data source.

If the changes reflect the current state of the data source, you no longer need to maintain this information. Typically, there are two times when the dataset and its source are in sync:

- Immediately after you have loaded information into the dataset, such as when you read data from the source.
- After sending changes from the dataset to the data source (but not before, because you would lose the change information that's required to send changes to the database).

You can commit the pending changes to the dataset by calling the [AcceptChanges](#) method. Typically, [AcceptChanges](#) is called at the following times:

- After you load the dataset. If you load a dataset by calling a `TableAdapter`'s `Fill` method, then the adapter automatically commits changes for you. However, if you load a dataset by merging another dataset into it, then you have to commit the changes manually.

NOTE

You can prevent the adapter from automatically committing the changes when you call the `Fill` method by setting the `AcceptChangesDuringFill` property of the adapter to `false`. If it's set to `false`, then the [RowState](#) of each row that's inserted during the fill is set to [Added](#).

- After you send dataset changes to another process, such as an XML web service.

Caution

Committing the change this way erases any change information. Do not commit changes until after you finish performing operations that require your application to know what changes have been made in the dataset.

This method accomplishes the following:

- Writes the [Current](#) version of a record into its [Original](#) version and overwrites the original version.
- Removes any row where the [RowState](#) property is set to [Deleted](#).
- Sets the [RowState](#) property of a record to [Unchanged](#).

The [AcceptChanges](#) method is available at three levels. You can call it on a [DataRow](#) object to commits changes for just that row. You can also call it on a [DataTable](#) object to commit all rows in a table. Finally, you can call it on the [DataSet](#) object to commit all pending changes in all records of all tables of the dataset.

The following table describes which changes are committed based on what object the method is called on:

| METHOD | RESULT |
|---|--|
| System.Data.DataRow.AcceptChanges | Changes are committed only on the specific row. |
| System.Data.DataTable.AcceptChanges | Changes are committed on all rows in the specific table. |

| METHOD | RESULT |
|---|---|
| System.Data.DataSet.AcceptChanges | Changes are committed on all rows in all tables of the dataset. |

NOTE

If you load a dataset by calling a TableAdapter's `Fill` method, you don't have to explicitly accept changes. By default, the `Fill` method calls the `AcceptChanges` method after it finishes populating the data table.

A related method, [RejectChanges](#), undoes the effect of changes by copying the [Original](#) version back into the [Current](#) version of records. It also sets the [RowState](#) of each record back to [Unchanged](#).

Data validation

In order to verify that the data in your application meets the requirements of the processes that it is passed to, you often have to add validation. This might involve checking that a user's entry in a form is correct, validating data that's sent to your application by another application, or even checking that information that's calculated within your component falls within the constraints of your data source and application requirements.

You can validate data in several ways:

- In the business layer, by adding code to your application to validate data. The dataset is one place you can do this. The dataset provides some of the advantages of back-end validation — such as the ability to validate changes as column and row values are changing. For more information, see [Validate data in datasets](#).
- In the presentation layer, by adding validation to forms. For more information, see [User input validation in Windows Forms](#).
- In the data back end, by sending data to the data source — for example, the database — and allowing it to accept or reject the data. If you are working with a database that has sophisticated facilities for validating data and providing error information, this can be a practical approach because you can validate the data no matter where it comes from. However, this approach might not accommodate application-specific validation requirements. Additionally, having the data source validate data can result in numerous round trips to the data source, depending on how your application facilitates the resolution of validation errors raised by the back end.

IMPORTANT

When using data commands with a [CommandType](#) property that's set to [Text](#), carefully check information that is sent from a client before passing it to your database. Malicious users might try to send (inject) modified or additional SQL statements in an effort to gain unauthorized access or damage the database. Before you transfer user input to a database, always verify that the information is valid. It's a best practice to always use parameterized queries or stored procedures when possible.

Transmit updates to the data source

After changes have been made in a dataset, you can transmit the changes to a data source. Most commonly, you do this by calling the `Update` method of a TableAdapter (or data adapter). The method loops through each record in a data table, determines what type of update is required (update, insert, or delete), if any, and then runs the appropriate command.

As an illustration of how updates are made, suppose your application uses a dataset that contains a single data

table. The application fetches two rows from the database. After the retrieval, the in-memory data table looks like this:

| (RowState) | CustomerID | Name | Status |
|-------------|------------|----------------|---------|
| (Unchanged) | c200 | Robert Lyon | Good |
| (Unchanged) | c400 | Nancy Buchanan | Pending |

Your application changes Nancy Buchanan's status to "Preferred." As a result of this change, the value of the `RowState` property for that row changes from `Unchanged` to `Modified`. The value of the `RowState` property for the first row remains `Unchanged`. The data table now looks like this:

| (RowState) | CustomerID | Name | Status |
|-------------|------------|----------------|-----------|
| (Unchanged) | c200 | Robert Lyon | Good |
| (Modified) | c400 | Nancy Buchanan | Preferred |

Your application now calls the `Update` method to transmit the dataset to the database. The method inspects each row in turn. For the first row, the method transmits no SQL statement to the database because that row has not changed since it was originally fetched from the database.

For the second row, however, the `Update` method automatically invokes the correct data command and transmits it to the database. The specific syntax of the SQL statement depends on the dialect of SQL that's supported by the underlying data store. But, the following general traits of the transmitted SQL statement are noteworthy:

- The transmitted SQL statement is an UPDATE statement. The adapter knows to use an UPDATE statement because the value of the `RowState` property is `Modified`.
- The transmitted SQL statement includes a WHERE clause indicating that the target of the UPDATE statement is the row where `CustomerID = 'c400'`. This part of the SELECT statement distinguishes the target row from all others because the `CustomerID` is the primary key of the target table. The information for the WHERE clause is derived from the original version of the record (`DataRowVersion.Original`), in case the values that are required to identify the row have changed.
- The transmitted SQL statement includes the SET clause, to set the new values of the modified columns.

NOTE

If the `TableAdapter's UpdateCommand` property has been set to the name of a stored procedure, the adapter does not construct an SQL statement. Instead, it invokes the stored procedure with the appropriate parameters passed in.

Pass parameters

You usually use parameters to pass the values for records that are going to be updated in the database. When the `TableAdapter's Update` method runs an UPDATE statement, it needs to fill in the parameter values. It gets these values from the `Parameters` collection for the appropriate data command — in this case, the `UpdateCommand` object in the `TableAdapter`.

If you've used the Visual Studio tools to generate a data adapter, the `UpdateCommand` object contains a collection of parameters that correspond to each parameter placeholder in the statement.

The `System.Data.SqlClient.SqlParameter.SourceColumn` property of each parameter points to a column in the data table. For example, the `SourceColumn` property for the `au_id` and `Original_au_id` parameters is set to whatever column in the data table contains the author id. When the adapter's `Update` method runs, it reads the

author id column from the record that's being updated and fills the values into the statement.

In an UPDATE statement, you need to specify both the new values (those that will be written to the record) as well as the old values (so that the record can be located in the database). There are, therefore, two parameters for each value: one for the SET clause and a different one for the WHERE clause. Both parameters read data from the record that's being updated, but they get different versions of the column value based on the parameter's [SourceVersion](#) property. The parameter for the SET clause gets the current version, and the parameter for the WHERE clause gets the original version.

NOTE

You can also set values in the `Parameters` collection yourself in code, which you would typically do in an event handler for the data adapter's [RowChanging](#) event.

See also

- [Dataset tools in Visual Studio](#)
- [Create and configure TableAdapters](#)
- [Update data by using a TableAdapter](#)
- [Bind controls to data in Visual Studio](#)
- [Validate data](#)
- [How to: Add, modify, and delete entities \(WCF data services\)](#)

Insert new records into a database

8/5/2021 • 3 minutes to read • [Edit Online](#)

To insert new records into a database, you can use the `TableAdapter.Update` method, or one of the `TableAdapter`'s `DBDirect` methods (specifically the `TableAdapter.Insert` method). For more information, see [TableAdapter](#).

If your application doesn't use `TableAdapters`, you can use command objects (for example, [SqlCommand](#)) to insert new records in your database.

If your application uses datasets to store data, use the `TableAdapter.Update` method. The `Update` method sends all changes (updates, inserts, and deletes) to the database.

If your application uses objects to store data, or if you want finer control over creating new records in the database, use the `TableAdapter.Insert` method.

If your `TableAdapter` doesn't have an `Insert` method, it means that either the `TableAdapter` is configured to use stored procedures or its `GenerateDBDirectMethods` property is set to `false`. Try setting the `TableAdapter`'s `GenerateDBDirectMethods` property to `true` from within the **Dataset Designer**, and then save the dataset. This will regenerate the `TableAdapter`. If the `TableAdapter` still doesn't have an `Insert` method, the table probably does not provide enough schema information to distinguish between individual rows (for example, there might be no primary key set on the table).

Insert new records by using TableAdapters

`TableAdapters` provide different ways to insert new records into a database, depending on the requirements of your application.

If your application uses datasets to store data, you can simply add new records to the desired [DataTable](#) in the dataset, and then call the `TableAdapter.Update` method. The `TableAdapter.Update` method sends any changes in the [DataTable](#) to the database (including modified and deleted records).

To insert new records into a database by using the TableAdapter.Update method

1. Add new records to the desired [DataTable](#) by creating a new [DataRow](#) and adding it to the [Rows](#) collection.
2. After the new rows are added to the [DataTable](#), call the `TableAdapter.Update` method. You can control the amount of data to update by passing in either an entire [DataSet](#), a [DataTable](#), an array of [DataRows](#), or a single [DataRow](#).

The following code shows how to add a new record to a [DataTable](#) and then call the `TableAdapter.Update` method to save the new row to the database. (This example uses the `Region` table in the Northwind database.)

```
' Create a new row.
Dim newRegionRow As NorthwindDataSet.RegionRow
newRegionRow = Me.NorthwindDataSet._Region.NewRegionRow()
newRegionRow.RegionID = 5
newRegionRow.RegionDescription = "NorthWestern"

' Add the row to the Region table
Me.NorthwindDataSet._Region.Rows.Add(newRegionRow)

' Save the new row to the database
Me.RegionTableAdapter.Update(Me.NorthwindDataSet._Region)
```

```
// Create a new row.
NorthwindDataSet.RegionRow newRegionRow;
newRegionRow = northwindDataSet.Region.NewRegionRow();
newRegionRow.RegionID = 5;
newRegionRow.RegionDescription = "NorthWestern";

// Add the row to the Region table
this.northwindDataSet.Region.Rows.Add(newRegionRow);

// Save the new row to the database
this.regionTableAdapter.Update(this.northwindDataSet.Region);
```

To insert new records into a database by using the `TableAdapter.Insert` method

If your application uses objects to store data, you can use the `TableAdapter.Insert` method to create new rows directly in the database. The `Insert` method accepts the individual values for each column as parameters. Calling the method inserts a new record into the database with the parameter values passed in.

- Call the `TableAdapter`'s `Insert` method, passing in the values for each column as parameters.

The following procedure demonstrates using the `TableAdapter.Insert` method to insert rows. This example inserts data into the `Region` table in the Northwind database.

NOTE

If you do not have an instance available, instantiate the `TableAdapter` you want to use.

```
Dim regionTableAdapter As New NorthwindDataSetTableAdapters.RegionTableAdapter

regionTableAdapter.Insert(5, "NorthWestern")
```

```
NorthwindDataSetTableAdapters.RegionTableAdapter regionTableAdapter =
    new NorthwindDataSetTableAdapters.RegionTableAdapter();

regionTableAdapter.Insert(5, "NorthWestern");
```

Insert new records by using command objects

You can insert new records directly into a database using command objects.

To insert new records into a database by using command objects

- Create a new command object, and then set its `Connection`, `CommandType`, and `CommandText` properties.

The following example demonstrates inserting records into a database using command object. It inserts data

into the `Region` table in the Northwind database.

```
Dim sqlConnection1 As New System.Data.SqlClient.SqlConnection("YOUR CONNECTION STRING")

Dim cmd As New System.Data.SqlClient.SqlCommand
cmd.CommandType = System.Data.CommandType.Text
cmd.CommandText = "INSERT Region (RegionID, RegionDescription) VALUES (5, 'NorthWestern')"
cmd.Connection = sqlConnection1

sqlConnection1.Open()
cmd.ExecuteNonQuery()
sqlConnection1.Close()
```

```
System.Data.SqlClient.SqlConnection sqlConnection1 =
    new System.Data.SqlClient.SqlConnection("YOUR CONNECTION STRING");

System.Data.SqlClient.SqlCommand cmd = new System.Data.SqlClient.SqlCommand();
cmd.CommandType = System.Data.CommandType.Text;
cmd.CommandText = "INSERT Region (RegionID, RegionDescription) VALUES (5, 'NorthWestern')";
cmd.Connection = sqlConnection1;

sqlConnection1.Open();
cmd.ExecuteNonQuery();
sqlConnection1.Close();
```

.NET security

You must have access to the database you are trying to connect to, as well as permission to perform inserts into the desired table.

See also

- [Save data back to the database](#)

Update data by using a TableAdapter

8/5/2021 • 2 minutes to read • [Edit Online](#)

After the data in your dataset has been modified and validated, you can send the updated data back to a database by calling the `Update` method of a `TableAdapter`. The `Update` method updates a single data table and runs the correct command (INSERT, UPDATE, or DELETE) based on the `RowState` of each data row in the table. When a dataset has related tables, Visual Studio generates a `TableAdapterManager` class that you use to do the updates. The `TableAdapterManager` class ensures that updates are made in the correct order based on the foreign-key constraints that are defined in the database. When you use data-bound controls, the databinding architecture creates a member variable of the `TableAdapterManager` class called `tableAdapterManager`.

NOTE

When you try to update a data source with the contents of a dataset, you can get errors. To avoid errors, we recommend that you put the code that calls the adapter's `Update` method inside a `try / catch` block.

The exact procedure for updating a data source can vary depending on business needs, but includes the following steps:

1. Call the adapter's `Update` method in a `try / catch` block.
2. If an exception is caught, locate the data row that caused the error.
3. Reconcile the problem in the data row (programmatically if you can, or by presenting the invalid row to the user for modification), and then try the update again ([HasErrors](#), [GetErrors](#)).

Save data to a database

Call the `Update` method of a `TableAdapter`. Pass the name of the data table that contains the values to be written to the database.

To update a database by using a TableAdapter

- Enclose the `TableAdapter`'s `Update` method in a `try / catch` block. The following example shows how to update the contents of the `Customers` table in `NorthwindDataSet` from within a `try / catch` block.

```
try
{
    this.Validate();
    this.customersBindingSource.EndEdit();
    this.customersTableAdapter.Update(this.northwindDataSet.Customers);
    MessageBox.Show("Update successful");
}
catch (System.Exception ex)
{
    MessageBox.Show("Update failed");
}
```

```
Try
    Me.Validate()
    Me.CustomersBindingSource.EndEdit()
    Me.CustomersTableAdapter.Update(Me.NorthwindDataSet.Customers)
    MsgBox("Update successful")

Catch ex As Exception
    MsgBox("Update failed")
End Try
```

See also

- [Save data back to the database](#)

Hierarchical update

8/5/2021 • 8 minutes to read • [Edit Online](#)

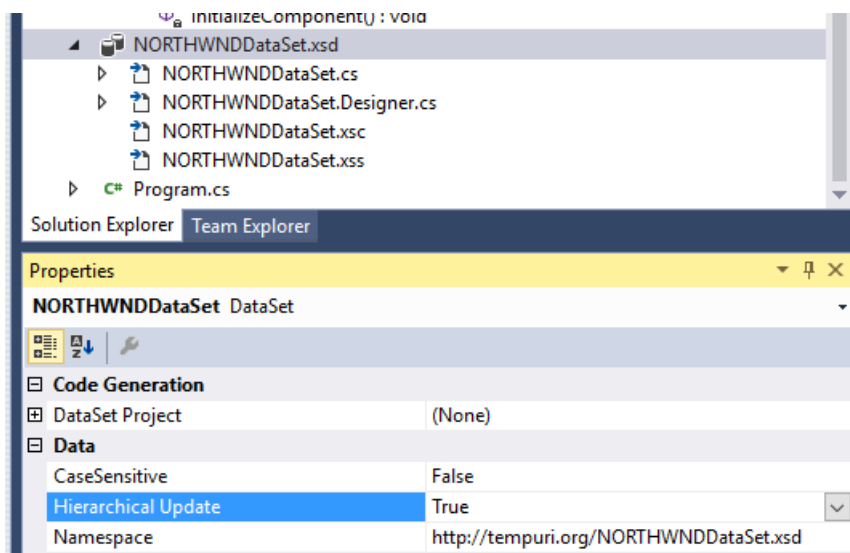
Hierarchical update refers to the process of saving updated data (from a dataset with two or more related tables) back to a database while maintaining referential integrity rules. *Referential integrity* refers to the consistency rules provided by the constraints in a database that control the behavior of inserting, updating, and deleting related records. For example, it's referential integrity that enforces the creation of a customer record before allowing orders to be created for that customer. For more information about relationships in datasets, see [Relationships in datasets](#).

The hierarchical update feature uses a `TableAdapterManager` to manage the `TableAdapter`s in a typed dataset. The `TableAdapterManager` component is a Visual Studio-generated class, not a .NET type. When you drag a table from the **Data Sources** window to a Windows Form or WPF page, Visual Studio adds a variable of type `TableAdapterManager` to the form or page, and you see it in the designer in the component tray. For detailed information about the `TableAdapterManager` class, see the `TableAdapterManager` Reference section of [TableAdapters](#).

By default, a dataset treats related tables as "relations only," which means that it doesn't enforce foreign key constraints. You can modify that setting at design time by using the **Dataset Designer**. Select the relation line between two tables to bring up the **Relation** dialog box. The changes you make here will determine how the `TableAdapterManager` behaves when it send the changes in the related tables back to the database.

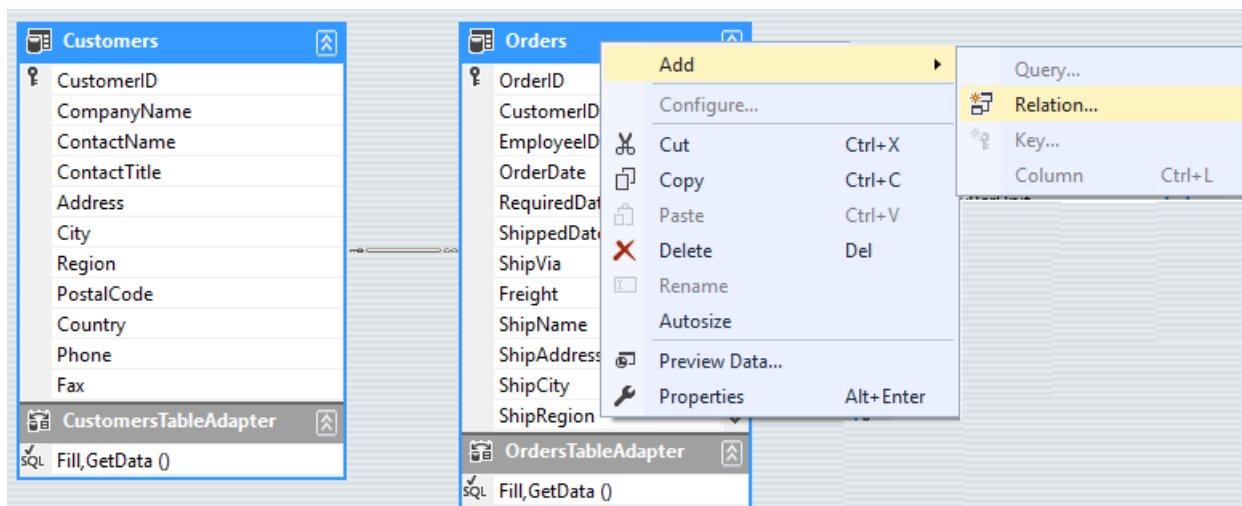
Enable hierarchical update in a dataset

By default, hierarchical update is enabled for all new datasets that are added or created in a project. Turn hierarchical update on or off by setting the **Hierarchical Update** property of a typed dataset in The dataset to **True** or **False**:



Create a new relation between tables

To create a new relation between two tables, in the Dataset Designer, select the title bar of each table, then right-click and select **Add relation**.



Understand foreign-key constraints, cascading updates, and deletes

It's important to understand how foreign-key constraints and cascading behavior in the database are created in the generated dataset code.

By default, the data tables in a dataset are generated with relationships ([DataRelation](#)) that match the relationships in the database. However, the relationship in the dataset is not generated as a foreign-key constraint. The [DataRelation](#) is configured as **Relation Only** without [UpdateRule](#) or [DeleteRule](#) in effect.

By default, cascading updates and cascading deletes are turned off even if the database relationship is set with cascading updates and/or cascading deletes turned on. For example, creating a new customer and a new order and then trying to save the data can cause a conflict with the foreign-key constraints that are defined in the database. For more information, see [Turn off constraints while filling a dataset](#).

Set the order to perform updates

Setting the order to perform updates sets the order of the individual inserts, updates, and deletes that are required to save all the modified data in all tables of a dataset. When hierarchical update is enabled, inserts are performed first, then updates, and then deletes. The `TableAdapterManager` provides an `UpdateOrder` property that can be set to perform updates first, then inserts, and then deletes.

NOTE

It's important to understand that the update order is all inclusive. That is, when updates are performed, inserts and then deletes are performed for all tables in the dataset.

To set the `UpdateOrder` property, after dragging items from the [Data Sources Window](#) onto a form, select the `TableAdapterManager` in the component tray, and then set the `UpdateOrder` property in the **Properties** window.

Create a backup copy of a dataset before performing a hierarchical update

When you save data (by calling the `TableAdapterManager.UpdateAll()` method), the `TableAdapterManager` attempts to update the data for each table in a single transaction. If any part of the update for any table fails, the whole transaction is rolled back. In most situations, the rollback returns your application to its original state.

However, sometimes you might want to restore the dataset from the backup copy. One example of this might occur when you're using auto-increment values. For example, if a save operation is not successful, auto-increment values are not reset in the dataset, and the dataset continues to create auto-incrementing values. This leaves a gap in numbering that might not be acceptable in your application. In situations where this is an issue,

the `TableAdapterManager` provides a `BackupDataSetBeforeUpdate` property that replaces the existing dataset with a backup copy if the transaction fails.

NOTE

The backup copy is only in memory while the `TableAdapterManager.UpdateAll` method is running. Therefore, there is no programmatic access to this backup dataset because it either replaces the original dataset or goes out of scope as soon as the `TableAdapterManager.UpdateAll` method finishes running.

Modify the generated save code to perform the hierarchical update

Save changes from the related data tables in the dataset to the database by calling the `TableAdapterManager.UpdateAll` method and passing in the name of the dataset that contains the related tables. For example, run the `TableAdapterManager.UpdateAll(NorthwindDataSet)` method to send updates from all the tables in `NorthwindDataSet` to the back-end database.

After you drop the items from the **Data Sources** window, code is automatically added to the `Form_Load` event to populate each table (the `TableAdapter.Fill` methods). Code is also added to the **Save** button click event of the [BindingNavigator](#) to save data from the dataset back to the database (the `TableAdapterManager.UpdateAll` method).

The generated save code also contains a line of code that calls the `CustomersBindingSource.EndEdit` method. More specifically, it calls the `EndEdit` method of the first [BindingSource](#) that's added to the form. In other words, this code is only generated for the first table that's dragged from the **Data Sources** window onto the form. The `EndEdit` call commits any changes that are in process in any data-bound controls that are currently being edited. Therefore, if a data-bound control still has focus and you click the **Save** button, all pending edits in that control are committed before the actual save (the `TableAdapterManager.UpdateAll` method).

NOTE

The **Dataset Designer** only adds the `BindingSource.EndEdit` code for the first table that's dropped onto the form. Therefore, you have to add a line of code to call the `BindingSource.EndEdit` method for each related table on the form. For this walkthrough, this means you have to add a call to the `OrdersBindingSource.EndEdit` method.

To update the code to commit changes to the related tables before saving

1. Double-click the **Save** button on the [BindingNavigator](#) to open **Form1** in the Code Editor.
2. Add a line of code to call the `OrdersBindingSource.EndEdit` method after the line that calls the `CustomersBindingSource.EndEdit` method. The code in the **Save** button click event should resemble the following:

```
Me.Validate()  
Me.CustomersBindingSource.EndEdit()  
Me.OrdersBindingSource.EndEdit()  
Me.TableAdapterManager.UpdateAll(Me.NorthwindDataSet)
```

```
this.Validate();  
this.customersBindingSource.EndEdit();  
this.ordersBindingSource.EndEdit();  
this.tableAdapterManager.UpdateAll(this.northwindDataSet);
```

In addition to committing changes on a related child table before saving data to a database, you might also have

to commit newly created parent records before adding new child records to a dataset. In other words, you might have to add the new parent record (`Customer`) to the dataset before foreign key constraints enable new child records (`orders`) to be added to the dataset. To accomplish this, you can use the child `BindingSource.AddingNew` event.

NOTE

Whether you have to commit new parent records depends on the type of control that's used to bind to your data source. In this walkthrough, you use individual controls to bind to the parent table. This requires the additional code to commit the new parent record. If the parent records were instead displayed in a complex binding control like the `DataGridView`, this additional `EndEdit` call for the parent record would not be necessary. This is because the underlying data-binding functionality of the control handles the committing of the new records.

To add code to commit parent records in the dataset before adding new child records

1. Create an event handler for the `OrdersBindingSource.AddingNew` event.
 - Open `Form1` in design view, select `OrdersBindingSource` in the component tray, select **Events** in the **Properties** window, and then double-click the **AddingNew** event.
2. Add a line of code to the event handler that calls the `CustomersBindingSource.EndEdit` method. The code in the `OrdersBindingSource_AddingNew` event handler should resemble the following:

```
Me.CustomersBindingSource.EndEdit()
```

```
this.customersBindingSource.EndEdit();
```

TableAdapterManager reference

By default, a `TableAdapterManager` class is generated when you create a dataset that contains related tables. To prevent the class from being generated, change the value of the `Hierarchical Update` property of the dataset to false. When you drag a table that has a relation onto the design surface of a Windows Form or WPF page, Visual Studio declares a member variable of the class. If you don't use databinding, you have to manually declare the variable.

The `TableAdapterManager` class is not a .NET type. Therefore, you can't look it up in the documentation. It's created at design time as part of the dataset creation process.

The following are the frequently used methods and properties of the `TableAdapterManager` class:

| MEMBER | DESCRIPTION |
|---|--|
| <code>UpdateAll</code> method | Saves all data from all data tables. |
| <code>BackUpDataSetBeforeUpdate</code> property | Determines whether to create a backup copy of the dataset before executing the <code>TableAdapterManager.UpdateAll</code> method. Boolean. |

| MEMBER | DESCRIPTION |
|---|---|
| <i>tableName</i> <code>TableAdapter</code> property | Represents a <code>TableAdapter</code> . The generated <code>TableAdapterManager</code> contains a property for each <code>TableAdapter</code> it manages. For example, a dataset with a Customers and Orders table is generated with a <code>TableAdapterManager</code> that contains <code>CustomersTableAdapter</code> and <code>OrdersTableAdapter</code> properties. |
| <code>UpdateOrder</code> property | Controls the order of the individual insert, update, and delete commands. Set this to one of the values in the <code>TableAdapterManager.UpdateOrderOption</code> enumeration. By default, the <code>UpdateOrder</code> is set to InsertUpdateDelete . This means that inserts, then updates, and then deletes are performed for all tables in the dataset. |

See also

- [Save data back to the database](#)

Handle a concurrency exception

8/5/2021 • 9 minutes to read • [Edit Online](#)

Concurrency exceptions ([System.Data.DBConcurrencyException](#)) are raised when two users attempt to change the same data in a database at the same time. In this walkthrough, you create a Windows application that illustrates how to catch a [DBConcurrencyException](#), locate the row that caused the error, and learn a strategy for how to handle it.

This walkthrough takes you through the following process:

1. Create a new **Windows Forms Application** project.
2. Create a new dataset based on the Northwind Customers table.
3. Create a form with a [DataGridView](#) to display the data.
4. Fill a dataset with data from the Customers table in the Northwind database.
5. Use the **Show Table Data** feature in **Server Explorer** to access the Customers-table's data and change a record.
6. Change the same record to a different value, update the dataset, and attempt to write the changes to the database, which results in a concurrency error being raised.
7. Catch the error, then display the different versions of the record, allowing the user to determine whether to continue and update the database, or cancel the update.

Prerequisites

This walkthrough uses SQL Server Express LocalDB and the Northwind sample database.

1. If you don't have SQL Server Express LocalDB, install it either from the [SQL Server Express download page](#), or through the **Visual Studio Installer**. In the **Visual Studio Installer**, you can install SQL Server Express LocalDB as part of the **Data storage and processing** workload, or as an individual component.
2. Install the Northwind sample database by following these steps:
 - a. In Visual Studio, open the **SQL Server Object Explorer** window. (SQL Server Object Explorer is installed as part of the **Data storage and processing** workload in the Visual Studio Installer.) Expand the **SQL Server** node. Right-click on your LocalDB instance and select **New Query**.

A query editor window opens.
 - b. Copy the [Northwind Transact-SQL script](#) to your clipboard. This T-SQL script creates the Northwind database from scratch and populates it with data.
 - c. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes running and the Northwind database is created.

Create a new project

Begin by creating a new Windows Forms application:

1. In Visual Studio, on the **File** menu, select **New > Project**.

2. Expand either **Visual C#** or **Visual Basic** in the left-hand pane, then select **Windows Desktop**.
3. In the middle pane, select the **Windows Forms App** project type.
4. Name the project **ConcurrencyWalkthrough**, and then choose **OK**.

The **ConcurrencyWalkthrough** project is created and added to **Solution Explorer**, and a new form opens in the designer.

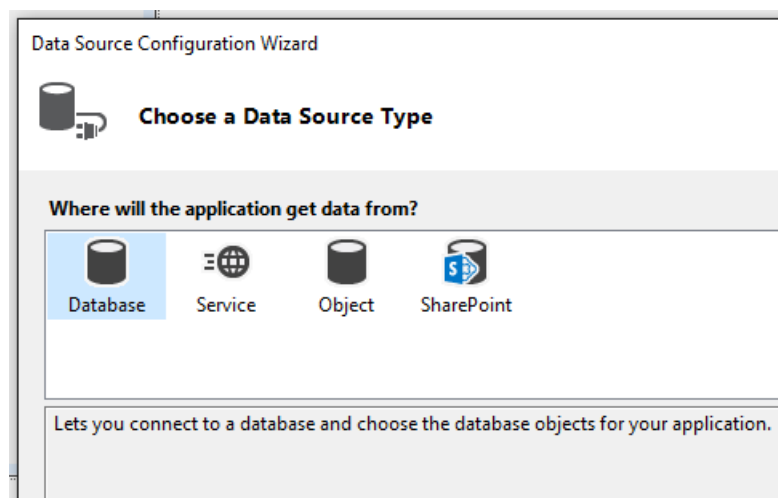
Create the Northwind dataset

Next, create a dataset named **NorthwindDataSet**:

1. On the **Data** menu, choose **Add New Data source**.

The Data Source Configuration Wizard opens.

2. On the **Choose a Data Source Type** screen, select **Database**.



3. Select a connection to the Northwind sample database from the list of available connections. If the connection is not available in the list of connections, select **New Connection**.

NOTE

If you're connecting to a local database file, select **No** when asked if you would you like to add the file to your project.

4. On the **Save connection string to the application configuration file** screen, select **Next**.
5. Expand the **Tables** node and select the **Customers** table. The default name for the dataset should be **NorthwindDataSet**.
6. Select **Finish** to add the dataset to the project.

Create a data-bound DataGridView control

In this section, you create a [System.Windows.Forms.DataGridView](#) by dragging the **Customers** item from the **Data Sources** window onto your Windows Form.

1. To open the **Data Sources** window, on the **Data** menu, choose **Show Data Sources**.
2. In the **Data Sources** window, expand the **NorthwindDataSet** node, and then select the **Customers** table.

3. Select the down arrow on the table node, and then select **DataGridView** in the drop-down list.
4. Drag the table onto an empty area of your form.

A [DataGridView](#) control named **CustomersDataGridView**, and a [BindingNavigator](#) named **CustomersBindingNavigator**, are added to the form that's bound to the [BindingSource](#). This is, in turn, bound to the Customers table in the NorthwindDataSet.

Test the form

You can now test the form to make sure it behaves as expected up to this point:

1. Select **F5** to run the application.

The form appears with a [DataGridView](#) control on it that's filled with data from the Customers table.

2. On the **Debug** menu, select **Stop Debugging**.

Handle concurrency errors

How you handle errors depends on the specific business rules that govern your application. For this walkthrough, we use the following strategy as an example for how to handle the concurrency error.

The application presents the user with three versions of the record:

- The current record in the database
- The original record that's loaded into the dataset
- The proposed changes in the dataset

The user is then able to either overwrite the database with the proposed version, or cancel the update and refresh the dataset with the new values from the database.

To enable the handling of concurrency errors

1. Create a custom error handler.
2. Display choices to the user.
3. Process the user's response.
4. Resend the update, or reset the data in the dataset.

Add code to handle the concurrency exception

When you attempt to perform an update and an exception is raised, you generally want to do something with the information that's provided by the raised exception. In this section, you add code that attempts to update the database. You also handle any [DBConcurrencyException](#) that might be raised, as well as any other exceptions.

NOTE

The `CreateMessage` and `ProcessDialogResults` methods are added later in the walkthrough.

1. Add the following code below the `Form1_Load` method:


```

private void UpdateDatabase()
{
    try
    {
        this.customersTableAdapter.Update(this.northwindDataSet.Customers);
        MessageBox.Show("Update successful");
    }
    catch (DBConcurrencyException dbcx)
    {
        DialogResult response = MessageBox.Show(CreateMessage((NorthwindDataSet.CustomersRow)
            (dbcx.Row)), "Concurrency Exception", MessageBoxButtons.YesNo);

        ProcessDialogResult(response);
    }
    catch (Exception ex)
    {
        MessageBox.Show("An error was thrown while attempting to update the database.");
    }
}

```

```

Private Sub UpdateDatabase()

    Try
        Me.CustomersTableAdapter.Update(Me.NorthwindDataSet.Customers)
        MsgBox("Update successful")

    Catch dbcx As Data.DBConcurrencyException
        Dim response As Windows.Forms.DialogResult

        response = MessageBox.Show(CreateMessage(CType(dbcx.Row, NorthwindDataSet.CustomersRow)),
            "Concurrency Exception", MessageBoxButtons.YesNo)

        ProcessDialogResult(response)

    Catch ex As Exception
        MsgBox("An error was thrown while attempting to update the database.")
    End Try
End Sub

```

2. Replace the `CustomersBindingNavigatorSaveItem_Click` method to call the `UpdateDatabase` method so it looks like the following:

```

private void customersBindingNavigatorSaveItem_Click(object sender, EventArgs e)
{
    UpdateDatabase();
}

```

```

Private Sub CustomersBindingNavigatorSaveItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CustomersBindingNavigatorSaveItem.Click
    UpdateDatabase()
End Sub

```

Display choices to the user

The code you just wrote calls the `CreateMessage` procedure to display error information to the user. For this walkthrough, you use a message box to display the different versions of the record to the user. This enables the user to choose whether to overwrite the record with the changes or cancel the edit. Once the user selects an option (clicks a button) on the message box, the response is passed to the `ProcessDialogResult` method.

Create the message by adding the following code to the **Code Editor**. Enter this code below the

UpdateDatabase method:

```
private string CreateMessage(NorthwindDataSet.CustomersRow cr)
{
    return
        "Database: " + GetRowData(GetCurrentRowInDB(cr), DataRowVersion.Default) + "\n" +
        "Original: " + GetRowData(cr, DataRowVersion.Original) + "\n" +
        "Proposed: " + GetRowData(cr, DataRowVersion.Current) + "\n" +
        "Do you still want to update the database with the proposed value?";
}

//-----
// This method loads a temporary table with current records from the database
// and returns the current values from the row that caused the exception.
//-----
private NorthwindDataSet.CustomersDataTable tempCustomersDataTable =
    new NorthwindDataSet.CustomersDataTable();

private NorthwindDataSet.CustomersRow GetCurrentRowInDB(NorthwindDataSet.CustomersRow RowWithError)
{
    this.customersTableAdapter.Fill(tempCustomersDataTable);

    NorthwindDataSet.CustomersRow currentRowInDb =
        tempCustomersDataTable.FindByCustomerID(RowWithError.CustomerID);

    return currentRowInDb;
}

//-----
// This method takes a CustomersRow and RowVersion
// and returns a string of column values to display to the user.
//-----
private string GetRowData(NorthwindDataSet.CustomersRow custRow, DataRowVersion RowVersion)
{
    string rowData = "";

    for (int i = 0; i < custRow.ItemArray.Length ; i++ )
    {
        rowData = rowData + custRow[i, RowVersion].ToString() + " ";
    }
    return rowData;
}
```

```

Private Function CreateMessage(ByVal cr As NorthwindDataSet.CustomersRow) As String
    Return "Database: " & GetRowData(GetCurrentRowInDB(cr),
                                     Data.DataRowVersion.Default) & vbCrLf &
           "Original: " & GetRowData(cr, Data.DataRowVersion.Original) & vbCrLf &
           "Proposed: " & GetRowData(cr, Data.DataRowVersion.Current) & vbCrLf &
           "Do you still want to update the database with the proposed value?"
End Function

'-----
' This method loads a temporary table with current records from the database
' and returns the current values from the row that caused the exception.
'-----
Private TempCustomersDataTable As New NorthwindDataSet.CustomersDataTable

Private Function GetCurrentRowInDB(
    ByVal RowWithError As NorthwindDataSet.CustomersRow
) As NorthwindDataSet.CustomersRow

    Me.CustomersTableAdapter.Fill(TempCustomersDataTable)

    Dim currentRowInDb As NorthwindDataSet.CustomersRow =
        TempCustomersDataTable.FindByCustomerID(RowWithError.CustomerID)

    Return currentRowInDb
End Function

'-----
' This method takes a CustomersRow and RowVersion
' and returns a string of column values to display to the user.
'-----
Private Function GetRowData(ByVal custRow As NorthwindDataSet.CustomersRow,
    ByVal RowVersion As Data.DataRowVersion) As String

    Dim rowData As String = ""

    For i As Integer = 0 To custRow.ItemArray.Length - 1
        rowData &= custRow.Item(i, RowVersion).ToString() & " "
    Next

    Return rowData
End Function

```

Process the user's response

You also need code to process the user's response to the message box. The options are either to overwrite the current record in the database with the proposed change, or abandon the local changes and refresh the data table with the record that's currently in the database. If the user chooses **Yes**, the [Merge](#) method is called with the *preserveChanges* argument set to **true**. This causes the update attempt to be successful, because the original version of the record now matches the record in the database.

Add the following code below the code that was added in the previous section:

```
// This method takes the DialogResult selected by the user and updates the database
// with the new values or cancels the update and resets the Customers table
// (in the dataset) with the values currently in the database.

private void ProcessDialogResult(DialogResult response)
{
    switch (response)
    {
        case DialogResult.Yes:
            northwindDataSet.Merge(tempCustomersDataTable, true, MissingSchemaAction.Ignore);
            UpdateDatabase();
            break;

        case DialogResult.No:
            northwindDataSet.Merge(tempCustomersDataTable);
            MessageBox.Show("Update cancelled");
            break;
    }
}
```

```
' This method takes the DialogResult selected by the user and updates the database
' with the new values or cancels the update and resets the Customers table
' (in the dataset) with the values currently in the database.

Private Sub ProcessDialogResult(ByVal response As Windows.Forms.DialogResult)

    Select Case response

        Case Windows.Forms.DialogResult.Yes
            NorthwindDataSet.Customers.Merge(TempCustomersDataTable, True)
            UpdateDatabase()

        Case Windows.Forms.DialogResult.No
            NorthwindDataSet.Customers.Merge(TempCustomersDataTable)
            MsgBox("Update cancelled")
    End Select
End Sub
```

Test the form behavior

You can now test the form to make sure it behaves as expected. To simulate a concurrency violation, you change data in the database after filling the NorthwindDataSet.

1. Select **F5** to run the application.
2. After the form appears, leave it running and switch to the Visual Studio IDE.
3. On the **View** menu, choose **Server Explorer**.
4. In **Server Explorer**, expand the connection your application is using, and then expand the **Tables** node.
5. Right-click the **Customers** table, and then select **Show Table Data**.
6. In the first record (**ALFKI**), change **ContactName** to **Maria Anders2**.

NOTE

Navigate to a different row to commit the change.

7. Switch to the ConcurrencyWalkthrough's running form.

8. In the first record on the form (ALFKI), change **ContactName** to **Maria Anders1**.

9. Select the **Save** button.

The concurrency error is raised, and the message box appears.

Selecting **No** cancels the update and updates the dataset with the values that are currently in the database. Selecting **Yes** writes the proposed value to the database.

See also

- [Save data back to the database](#)

How to: Save data by using a transaction

8/5/2021 • 2 minutes to read • [Edit Online](#)

You save data in a transaction by using the [System.Transactions](#) namespace. Use the [TransactionScope](#) object to participate in a transaction that is automatically managed for you.

Projects are not created with a reference to the *System.Transactions* assembly, so you need to manually add a reference to projects that use transactions.

The easiest way to implement a transaction is to instantiate a [TransactionScope](#) object in a `using` statement. (For more information, see [Using statement](#), and [Using statement](#).) The code that runs within the `using` statement participates in the transaction.

To commit the transaction, call the [Complete](#) method as the last statement in the using block.

To roll back the transaction, throw an exception prior to calling the [Complete](#) method.

To add a reference to the System.Transactions.dll

1. On the **Project** menu, select **Add Reference**.
2. On the **.NET** tab (**SQL Server** tab for SQL Server projects), select **System.Transactions**, and then select **OK**.

A reference to *System.Transactions.dll* is added to the project.

To save data in a transaction

- Add code to save data within the using statement that contains the transaction. The following code shows how to create and instantiate a [TransactionScope](#) object in a using statement:

```
Using updateTransaction As New Transactions.TransactionScope

    ' Add code to save your data here.
    ' Throw an exception to roll back the transaction.

    ' Call the Complete method to commit the transaction
    updateTransaction.Complete()
End Using
```

```
using (System.Transactions.TransactionScope updateTransaction =
    new System.Transactions.TransactionScope())
{
    // Add code to save your data here.
    // Throw an exception to roll back the transaction.

    // Call the Complete method to commit the transaction
    updateTransaction.Complete();
}
```

See also

- [Save data back to the database](#)

- [Walkthrough: Save data in a transaction](#)

Walkthrough: Save data in a transaction

8/5/2021 • 6 minutes to read • [Edit Online](#)

This walkthrough demonstrates how to save data in a transaction by using the [System.Transactions](#) namespace. In this walkthrough, you'll create a Windows Forms application. You'll use the Data Source Configuration Wizard to create a dataset for two tables in the Northwind sample database. You'll add data bound controls to a Windows form, and you'll modify the code for the BindingNavigator's save button to update the database inside a TransactionScope.

Prerequisites

This walkthrough uses SQL Server Express LocalDB and the Northwind sample database.

1. If you don't have SQL Server Express LocalDB, install it either from the [SQL Server Express download page](#), or through the **Visual Studio Installer**. In the Visual Studio Installer, SQL Server Express LocalDB can be installed as part of the **.NET desktop development** workload, or as an individual component.
2. Install the Northwind sample database by following these steps:
 - a. In Visual Studio, open the **SQL Server Object Explorer** window. (SQL Server Object Explorer is installed as part of the **Data storage and processing** workload in the Visual Studio Installer.) Expand the **SQL Server** node. Right-click on your LocalDB instance and select **New Query**.

A query editor window opens.
 - b. Copy the [Northwind Transact-SQL script](#) to your clipboard. This T-SQL script creates the Northwind database from scratch and populates it with data.
 - c. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes running and the Northwind database is created.

Create a Windows Forms application

The first step is to create a **Windows Forms Application**.

1. In Visual Studio, on the **File** menu, select **New > Project**.
2. Expand either **Visual C#** or **Visual Basic** in the left-hand pane, then select **Windows Desktop**.
3. In the middle pane, select the **Windows Forms App** project type.
4. Name the project **SavingDataInATransactionWalkthrough**, and then choose **OK**.

The **SavingDataInATransactionWalkthrough** project is created and added to **Solution Explorer**.

Create a database data source

This step uses the **Data Source Configuration Wizard** to create a data source based on the `Customers` and `Orders` tables in the Northwind sample database.

1. To open the **Data Sources** window, on the **Data** menu, select **Show Data Sources**.
2. In the **Data Sources** window, select **Add New Data Source** to start the **Data Source Configuration Wizard**.

3. On the **Choose a Data Source Type** screen, select **Database**, and then select **Next**.
4. On the **Choose your Data Connection** screen do one of the following:
 - If a data connection to the Northwind sample database is available in the drop-down list, select it.
 - or-
 - Select **New Connection** to launch the **Add/Modify Connection** dialog box and create a connection to the Northwind database.
5. If your database requires a password, select the option to include sensitive data, and then select **Next**.
6. On the **Save connection string to the Application Configuration file** screen, select **Next**.
7. On the **Choose your Database Objects** screen, expand the **Tables** node.
8. Select the **Customers** and **Orders** tables, and then select **Finish**.

The **NorthwindDataSet** is added to your project and the **Customers** and **Orders** tables appear in the **Data Sources** window.

Add controls to the form

You can create the data-bound controls by dragging items from the **Data Sources** window onto your form.

1. In the **Data Sources** window, expand the **Customers** node.
2. Drag the main **Customers** node from the **Data Sources** window onto **Form1**.

A **DataGridView** control and a tool strip (**BindingNavigator**) for navigating records appear on the form. A **NorthwindDataSet**, **CustomersTableAdapter**, **BindingSource**, and **BindingNavigator** appear in the component tray.

3. Drag the related **Orders** node (not the main **Orders** node, but the related child-table node below the **Fax** column) onto the form below the **CustomersDataGridView**.

A **DataGridView** appears on the form. An **OrdersTableAdapter** and **BindingSource** appear in the component tray.

Add a reference to the System.Transactions assembly

Transactions use the **System.Transactions** namespace. A project reference to the system.transactions assembly is not added by default, so you need to manually add it.

To add a reference to the System.Transactions DLL file

1. On the **Project** menu, select **Add Reference**.
2. Select **System.Transactions** (on the **.NET** tab), and then select **OK**.

A reference to **System.Transactions** is added to the project.

Modify the code in the BindingNavigator's SaveItem button

For the first table dropped onto your form, code is added by default to the **click** event of the save button on the **BindingNavigator**. You need to manually add code to update any additional tables. For this walkthrough, we refactor the existing save code out of the save button's click event handler. We also create a few more methods to provide specific update functionality based on whether the row needs to be added or deleted.

To modify the auto-generated save code

1. Select the **Save** button on the **CustomersBindingNavigator** (the button with the floppy disk icon).
2. Replace the `CustomersBindingNavigatorSaveItem_Click` method with the following code:

```
Private Sub CustomersBindingNavigatorSaveItem_Click() Handles CustomersBindingNavigatorSaveItem.Click
    UpdateData()
End Sub

Private Sub UpdateData()
    Me.Validate()
    Me.CustomersBindingSource.EndEdit()
    Me.OrdersBindingSource.EndEdit()

    Using updateTransaction As New Transactions.TransactionScope

        DeleteOrders()
        DeleteCustomers()
        AddNewCustomers()
        AddNewOrders()

        updateTransaction.Complete()
        NorthwindDataSet.AcceptChanges()
    End Using
End Sub
```

```
private void customersBindingNavigatorSaveItem_Click(object sender, EventArgs e)
{
    UpdateData();
}

private void UpdateData()
{
    this.Validate();
    this.customersBindingSource.EndEdit();
    this.ordersBindingSource.EndEdit();

    using (System.Transactions.TransactionScope updateTransaction =
        new System.Transactions.TransactionScope())
    {
        DeleteOrders();
        DeleteCustomers();
        AddNewCustomers();
        AddNewOrders();

        updateTransaction.Complete();
        northwindDataSet.AcceptChanges();
    }
}
```

The order for reconciling changes to related data is as follows:

- Delete child records. (In this case, delete records from the `Orders` table.)
- Delete parent records. (In this case, delete records from the `Customers` table.)
- Insert parent records. (In this case, insert records in the `Customers` table.)
- Insert child records. (In this case, insert records in the `Orders` table.)

To delete existing orders

- Add the following `DeleteOrders` method to **Form1**:

```

Private Sub DeleteOrders()

    Dim deletedOrders As NorthwindDataSet.OrdersDataTable
    deletedOrders = CType(NorthwindDataSet.Orders.GetChanges(Data.DataRowState.Deleted),
        NorthwindDataSet.OrdersDataTable)

    If Not IsNothing(deletedOrders) Then
        Try
            OrdersTableAdapter.Update(deletedOrders)

            Catch ex As Exception
                MessageBox.Show("DeleteOrders Failed")
            End Try
        End If
    End Sub

```

```

private void DeleteOrders()
{
    NorthwindDataSet.OrdersDataTable deletedOrders;
    deletedOrders = (NorthwindDataSet.OrdersDataTable)
        northwindDataSet.Orders.GetChanges(DataRowState.Deleted);

    if (deletedOrders != null)
    {
        try
        {
            ordersTableAdapter.Update(deletedOrders);
        }
        catch (System.Exception ex)
        {
            MessageBox.Show("DeleteOrders Failed");
        }
    }
}

```

To delete existing customers

- Add the following `DeleteCustomers` method to **Form1**:

```

Private Sub DeleteCustomers()

    Dim deletedCustomers As NorthwindDataSet.CustomersDataTable
    deletedCustomers = CType(NorthwindDataSet.Customers.GetChanges(Data.DataRowState.Deleted),
        NorthwindDataSet.CustomersDataTable)

    If Not IsNothing(deletedCustomers) Then
        Try
            CustomersTableAdapter.Update(deletedCustomers)

            Catch ex As Exception
                MessageBox.Show("DeleteCustomers Failed" & vbCrLf & ex.Message)
            End Try
        End If
    End Sub

```

```

private void DeleteCustomers()
{
    NorthwindDataSet.CustomersDataTable deletedCustomers;
    deletedCustomers = (NorthwindDataSet.CustomersDataTable)
        northwindDataSet.Customers.GetChanges(DataRowState.Deleted);

    if (deletedCustomers != null)
    {
        try
        {
            customersTableAdapter.Update(deletedCustomers);
        }
        catch (System.Exception ex)
        {
            MessageBox.Show("DeleteCustomers Failed");
        }
    }
}

```

To add new customers

- Add the following `AddNewCustomers` method to **Form1**:

```

Private Sub AddNewCustomers()

    Dim newCustomers As NorthwindDataSet.CustomersDataTable
    newCustomers = CType(NorthwindDataSet.Customers.GetChanges(Data.DataRowState.Added),
        NorthwindDataSet.CustomersDataTable)

    If Not IsNothing(newCustomers) Then
        Try
            CustomersTableAdapter.Update(newCustomers)

            Catch ex As Exception
                MessageBox.Show("AddNewCustomers Failed" & vbCrLf & ex.Message)
            End Try
        End If
    End Sub

```

```

private void AddNewCustomers()
{
    NorthwindDataSet.CustomersDataTable newCustomers;
    newCustomers = (NorthwindDataSet.CustomersDataTable)
        northwindDataSet.Customers.GetChanges(DataRowState.Added);

    if (newCustomers != null)
    {
        try
        {
            customersTableAdapter.Update(newCustomers);
        }
        catch (System.Exception ex)
        {
            MessageBox.Show("AddNewCustomers Failed");
        }
    }
}

```

To add new orders

- Add the following `AddNewOrders` method to **Form1**:

```

Private Sub AddNewOrders()

    Dim newOrders As NorthwindDataSet.OrdersDataTable
    newOrders = CType(NorthwindDataSet.Orders.GetChanges(Data.DataRowState.Added),
        NorthwindDataSet.OrdersDataTable)

    If Not IsNothing(newOrders) Then
        Try
            OrdersTableAdapter.Update(newOrders)

            Catch ex As Exception
                MessageBox.Show("AddNewOrders Failed" & vbCrLf & ex.Message)
            End Try
        End If
    End Sub

```

```

private void AddNewOrders()
{
    NorthwindDataSet.OrdersDataTable newOrders;
    newOrders = (NorthwindDataSet.OrdersDataTable)
        northwindDataSet.Orders.GetChanges(DataRowState.Added);

    if (newOrders != null)
    {
        try
        {
            ordersTableAdapter.Update(newOrders);
        }
        catch (System.Exception ex)
        {
            MessageBox.Show("AddNewOrders Failed");
        }
    }
}

```

Run the application

Press F5 to run the application.

See also

- [How to: save data by using a transaction](#)
- [Save data back to the database](#)

Save data to a database (multiple tables)

8/5/2021 • 6 minutes to read • [Edit Online](#)

One of the most common scenarios in application development is to display data on a form in a Windows application, edit the data, and send the updated data back to the database. This walkthrough creates a form that displays data from two related tables and shows how to edit records and save changes back to the database. This example uses the `Customers` and `Orders` tables from the Northwind sample database.

You can save data in your application back to the database by calling the `Update` method of a `TableAdapter`. When you drag tables from the **Data Sources** window onto a form, the code that's required to save data is automatically added. Any additional tables that are added to a form require the manual addition of this code. This walkthrough shows how to add code to save updates from more than one table.

Tasks illustrated in this walkthrough include:

- Creating and configuring a data source in your application with the [Data Source Configuration Wizard](#).
- Setting the controls of the items in the [Data Sources window](#). For more information, see [Set the control to be created when dragging from the Data Sources window](#).
- Creating data-bound controls by dragging items from the **Data Sources** window onto your form.
- Modifying a few records in each table in the dataset.
- Modifying the code to send the updated data in the dataset back to the database.

Prerequisites

This walkthrough uses SQL Server Express LocalDB and the Northwind sample database.

1. If you don't have SQL Server Express LocalDB, install it either from the [SQL Server Express download page](#), or through the **Visual Studio Installer**. In the **Visual Studio Installer**, you can install SQL Server Express LocalDB as part of the **Data storage and processing** workload, or as an individual component.
2. Install the Northwind sample database by following these steps:
 - a. In Visual Studio, open the **SQL Server Object Explorer** window. (SQL Server Object Explorer is installed as part of the **Data storage and processing** workload in the Visual Studio Installer.) Expand the **SQL Server** node. Right-click on your LocalDB instance and select **New Query**.

A query editor window opens.
 - b. Copy the [Northwind Transact-SQL script](#) to your clipboard. This T-SQL script creates the Northwind database from scratch and populates it with data.
 - c. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes running and the Northwind database is created.

Create the Windows Forms application

Create a new **Windows Forms App** project for either C# or Visual Basic. Name the project `UpdateMultipleTablesWalkthrough`.

Create the data source

This step creates a data source from the Northwind database using the **Data Source Configuration Wizard**. You must have access to the Northwind sample database to create the connection. For information about setting up the Northwind sample database, see [How to: Install sample databases](#).

1. On the **Data** menu, select **Show Data Sources**.

The **Data Sources** window opens.

2. In the **Data Sources** window, select **Add New Data Source** to start the **Data Source Configuration Wizard**.
3. On the **Choose a Data Source Type** screen, select **Database**, and then select **Next**.
4. On the **Choose your Data Connection** screen, do one of the following:
 - If a data connection to the Northwind sample database is available in the drop-down list, select it.
 - or-
 - Select **New Connection** to open the **Add/Modify Connection** dialog box.
5. If your database requires a password, select the option to include sensitive data, and then select **Next**.
6. On the **Save connection string to the Application Configuration file**, select **Next**.
7. On the **Choose your Database Objects** screen, expand the **Tables** node.
8. Select the **Customers** and **Orders** tables, and then select **Finish**.

The **NorthwindDataSet** is added to your project, and the tables appear in the **Data Sources** window.

Set the controls to be created

For this walkthrough, the data in the **Customers** table is in a **Details** layout where data is displayed in individual controls. The data from the **Orders** table is in a **Grid** layout that's displayed in a **DataGridView** control.

To set the drop type for the items in the Data Sources window

1. In the **Data Sources** window, expand the **Customers** node.
2. On the **Customers** node, select **Details** from the control list to change the control of the **Customers** table to individual controls. For more information, see [Set the control to be created when dragging from the Data Sources window](#).

Create the data-bound form

You can create the data-bound controls by dragging items from the **Data Sources** window onto your form.

1. Drag the main **Customers** node from the **Data Sources** window onto **Form1**.

Data-bound controls with descriptive labels appear on the form, along with a tool strip (**BindingNavigator**) for navigating records. A **NorthwindDataSet**, **CustomersTableAdapter**, **BindingSource**, and **BindingNavigator** appear in the component tray.

2. Drag the related **Orders** node from the **Data Sources** window onto **Form1**.

NOTE

The related **Orders** node is located below the **Fax** column and is a child node of the **Customers** node.

A [DataGridView](#) control and a tool strip ([BindingNavigator](#)) for navigating records appear on the form. An `OrdersTableAdapter` and [BindingSource](#) appear in the component tray.

Add code to update the database

You can update the database by calling the `Update` methods of the **Customers** and **Orders** TableAdapters. By default, an event handler for the **Save** button of the [BindingNavigator](#) is added to the form's code to send updates to the database. This procedure modifies the code to send updates in the correct order. This eliminates the possibility of raising referential integrity errors. The code also implements error handling by wrapping the update call in a try-catch block. You can modify the code to suit the needs of your application.

NOTE

For clarity, this walkthrough does not use a transaction. However, if you're updating two or more related tables, include all the update logic within a transaction. A transaction is a process that assures that all related changes to a database are successful before any changes are committed. For more information, see [Transactions and Concurrency](#).

To add update logic to the application

1. Select the **Save** button on the [BindingNavigator](#). This opens the Code Editor to the `bindingNavigatorSaveItem_Click` event handler.
2. Replace the code in the event handler to call the `Update` methods of the related TableAdapters. The following code first creates three temporary data tables to hold the updated information for each [DataRowState](#) ([Deleted](#), [Added](#), and [Modified](#)). The updates are run in the correct order. The code should look like the following:


```

Me.Validate()
Me.OrdersBindingSource.EndEdit()
Me.CustomersBindingSource.EndEdit()

Dim deletedOrders As NorthwindDataSet.OrdersDataTable = CType(
    NorthwindDataSet.Orders.GetChanges(Data.DataRowState.Deleted), NorthwindDataSet.OrdersDataTable)

Dim newOrders As NorthwindDataSet.OrdersDataTable = CType(
    NorthwindDataSet.Orders.GetChanges(Data.DataRowState.Added), NorthwindDataSet.OrdersDataTable)

Dim modifiedOrders As NorthwindDataSet.OrdersDataTable = CType(
    NorthwindDataSet.Orders.GetChanges(Data.DataRowState.Modified), NorthwindDataSet.OrdersDataTable)

Try
    ' Remove all deleted orders from the Orders table.
    If Not deletedOrders Is Nothing Then
        OrdersTableAdapter.Update(deletedOrders)
    End If

    ' Update the Customers table.
    CustomersTableAdapter.Update(NorthwindDataSet.Customers)

    ' Add new orders to the Orders table.
    If Not newOrders Is Nothing Then
        OrdersTableAdapter.Update(newOrders)
    End If

    ' Update all modified Orders.
    If Not modifiedOrders Is Nothing Then
        OrdersTableAdapter.Update(modifiedOrders)
    End If

    NorthwindDataSet.AcceptChanges()

Catch ex As Exception
    MsgBox("Update failed")

Finally
    If Not deletedOrders Is Nothing Then
        deletedOrders.Dispose()
    End If

    If Not newOrders Is Nothing Then
        newOrders.Dispose()
    End If

    If Not modifiedOrders Is Nothing Then
        modifiedOrders.Dispose()
    End If
End Try

```

```

this.Validate();
this.ordersBindingSource.EndEdit();
this.customersBindingSource.EndEdit();

NorthwindDataSet.OrdersDataTable deletedOrders = (NorthwindDataSet.OrdersDataTable)
    northwindDataSet.Orders.GetChanges(DataRowState.Deleted);

NorthwindDataSet.OrdersDataTable newOrders = (NorthwindDataSet.OrdersDataTable)
    northwindDataSet.Orders.GetChanges(DataRowState.Added);

NorthwindDataSet.OrdersDataTable modifiedOrders = (NorthwindDataSet.OrdersDataTable)
    northwindDataSet.Orders.GetChanges(DataRowState.Modified);

try
{
    // Remove all deleted orders from the Orders table.
    if (deletedOrders != null)
    {
        ordersTableAdapter.Update(deletedOrders);
    }

    // Update the Customers table.
    customersTableAdapter.Update(northwindDataSet.Customers);

    // Add new orders to the Orders table.
    if (newOrders != null)
    {
        ordersTableAdapter.Update(newOrders);
    }

    // Update all modified Orders.
    if (modifiedOrders != null)
    {
        ordersTableAdapter.Update(modifiedOrders);
    }

    northwindDataSet.AcceptChanges();
}

catch (System.Exception ex)
{
    MessageBox.Show("Update failed");
}

finally
{
    if (deletedOrders != null)
    {
        deletedOrders.Dispose();
    }
    if (newOrders != null)
    {
        newOrders.Dispose();
    }
    if (modifiedOrders != null)
    {
        modifiedOrders.Dispose();
    }
}

```

Test the application

1. Press F5.
2. Make some changes to the data of one or more records in each table.

3. Select the **Save** button.
4. Check the values in the database to verify that the changes were saved.

See also

- [Save data back to the database](#)

Save data from an object to a database

8/5/2021 • 3 minutes to read • [Edit Online](#)

You can save data in objects to a database by passing the values from your object to one of the `TableAdapter`'s `DBDirect` methods (for example, `TableAdapter.Insert`). For more information, see [TableAdapter](#).

To save data from a collection of objects, loop through the collection of objects (for example, a for-next loop), and send the values for each object to the database by using one of the `TableAdapter`'s `DBDirect` methods.

By default, `DBDirect` methods are created on a `TableAdapter` that can be run directly against the database. These methods can be called directly and don't require [DataSet](#) or [DataTable](#) objects to reconcile changes in order to send updates to a database.

NOTE

When you're configuring a `TableAdapter`, the main query must provide enough information for the `DBDirect` methods to be created. For example, if a `TableAdapter` is configured to query data from a table that does not have a primary key column defined, it does not generate `DBDirect` methods.

| TABLEADAPTER DBDIRECT METHOD | DESCRIPTION |
|----------------------------------|---|
| <code>TableAdapter.Insert</code> | Adds new records to a database and enables you to pass in individual column values as method parameters. |
| <code>TableAdapter.Update</code> | <p>Updates existing records in a database. The <code>Update</code> method takes original and new column values as method parameters. The original values are used to locate the original record, and the new values are used to update that record.</p> <p>The <code>TableAdapter.Update</code> method is also used to reconcile changes in a dataset back to the database by taking a DataSet, DataTable, DataRow, or an array of DataRows as method parameters.</p> |
| <code>TableAdapter.Delete</code> | Deletes existing records from the database based on the original column values passed in as method parameters. |

To save new records from an object to a database

- Create the records by passing the values to the `TableAdapter.Insert` method.

The following example creates a new customer record in the `Customers` table by passing the values in the `currentCustomer` object to the `TableAdapter.Insert` method.

```
private void AddNewCustomers(Customer currentCustomer)
{
    customersTableAdapter.Insert(
        currentCustomer.CustomerID,
        currentCustomer.CompanyName,
        currentCustomer.ContactName,
        currentCustomer.ContactTitle,
        currentCustomer.Address,
        currentCustomer.City,
        currentCustomer.Region,
        currentCustomer.PostalCode,
        currentCustomer.Country,
        currentCustomer.Phone,
        currentCustomer.Fax);
}
```

```
Private Sub AddNewCustomer(ByVal currentCustomer As Customer)

    CustomersTableAdapter.Insert(
        currentCustomer.CustomerID,
        currentCustomer.CompanyName,
        currentCustomer.ContactName,
        currentCustomer.ContactTitle,
        currentCustomer.Address,
        currentCustomer.City,
        currentCustomer.Region,
        currentCustomer.PostalCode,
        currentCustomer.Country,
        currentCustomer.Phone,
        currentCustomer.Fax)

End Sub
```

To update existing records from an object to a database

- Modify the records by calling the `TableAdapter.Update` method, passing in the new values to update the record, and passing in the original values to locate the record.

NOTE

Your object needs to maintain the original values in order to pass them to the `Update` method. This example uses properties with an `orig` prefix to store the original values.

The following example updates an existing record in the `Customers` table by passing the new and original values in the `Customer` object to the `TableAdapter.Update` method.

```

private void UpdateCustomer(Customer cust)
{
    customersTableAdapter.Update(
        cust.CustomerID,
        cust.CompanyName,
        cust.ContactName,
        cust.ContactTitle,
        cust.Address,
        cust.City,
        cust.Region,
        cust.PostalCode,
        cust.Country,
        cust.Phone,
        cust.Fax,
        cust.origCustomerID,
        cust.origCompanyName,
        cust.origContactName,
        cust.origContactTitle,
        cust.origAddress,
        cust.origCity,
        cust.origRegion,
        cust.origPostalCode,
        cust.origCountry,
        cust.origPhone,
        cust.origFax);
}

```

```

Private Sub UpdateCustomer(ByVal cust As Customer)

    CustomersTableAdapter.Update(
        cust.CustomerID,
        cust.CompanyName,
        cust.ContactName,
        cust.ContactTitle,
        cust.Address,
        cust.City,
        cust.Region,
        cust.PostalCode,
        cust.Country,
        cust.Phone,
        cust.Fax,
        cust.origCustomerID,
        cust.origCompanyName,
        cust.origContactName,
        cust.origContactTitle,
        cust.origAddress,
        cust.origCity,
        cust.origRegion,
        cust.origPostalCode,
        cust.origCountry,
        cust.origPhone,
        cust.origFax)

End Sub

```

To delete existing records from a database

- Delete the records by calling the `TableAdapter.Delete` method and passing in the original values to locate the record.

NOTE

Your object needs to maintain the original values in order to pass them to the `Delete` method. This example uses properties with an `orig` prefix to store the original values.

The following example deletes a record from the `Customers` table by passing the original values in the `Customer` object to the `TableAdapter.Delete` method.

```
private void DeleteCustomer(Customer cust)
{
    customersTableAdapter.Delete(
        cust.origCustomerID,
        cust.origCompanyName,
        cust.origContactName,
        cust.origContactTitle,
        cust.origAddress,
        cust.origCity,
        cust.origRegion,
        cust.origPostalCode,
        cust.origCountry,
        cust.origPhone,
        cust.origFax);
}
```

```
Private Sub DeleteCustomer(ByVal cust As Customer)

    CustomersTableAdapter.Delete(
        cust.origCustomerID,
        cust.origCompanyName,
        cust.origContactName,
        cust.origContactTitle,
        cust.origAddress,
        cust.origCity,
        cust.origRegion,
        cust.origPostalCode,
        cust.origCountry,
        cust.origPhone,
        cust.origFax)

End Sub
```

.NET security

You must have permission to perform the selected `INSERT`, `UPDATE`, or `DELETE` on the table in the database.

See also

- [Save data back to the database](#)

Save data with the TableAdapter DBDirect methods

8/5/2021 • 5 minutes to read • [Edit Online](#)

This walkthrough provides detailed instructions for running SQL statements directly against a database by using the DBDirect methods of a TableAdapter. The DBDirect methods of a TableAdapter provide a fine level of control over your database updates. You can use them to run specific SQL statements and stored procedures by calling the individual `Insert`, `Update`, and `Delete` methods as needed by your application (as opposed to the overloaded `Update` method that performs the UPDATE, INSERT, and DELETE statements all in one call).

During this walkthrough, you will learn how to:

- Create a new **Windows Forms Application**.
- Create and configure a dataset with the [Data Source Configuration Wizard](#).
- Select the control to be created on the form when dragging items from the **Data Sources** window. For more information, see [Set the control to be created when dragging from the Data Sources window](#).
- Create a data-bound form by dragging items from the **Data Sources** window onto the form.
- Add methods to directly access the database and perform inserts, updates, and deletes.

Prerequisites

This walkthrough uses SQL Server Express LocalDB and the Northwind sample database.

1. If you don't have SQL Server Express LocalDB, install it either from the [SQL Server Express download page](#), or through the **Visual Studio Installer**. In the **Visual Studio Installer**, you can install SQL Server Express LocalDB as part of the **Data storage and processing** workload, or as an individual component.
2. Install the Northwind sample database by following these steps:
 - a. In Visual Studio, open the **SQL Server Object Explorer** window. (SQL Server Object Explorer is installed as part of the **Data storage and processing** workload in the Visual Studio Installer.) Expand the **SQL Server** node. Right-click on your LocalDB instance and select **New Query**.

A query editor window opens.
 - b. Copy the [Northwind Transact-SQL script](#) to your clipboard. This T-SQL script creates the Northwind database from scratch and populates it with data.
 - c. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes running and the Northwind database is created.

Create a Windows Forms application

The first step is to create a **Windows Forms Application**.

1. In Visual Studio, on the **File** menu, select **New > Project**.
2. Expand either **Visual C#** or **Visual Basic** in the left-hand pane, then select **Windows Desktop**.
3. In the middle pane, select the **Windows Forms App** project type.

4. Name the project **TableAdapterDbDirectMethodsWalkthrough**, and then choose **OK**.

The **TableAdapterDbDirectMethodsWalkthrough** project is created and added to **Solution Explorer**.

Create a data source from your database

This step uses the **Data Source Configuration Wizard** to create a data source based on the **Region** table in the Northwind sample database. You must have access to the Northwind sample database to create the connection. For information about setting up the Northwind sample database, see [How to: Install sample databases](#).

To create the data source

1. On the **Data** menu, select **Show Data Sources**.

The **Data Sources** window opens.

2. In the **Data Sources** window, select **Add New Data Source** to start the **Data Source Configuration Wizard**.
3. On the **Choose a Data Source Type** screen, select **Database**, and then select **Next**.
4. On the **Choose your Data Connection** screen, do one of the following:
 - If a data connection to the Northwind sample database is available in the drop-down list, select it.
 - or-
 - Select **New Connection** to launch the **Add/Modify Connection** dialog box.
5. If your database requires a password, select the option to include sensitive data, and then select **Next**.
6. On the **Save connection string to the Application Configuration file** screen, select **Next**.
7. On the **Choose your Database Objects** screen, expand the **Tables** node.
8. Select the **Region** table, and then select **Finish**.

The **NorthwindDataSet** is added to your project and the **Region** table appears in the **Data Sources** window.

Add controls to the form to display the data

Create the data-bound controls by dragging items from the **Data Sources** window onto your form.

To create data bound controls on the Windows form, drag the main **Region** node from the **Data Sources** window onto the form.

A **DataGridView** control and a tool strip (**BindingNavigator**) for navigating records appear on the form. A **NorthwindDataSet**, **RegionTableAdapter**, **BindingSource**, and **BindingNavigator** appear in the component tray.

To add buttons that will call the individual TableAdapter DbDirect methods

1. Drag three **Button** controls from the **Toolbox** onto **Form1** (below the **RegionDataGridView**).
2. Set the following **Name** and **Text** properties on each button.

| NAME | TEXT |
|---------------------|---------------|
| InsertButton | Insert |

| NAME | TEXT |
|---|--------|
| <input type="button" value="UpdateButton"/> | Update |
| <input type="button" value="DeleteButton"/> | Delete |

To add code to insert new records into the database

1. Select **InsertButton** to create an event handler for the click event and open your form in the code editor.
2. Replace the `InsertButton_Click` event handler with the following code:

```
Private Sub InsertButton_Click() Handles InsertButton.Click

    Dim newRegionID As Integer = 5
    Dim newRegionDescription As String = "NorthEastern"

    Try
        RegionTableAdapter1.Insert(newRegionID, newRegionDescription)

    Catch ex As Exception
        MessageBox.Show("Insert Failed")
    End Try

    RefreshDataset()
End Sub

Private Sub RefreshDataset()
    Me.RegionTableAdapter1.Fill(Me.NorthwindDataSet1._Region)
End Sub
```

```
private void InsertButton_Click(object sender, EventArgs e)
{
    Int32 newRegionID = 5;
    String newRegionDescription = "NorthEastern";

    try
    {
        regionTableAdapter1.Insert(newRegionID, newRegionDescription);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Insert Failed");
    }
    RefreshDataset();
}

private void RefreshDataset()
{
    this.regionTableAdapter1.Fill(this.northwindDataSet1.Region);
}
```

To add code to update records in the database

1. Double-click the **UpdateButton** to create an event handler for the click event and open your form in the code editor.
2. Replace the `UpdateButton_Click` event handler with the following code:

```

Private Sub UpdateButton_Click() Handles UpdateButton.Click

    Dim newRegionID As Integer = 5

    Try
        RegionTableAdapter1.Update(newRegionID, "Updated Region Description", 5, "NorthEastern")

    Catch ex As Exception
        MessageBox.Show("Update Failed")
    End Try

    RefreshDataset()
End Sub

```

```

private void UpdateButton_Click(object sender, EventArgs e)
{
    Int32 newRegionID = 5;

    try
    {
        regionTableAdapter1.Update(newRegionID, "Updated Region Description", 5, "NorthEastern");
    }
    catch (Exception ex)
    {
        MessageBox.Show("Update Failed");
    }
    RefreshDataset();
}

```

To add code to delete records from the database

1. Select **DeleteButton** to create an event handler for the click event and open your form in the code editor.
2. Replace the `DeleteButton_Click` event handler with the following code:

```

Private Sub DeleteButton_Click() Handles DeleteButton.Click

    Try
        RegionTableAdapter1.Delete(5, "Updated Region Description")

    Catch ex As Exception
        MessageBox.Show("Delete Failed")
    End Try

    RefreshDataset()
End Sub

```

```

private void DeleteButton_Click(object sender, EventArgs e)
{
    try
    {
        regionTableAdapter1.Delete(5, "Updated Region Description");
    }
    catch (Exception ex)
    {
        MessageBox.Show("Delete Failed");
    }
    RefreshDataset();
}

```

Run the application

- Select **F5** to run the application.
- Select the **Insert** button, and verify that the new record appears in the grid.
- Select the **Update** button, and verify that the record is updated in the grid.
- Select the **Delete** button, and verify that the record is removed from the grid.

Next steps

Depending on your application requirements, there are several steps you might want to perform after creating a data-bound form. Some enhancements you could make to this walkthrough include:

- Adding search functionality to the form.
- Adding additional tables to the dataset by selecting **Configure DataSet with Wizard** from within the **Data Sources** window. You can add controls that display related data by dragging the related nodes onto the form. For more information, see [Relationships in Datasets](#).

See also

- [Save data back to the database](#)

Save a dataset as XML

8/5/2021 • 2 minutes to read • [Edit Online](#)

Access the XML data in a dataset by calling the available XML methods on the dataset. To save the data in XML format, you can call either the [GetXml](#) method or the [WriteXml](#) method of a [DataSet](#).

Calling the [GetXml](#) method returns a string that contains the data from all data tables in the dataset that's formatted as XML.

Calling the [WriteXml](#) method sends the XML-formatted data to a file that you specify.

To save the data in a dataset as XML to a variable

- The [GetXml](#) method returns a [String](#). Declare a variable of type [String](#) and assign it the results of the [GetXml](#) method.

```
Dim xmlData As String = NorthwindDataSet.GetXml()
```

```
string xmlData = northwindDataSet.GetXml();
```

To save the data in a dataset as XML to a file

- The [WriteXml](#) method has several overloads. Declare a variable and assign it a valid path to save the file to. The following code shows how to save the data to a file:

```
Dim filePath As String = "ENTER A VALID FILEPATH"  
NorthwindDataSet.WriteXml(filePath)
```

```
string filePath = "ENTER A VALID FILEPATH";  
northwindDataSet.WriteXml(filePath);
```

See also

- [Save data back to the database](#)

Query datasets

8/5/2021 • 4 minutes to read • [Edit Online](#)

To search for specific records in a dataset, use the `FindBy` method on the `DataTable`, write your own `foreach` statement to loop over the table's `Rows` collection, or use [LINQ to DataSet](#).

Dataset case sensitivity

Within a dataset, table and column names are case-insensitive by default — that is, a table in a dataset called "Customers" can also be referred to as "customers." This matches the naming conventions in many databases, including SQL Server. In SQL Server, the default behavior is that the names of data elements cannot be distinguished only by case.

NOTE

Unlike datasets, XML documents are case-sensitive, so the names of data elements defined in schemas are case-sensitive. For example, schema protocol allows the schema to define a table called "Customers" and a different table called "customers." This can result in name collisions when a schema that contains elements that differ only by case is used to generate a dataset class.

Case sensitivity, however, can be a factor in how data is interpreted within the dataset. For example, if you filter data in a dataset table, the search criteria might return different results depending on whether the comparison is case-sensitive. You can control the case sensitivity of filtering, searching, and sorting by setting the dataset's [CaseSensitive](#) property. All the tables in the dataset inherit the value of this property by default. (You can override this property for each individual table by setting the table's [CaseSensitive](#) property.)

Locate a specific row in a data table

To find a row in a typed dataset with a primary key value

- To locate a row, call the strongly typed `FindBy` method that uses the table's primary key.

In the following example, the `CustomerID` column is the primary key of the `Customers` table. This means that the generated `FindBy` method is `FindByCustomerID`. The example shows how to assign a specific [DataRow](#) to a variable by using the generated `FindBy` method.

```
NorthwindDataSet.CustomersRow customersRow =  
    northwindDataSet1.Customers.FindByCustomerID("ALFKI");
```

```
Dim customersRow As NorthwindDataSet.CustomersRow  
customersRow = NorthwindDataSet1.Customers.FindByCustomerID("ALFKI")
```

To find a row in an untyped dataset with a primary key value

- Call the [Find](#) method of a [DataRowCollection](#) collection, passing the primary key as a parameter.

The following example shows how to declare a new row called `foundRow` and assign it the return value of the [Find](#) method. If the primary key is found, the contents of column index 1 are displayed in a message box.

```

string s = "primaryKeyValue";
DataRow foundRow = dataSet1.Tables["AnyTable"].Rows.Find(s);

if (foundRow != null)
{
    MessageBox.Show(foundRow[0].ToString());
}
else
{
    MessageBox.Show("A row with the primary key of " + s + " could not be found");
}

```

```

Dim s As String = "primaryKeyValue"
Dim foundRow As DataRow = DataSet1.Tables("AnyTable").Rows.Find(s)

If foundRow IsNot Nothing Then
    MsgBox(foundRow(1).ToString())
Else
    MsgBox("A row with the primary key of " & s & " could not be found")
End If

```

Find rows by column values

To find rows based on the values in any column

- Data tables are created with the [Select](#) method, which returns an array of [DataRows](#) based on the expression passed to the [Select](#) method. For more information about creating valid expressions, see the "Expression Syntax" section of the page about the [Expression](#) property.

The following example shows how to use the [Select](#) method of the [DataTable](#) to locate specific rows.

```

DataRow[] foundRows;
foundRows = dataSet1.Tables["Customers"].Select("CompanyName Like 'A%'");

```

```

Dim foundRows() As Data.DataRow
foundRows = DataSet1.Tables("Customers").Select("CompanyName Like 'A%'")

```

Access related records

When tables in a dataset are related, a [DataRelation](#) object can make the related records available in another table. For example, a dataset containing [Customers](#) and [Orders](#) tables can be made available.

You can use a [DataRelation](#) object to locate related records by calling the [GetChildRows](#) method of a [DataRow](#) in the parent table. This method returns an array of related child records. Or, you can call the [GetParentRow](#) method of a [DataRow](#) in the child table. This method returns a single [DataRow](#) from the parent table.

This page provides examples using typed datasets. For information about navigating relationships in untyped datasets, see [Navigating DataRelations](#).

NOTE

If you are working in a Windows Forms application and using the data-binding features to display data, the designer-generated form might provide enough functionality for your application. For more information, see [Bind controls to data in Visual Studio](#). Specifically, see [Relationships in Datasets](#).

The following code examples demonstrate how to navigate up and down relationships in typed datasets. The code examples use typed [DataRow](#)s (`NorthwindDataSet.OrdersRow`) and the generated `FindByPrimaryKey` (`FindByCustomerID`) methods to locate a desired row and return the related records. The examples compile and run correctly only if you have:

- An instance of a dataset named `NorthwindDataSet` with a `Customers` table.
- An `Orders` table.
- A relationship named `FK_Orders_Customers` relating the two tables.

Additionally, both tables need to be filled with data for any records to be returned.

To return the child records of a selected parent record

- Call the [GetChildRows](#) method of a specific `Customers` data row, and return an array of rows from the `Orders` table:

```
string custID = "ALFKI";
NorthwindDataSet.OrdersRow[] orders;

orders = (NorthwindDataSet.OrdersRow[])northwindDataSet.Customers.
    FindByCustomerID(custID).GetChildRows("FK_Orders_Customers");

MessageBox.Show(orders.Length.ToString());
```

```
Dim customerID As String = "ALFKI"
Dim orders() As NorthwindDataSet.OrdersRow

orders = CType(NorthwindDataSet.Customers.FindByCustomerID(customerID).
    GetChildRows("FK_Orders_Customers"), NorthwindDataSet.OrdersRow())

MessageBox.Show(orders.Length.ToString())
```

To return the parent record of a selected child record

- Call the [GetParentRow](#) method of a specific `Orders` data row, and return a single row from the `Customers` table:

```
int orderID = 10707;
NorthwindDataSet.CustomersRow customer;

customer = (NorthwindDataSet.CustomersRow)northwindDataSet.Orders.
    FindByOrderID(orderID).GetParentRow("FK_Orders_Customers");

MessageBox.Show(customer.CompanyName);
```

```
Dim orderID As Integer = 10707
Dim customer As NorthwindDataSet.CustomersRow

customer = CType(NorthwindDataSet.Orders.FindByOrderID(orderID).
    GetParentRow("FK_Orders_Customers"), NorthwindDataSet.CustomersRow)

MessageBox.Show(customer.CompanyName)
```

See also

- [Dataset tools in Visual Studio](#)

Add validation to an n-tier dataset

8/5/2021 • 5 minutes to read • [Edit Online](#)

Adding validation to a dataset that is separated into an n-tier solution is basically the same as adding validation to a single-file dataset (a dataset in a single project). The suggested location for performing validation on data is during the [ColumnChanging](#) and/or [RowChanging](#) events of a data table.

The dataset provides the functionality to create partial classes to which you can add user code to column- and row-changing events of the data tables in the dataset. For more information about adding code to a dataset in an n-tier solution, see [Add code to datasets in n-tier applications](#), and [Add code to TableAdapters in n-tier applications](#). For more information about partial classes, see [How to: Split a class into partial classes \(Class Designer\)](#) or [Partial classes and methods](#).

NOTE

When you separate datasets from TableAdapters (by setting the **DataSet Project** property), existing partial dataset classes in the project won't be moved automatically. Existing partial dataset classes must be moved manually to the dataset project.

NOTE

The dataset Designer does not automatically create event handlers in C# for the [ColumnChanging](#) and [RowChanging](#) events. You have to manually create an event handler and hook up the event handler to the underlying event. The following procedures describe how to create the required event handlers in both Visual Basic and C#.

Validate changes to individual columns

Validate values in individual columns by handling the [ColumnChanging](#) event. The [ColumnChanging](#) event is raised when a value in a column is modified. Create an event handler for the [ColumnChanging](#) event by double-clicking the desired column on the **Dataset Designer**.

The first time that you double-click a column, the designer generates an event handler for the [ColumnChanging](#) event. An **If...Then** statement is also created that tests for the specific column. For example, the following code is generated when you double-click the **RequiredDate** column on the Northwind Orders table:

```
Private Sub OrdersDataTable_ColumnChanging(ByVal sender As System.Object, ByVal e As
System.Data.DataColumnChangeEventArgs) Handles Me.ColumnChanging
    If (e.Column.ColumnName = Me.RequiredDateColumn.ColumnName) Then
        ' Add validation code here.
    End If
End Sub
```

NOTE

In C# projects, the Dataset Designer only creates partial classes for the dataset and individual tables in the dataset. The Dataset Designer does not automatically create event handlers for the [ColumnChanging](#) and [RowChanging](#) events in C# like it does in Visual Basic. In C# projects, you have to manually construct a method to handle the event and hook up the method to the underlying event. The following procedure provides the steps to create the required event handlers in both Visual Basic and C#.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To add validation during changes to individual column values

1. Open the dataset by double-clicking the *.xsd* file in **Solution Explorer**. For more information, see [Walkthrough: Creating a dataset in the Dataset Designer](#).
2. Double-click the column you want to validate. This action creates the [ColumnChanging](#) event handler.

NOTE

The Dataset Designer does not automatically create an event handler for the C# event. The code that's necessary to handle the event in C# is included in the next section. `SampleColumnChangingEvent` is created and then hooked up to the [ColumnChanging](#) event in the [EndInit](#) method.

3. Add code to verify that `e.ProposedValue` contains data that meets the requirements of your application. If the proposed value is unacceptable, set the column to indicate that it contains an error.

The following code example validates that the **Quantity** column contains a value greater than 0. If **Quantity** is less than or equal to 0, the column is set to an error. The `Else` clause clears the error if **Quantity** is more than 0. The code in the column-changing event handler should resemble the following:

```
If (e.Column.ColumnName = Me.QuantityColumn.ColumnName) Then
    If CType(e.ProposedValue, Short) <= 0 Then
        e.Row.SetColumnError(e.Column, "Quantity must be greater than 0")
    Else
        e.Row.SetColumnError(e.Column, "")
    End If
End If
```

```
// Add this code to the DataTable partial class.
```

```
public override void EndInit()
{
    base.EndInit();
    // Hook up the ColumnChanging event
    // to call the SampleColumnChangingEvent method.
    ColumnChanging += SampleColumnChangingEvent;
}

public void SampleColumnChangingEvent(object sender, System.Data.DataColumnChangeEventArgs e)
{
    if (e.Column.ColumnName == QuantityColumn.ColumnName)
    {
        if ((short)e.ProposedValue <= 0)
        {
            e.Row.SetColumnError("Quantity", "Quantity must be greater than 0");
        }
        else
        {
            e.Row.SetColumnError("Quantity", "");
        }
    }
}
```

Validate changes to whole rows

Validate values in whole rows by handling the [RowChanging](#) event. The [RowChanging](#) event is raised when the values in all columns are committed. It is necessary to validate in the [RowChanging](#) event when the value in one column relies on the value in another column. For example, consider OrderDate and RequiredDate in the Orders table in Northwind.

When orders are being entered, validation makes sure that an order is not entered with a RequiredDate that is on or before the OrderDate. In this example, the values for both the RequiredDate and OrderDate columns need to be compared, so validating an individual column change does not make sense.

Create an event handler for the [RowChanging](#) event by double-clicking the table name in the title bar of the table on the **Dataset Designer**.

To add validation during changes to whole rows

1. Open the dataset by double-clicking the *.xsd* file in **Solution Explorer**. For more information, see [Walkthrough: Creating a dataset in the Dataset Designer](#).
2. Double-click the title bar of the data table on the designer.

A partial class is created with a `RowChanging` event handler and opens in the Code Editor.

NOTE

The Dataset Designer does not automatically create an event handler for the [RowChanging](#) event in C# projects. You have to create a method to handle the [RowChanging](#) event and run code then hook up the event in the table's initialization method.

3. Add user code inside the partial class declaration.
4. The following code shows where to add user code to validate during the [RowChanging](#) event. The C# example also includes code to hook the event handler method up to the `OrdersRowChanging` event.

```
Partial Class OrdersDataTable
    Private Sub OrdersDataTable_OrdersRowChanging(ByVal sender As System.Object, ByVal e As
OrdersRowChangeEvent) Handles Me.OrdersRowChanging
        ' Add logic to validate columns here.
        If e.Row.RequiredDate <= e.Row.OrderDate Then
            ' Set the RowError if validation fails.
            e.Row.RowError = "Required Date cannot be on or before the OrderDate"
        Else
            ' Clear the RowError when validation passes.
            e.Row.RowError = ""
        End If
    End Sub
End Sub
End Class
```

```

partial class OrdersDataTable
{
    public override void EndInit()
    {
        base.EndInit();
        // Hook up the event to the
        // RowChangingEvent method.
        OrdersRowChanging += RowChangingEvent;
    }

    public void RowChangingEvent(object sender, OrdersRowChangeEvent e)
    {
        // Perform the validation logic.
        if (e.Row.RequiredDate <= e.Row.OrderDate)
        {
            // Set the row to an error when validation fails.
            e.Row.RowError = "Required Date cannot be on or before the OrderDate";
        }
        else
        {
            // Clear the RowError if validation passes.
            e.Row.RowError = "";
        }
    }
}

```

See also

- [N-Tier data applications overview](#)
- [Walkthrough: Creating an N-Tier data application](#)
- [Validate data in datasets](#)

Add code to DataSets in n-tier applications

8/5/2021 • 2 minutes to read • [Edit Online](#)

You can extend the functionality of a dataset by creating a partial class file for the dataset and adding code to it (instead of adding code to the *DataSetName.Designer* file). Partial classes enable code for a specific class to be divided among multiple physical files. For more information, see [Partial](#) or [Partial classes and methods](#).

The code that defines a dataset is generated every time changes are made to the dataset definition (in the typed dataset). This code is also generated when you make changes during the running of any wizard that modifies the configuration of a dataset. To prevent your code from being deleted during the regeneration of a dataset, add code to the dataset's partial class file.

By default, after you separate the dataset and TableAdapter code, the result is a discrete class file in each project. The original project has a file named *DataSetName.Designer.vb* (or *DataSetName.Designer.cs*) that contains the TableAdapter code. The project that's designated in the **DataSet Project** property has a file that's named *DataSetName.DataSet.Designer.vb* (or *DataSetName.DataSet.Designer.cs*). This file contains the dataset code.

NOTE

When you separate DataSets and TableAdapters (by setting the **DataSet Project** property), existing partial dataset classes in the project won't be moved automatically. Existing dataset partial classes must be moved manually to the dataset project.

NOTE

When validation code needs to be added, the typed dataset provides functionality for generating [ColumnChanging](#) and [RowChanging](#) event handlers. For more information, see [Add validation to an n-tier dataset](#).

To add code to DataSets in n-tier applications

1. Locate the project that contains the *.xsd* file.
2. Select the *.xsd* file to open the dataset.
3. Right-click the data table to which you want to add code (the table name in the title bar), and then select **View Code**.

A partial class is created and opens in the Code Editor.

4. Add code inside the partial class declaration.

The following example shows where to add code to the CustomersDataTable in the NorthwindDataSet:

```
Partial Public Class CustomersDataTable
    ' Add code here to add functionality
    ' to the CustomersDataTable.
End Class
```

```
partial class CustomersDataTable
{
    // Add code here to add functionality
    // to the CustomersDataTable.
}
```

See also

- [N-Tier data applications overview](#)
- [Add code to TableAdapters in n-tier applications](#)
- [Create and configure TableAdapters](#)
- [Hierarchical update overview](#)
- [DataSet tools in Visual Studio](#)

Add code to TableAdapters in n-tier applications

8/5/2021 • 2 minutes to read • [Edit Online](#)

You can extend the functionality of a TableAdapter by creating a partial class file for the TableAdapter and adding code to it (instead of adding code to the *DataSetName.DataSet.Designer* file). Partial classes enable code for a specific class to be divided among multiple physical files. For more information, see [Partial](#) or [partial \(Type\)](#).

The code that defines a TableAdapter is generated every time changes are made to the TableAdapter in the dataset. This code is also generated when changes are made during the running of any wizard that modifies the configuration of the TableAdapter. To prevent your code from being deleted during the regeneration of a TableAdapter, add code to the partial class file of the TableAdapter.

By default, after you separate the dataset and TableAdapter code, the result is a discrete class file in each project. The original project has a file named *DataSetName.Designer.vb* (or *DataSetName.Designer.cs*) that contains the TableAdapter code. The project that's designated in the **DataSet Project** property has a file named *DataSetName.DataSet.Designer.vb* (or *DataSetName.DataSet.Designer.cs*) that contains the dataset code.

NOTE

When you separate datasets and TableAdapters (by setting the **DataSet Project** property), existing partial dataset classes in the project will not be moved automatically. Existing partial dataset classes must be moved manually to the dataset project.

NOTE

The dataset provides functionality for generating [ColumnChanging](#) and [RowChanging](#) event handlers when validation is needed. For more information, see [Add validation to an n-tier dataset](#).

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To add user code to a TableAdapter in an n-tier application

1. Locate the project that contains the *.xsd* file.
2. Double click the *.xsd* file to open the **Dataset Designer**.
3. Right-click the TableAdapter that you want to add code to, and then select **View Code**.

A partial class is created and opens in the Code Editor.

4. Add code inside the partial class declaration.
5. The following example shows where to add code to the `CustomersTableAdapter` in the `NorthwindDataSet`:

```
Partial Public Class CustomersTableAdapter
    ' Add code here to add functionality
    ' to the CustomersTableAdapter.
End Class
```

```
public partial class CustomersTableAdapter
{
    // Add code here to add functionality
    // to the CustomersTableAdapter.
}
```

See also

- [N-Tier data applications overview](#)
- [Add code to datasets in n-tier applications](#)
- [Create and configure TableAdapters](#)
- [Hierarchical update overview](#)

Separate datasets and TableAdapters into different projects

8/5/2021 • 2 minutes to read • [Edit Online](#)

Typed datasets have been enhanced so that the [TableAdapters](#) and dataset classes can be generated into separate projects. This enables you to quickly separate application layers and generate n-tier data applications.

The following procedure describes the process of using the **Dataset Designer** to generate dataset code into a project that is separate from the project that contains the generated TableAdapter code.

Separate datasets and TableAdapters

When you separate dataset code from TableAdapter code, the project that contains the dataset code must be located in the current solution. If this project is not located in the current solution, it won't be available in the **DataSet Project** list in the **Properties** window.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To separate the dataset into a different project

1. Open a solution that contains a dataset (.xsd file).

NOTE

If the solution does not contain the project into which you want to separate your dataset code, create the project, or add an existing project to the solution.

2. Double-click a typed dataset file (an .xsd file) in **Solution Explorer** to open the dataset in the **Dataset Designer**.
3. Select an empty area of the **Dataset Designer**.
4. In the **Properties** window, locate the **DataSet Project** node.
5. In the **DataSet Project** list, select the name of the project into which you want to generate the dataset code.

After you select the project into which you want to generate the dataset code, the **DataSet File** property is populated with a default file name. You can change this name if necessary. Additionally, if you want to generate the dataset code into a specific directory, you can set the **Project Folder** property to the name of a folder.

NOTE

When you separate datasets and TableAdapters (by setting the **DataSet Project** property), existing partial dataset classes in the project won't be moved automatically. Existing partial dataset classes must be moved manually to the dataset project.

6. Save the dataset.

The dataset code is generated into the selected project in the **DataSet Project** property, and the **TableAdapter** code is generated into the current project.

By default, after you separate the dataset and TableAdapter code, the result is a discrete class file in each project. The original project has a file named *DataSetName.Designer.vb* (or *DataSetName.Designer.cs*) that contains the TableAdapter code. The project that's designated in the **DataSet Project** property has a file named *DataSetName.DataSet.Designer.vb* (or *DataSetName.DataSet.Designer.cs*) that contains the dataset code.

NOTE

To view the generated class file, select the dataset or TableAdapter project. Then, in **Solution Explorer**, select **Show All Files**.

See also

- [N-tier data applications overview](#)
- [Walkthrough: Creating an N-tier data application](#)
- [Hierarchical update](#)
- [Accessing data in Visual Studio](#)
- [ADO.NET](#)

Bind controls to data in Visual Studio

8/5/2021 • 3 minutes to read • [Edit Online](#)

You can display data to users of your application by binding data to controls. You can create these data-bound controls by dragging items from the **Data Sources** window onto a design surface or controls on a surface in Visual Studio.

This topic describes the data sources you can use to create data-bound controls. It also describes some of the general tasks involved in data binding. For more specific details about how to create data-bound controls, see [Bind Windows Forms controls to data in Visual Studio](#) and [Bind WPF controls to data in Visual Studio](#).

Data sources

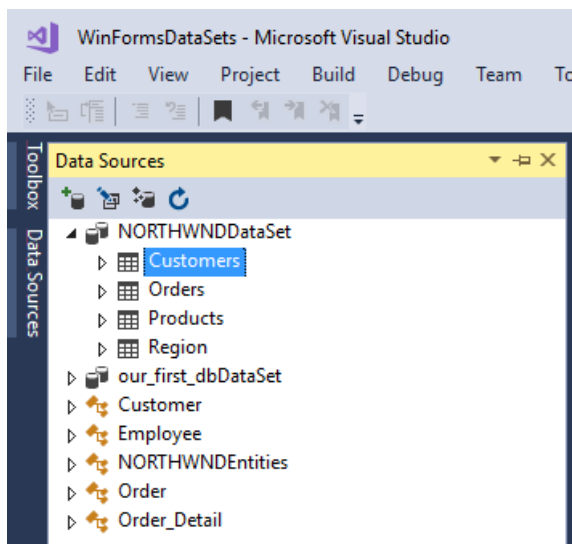
In the context of data binding, a data source represents the data in memory that can be bound to your user interface. In practical terms, a data source can be an Entity Framework class, a dataset, a service endpoint that is encapsulated in a .NET proxy object, a LINQ to SQL class, or any .NET object or collection. Some data sources enable you to create data-bound controls by dragging items from the **Data Sources** window, while other data sources do not. The following table shows which data sources are supported.

| DATA SOURCE | DRAG-AND-DROP SUPPORT IN THE WINDOWS FORMS DESIGNER | DRAG-AND-DROP SUPPORT IN THE WPF DESIGNER | DRAG-AND-DROP SUPPORT IN THE SILVERLIGHT DESIGNER |
|--|---|--|---|
| Dataset | Yes | Yes | No |
| Entity Data Model | Yes ¹ | Yes | Yes |
| LINQ to SQL classes | No ² | No ² | No ² |
| Services (including WCF Data Services, WCF services, and web services) | Yes | Yes | Yes |
| Object | Yes | Yes | Yes |
| SharePoint | Yes | Yes | Yes |

1. Generate the model using the **Entity Data Model** wizard, then drag those objects to the designer.
2. LINQ to SQL classes do not appear in the **Data Sources** window. However, you can add a new object data source that is based on LINQ to SQL classes, and then drag those objects to the designer to create data-bound controls. For more information, see [Walkthrough: Creating LINQ to SQL Classes \(O-R Designer\)](#).

Data Sources window

Data sources are available to your project as items in the **Data Sources** window. This window is visible when a form design surface is the active window in your project, or you can open it (when a project is open) by choosing **View** > **Other Windows** > **Data Sources**. You can drag items from this window to create controls that are bound to the underlying data, and you can also configure the data sources by right-clicking.



For each data type that appears in the **Data Sources** window, a default control is created when you drag the item to the designer. Before you drag an item from the **Data Sources** window, you can change the control that is created. For more information, see [Set the control to be created when dragging from the Data Sources window](#).

Tasks involved in binding controls to data

The following table lists some of the most common tasks you perform to bind controls to data.

| TASK | MORE INFORMATION |
|--|--|
| Open the Data Sources window. | Open a design surface in the editor and choose View > Data Sources . |
| Add a data source to your project. | Add new data sources |
| Set the control that is created when you drag an item from the Data Sources window to the designer. | Set the control to be created when dragging from the Data Sources window |
| Modify the list of controls that are associated with items in the Data Sources window. | Add custom controls to the Data Sources window |
| Create data-bound controls. | Bind Windows Forms controls to data in Visual Studio Bind WPF controls to data in Visual Studio |
| Bind to an object or collection. | Bind objects in Visual Studio |
| Filter data that appears in the UI. | Filter and sort data in a Windows Forms application |
| Customize captions for controls. | Customize how Visual Studio creates captions for data-bound controls |

See also

- [Visual Studio data tools for .NET](#)
- [Windows Forms data binding](#)

Set the control to be created when dragging from the Data Sources window

8/5/2021 • 3 minutes to read • [Edit Online](#)

You can create data-bound controls by dragging items from the **Data Sources** window onto the WPF designer or Windows Forms designer. Each item in the **Data Sources** window has a default control that is created when you drag it to the designer. However, you can choose to create a different control.

Set the controls to be created for data tables or objects

Before you drag items that represent data tables or objects from the **Data Sources** window, you can choose to display all the data in one control, or to display each column or property in a separate control.

In this context, the term *object* refers to a custom business object, an entity (in an Entity Data Model), or an object returned by a service.

To set the controls to be created for data tables or objects

1. Be sure the **WPF** designer or the **Windows Forms** designer is open.
2. In the **Data Sources** window, select the item that represents the data table or object you want to set.

TIP

If the **Data Sources** window is not open, you can open it by selecting **View > Other Windows > Data Sources**.

3. Click the drop-down menu for the item, and then click one of the following items in the menu:
 - To display each data field in a separate control, click **Details**. When you drag the data item to the designer, this action will create a different data-bound control for each column or property of the parent data table or object, along with labels for each control.
 - To display all of the data in a single control, select a different control in the list, such as **DataGrid** or **List** in a WPF application, or **DataGridView** in a Windows Forms application.

The list of available controls depends on which designer you have open, which version of .NET your project targets, and whether you have added custom controls that support data binding to the **Toolbox**. If the control you want to create is not in the list of available controls, you can add the control to the list. For more information, see [Add custom controls to the Data Sources window](#).

To learn how to create a custom Windows Forms control that can be added to the list of controls for data tables or objects in the **Data Sources** window, see [Create a Windows Forms user control that supports complex data binding](#).

Set the controls to be created for data columns or properties

Before you drag an item that represents a column or a property of an object from the **Data Sources** window to the designer, you can set the control to be created.

To set the controls to be created for columns or properties

1. Be sure the **WPF** designer or the **Windows Forms** designer is open.

2. In the **Data Sources** window, expand the desired table or object to display its columns or properties.
3. Select each column or property for which you want to set the control to be created.
4. Click the drop-down menu for the column or property, and then select the control you want to create when the item is dragged to the designer.

The list of available controls depends on which designer you have open, which version of .NET your project targets, and which custom controls that support data binding you have added to the **Toolbox**. If the control you want to create is in the list of available controls, you can add the control to the list. For more information, see [Add custom controls to the Data Sources window](#).

To learn how to create a custom control that can be added to the list of controls for data columns or properties in the **Data Sources** window, see [Create a Windows Forms user control that supports simple data binding](#).

If you don't want to create a control for the column or property, select **None** in the drop-down menu. This is useful if you want to drag the parent table or object to the designer, but you do not want to include the specific column or property.

See also

- [Bind controls to data in Visual Studio](#)

Add custom controls to the Data Sources window

8/5/2021 • 4 minutes to read • [Edit Online](#)

When you drag an item from the Data Sources window to a design surface to create a data-bound control, you can select the type of control that you create. Each item in the window has a drop-down list that displays the controls that you can choose from. The set of controls associated with each item is determined by the data type of the item. If the control that you want to create does not appear in the list, you can follow the instructions in this topic to add the control to the list.

For more information about selecting data-bound controls to create for items in the Data Sources window, see [Set the control to be created when dragging from the Data Sources window](#).

Customize the bindable controls list

To add or remove controls from the list of available controls for items in the Data Sources window that have a specific data type, perform the following steps.

To select the controls to be listed for a data type

1. Make sure that the WPF Designer or the Windows Forms Designer is open.
2. In the **Data Sources** window, click an item that is part of a data source you added to the window, and then click the drop-down menu for the item.

TIP

If the Data Sources window isn't open, open it by selecting **View > Other Windows > Data Sources**.

3. In the drop-down menu, click **Customize**. One of the following dialog boxes opens:
 - If the **Windows Forms Designer** is open, the **Data UI Customization** page of the **Options** dialog box opens. For more information, see [Data UI Customization options dialog box](#).
 - If the **WPF Designer** is open, the **Customize Control Binding** dialog box opens.
4. In the dialog box, select a data type from the **Data type** drop-down list.
 - To customize the list of controls for a table or object, select **[List]**.
 - To customize the list of controls for a column of a table or a property of an object, select the data type of the column or property in the underlying data store.
 - To customize the list of controls to display data objects that have user-defined shapes, select **[Other]**. For example, select **[Other]** if your application has a custom control that displays data from more than one property of a particular object.
5. In the **Associated controls** box, select each control that you want to be available for the selected data type, or clear the selection of any controls that you want to remove from the list.

NOTE

If the control that you want to select does not appear in the **Associated controls** box, you must add the control to the list. For more information, see [Add associated controls](#).

6. Click **OK**.
7. In the **Data Sources** window, click an item of the data type that you just associated one or more controls, and then click the drop-down menu for the item.

The controls you selected in the **Associated controls** box now appear in the drop-down menu for the item.

Add associated controls

If you want to associate a control with a data type, but the control does not appear in the **Associated controls** box, you must add the control to the list. The control must be located in the current solution or in a referenced assembly. It must also be available in the **Toolbox** and have an attribute that specifies the control's data binding behavior.

To add controls to the list of associated controls:

1. Add the desired control to the **Toolbox** by right-clicking the **Toolbox** and selecting **Choose Items**.

The control must have one of the following attributes:

| ATTRIBUTE | DESCRIPTION |
|---|---|
| DefaultBindingPropertyAttribute | Implement this attribute on simple controls that display a single column (or property) of data, such as a TextBox . |
| ComplexBindingPropertiesAttribute | Implement this attribute on controls that display lists (or tables) of data, such as a DataGridView . |
| LookupBindingPropertiesAttribute | Implement this attribute on controls that display lists (or tables) of data, but also need to present a single column or property, such as a ComboBox . |

2. For Windows Forms, on the **Options** dialog box, open the **Data UI Customization** page. Or, for WPF, open the **Customize Control Binding** dialog box. For more information, see [Customize the list of bindable controls for a data type](#).
3. In the **Associated controls** box, the control that you just added to the **Toolbox** should now appear.

NOTE

Only controls that are located within the current solution or in a referenced assembly can be added to the list of associated controls. (The controls must also implement one of the data-binding attributes in the previous table.) To bind data to a custom control that is not available in the Data Sources window, drag the control from the **Toolbox** onto the design surface, and then drag the item to bind to from the **Data Sources** window onto the control.

See also

- [Bind controls to data in Visual Studio](#)
- [Data UI Customization options dialog box](#)

Bind WPF controls to data in Visual Studio

8/5/2021 • 6 minutes to read • [Edit Online](#)

You can display data to users of your application by binding data to WPF controls. To create these data-bound controls, you can drag items from the **Data Sources** window onto the WPF Designer in Visual Studio. This topic describes some of the most common tasks, tools, and classes that you can use to create data-bound WPF applications.

For general information about how to create data-bound controls in Visual Studio, see [Bind controls to data in Visual Studio](#). For more information about WPF data binding, see [Data Binding Overview](#).

Tasks involved in binding WPF controls to data

The following table lists the tasks that can be accomplished by dragging items from the **Data Sources** window to the WPF Designer.

| TASK | MORE INFORMATION |
|---|---|
| Create new data-bound controls. Bind existing controls to data. | Bind WPF controls to a dataset |
| Create controls that display related data in a parent-child relationship: when the user selects a parent data record in one control, another control displays related child data for the selected record. | Display related data in WPF applications |
| Create a <i>lookup table</i> that displays information from one table based on the value of a foreign-key field in another table. | Create lookup tables in WPF applications |
| Bind a control to an image in a database. | Bind controls to pictures from a database |

Valid drop targets

You can drag items in the **Data Sources** window only to valid drop targets in the WPF Designer. There are two main kinds of valid drop targets: containers and controls. A container is a user interface element that typically contains controls. For example, a grid is a container, and so is a window.

Generated XAML and code

When you drag an item from the **Data Sources** window to the WPF Designer, Visual Studio generates XAML that defines a new data-bound control (or binds an existing control to the data source). For some data sources, Visual Studio also generates code in the code-behind file that fills the data source with data.

The following table lists the XAML and code that Visual Studio generates for each type of data source in the **Data Sources** window.

| DATA SOURCE | GENERATE XAML THAT BINDS A CONTROL TO THE DATA SOURCE | GENERATE CODE THAT FILLS THE DATA SOURCE WITH DATA |
|-------------------|---|--|
| Dataset | Yes | Yes |
| Entity Data Model | Yes | Yes |
| Service | Yes | No |
| Object | Yes | No |

Datasets

When you drag a table or column from the **Data Sources** window to the designer, Visual Studio generates XAML that does the following:

- Adds the dataset and a new [CollectionViewSource](#) to the resources of the container you dragged the item to. The [CollectionViewSource](#) is an object that can be used to navigate and display the data in the dataset.
- Creates a data binding for a control. If you drag the item to an existing control in the designer, the XAML binds the control to the item. If you drag the item to a container, the XAML creates the control that was selected for the dragged item, and it binds the control to the item. The control is created inside a new [Grid](#).

Visual Studio also makes the following changes to the code-behind file:

- Creates a [Loaded](#) event handler for the UI element that contains the control. The event handler fills the table with data, retrieves the [CollectionViewSource](#) from the container's resources, and then makes the first data item the current item. If a [Loaded](#) event handler already exists, Visual Studio adds this code to the existing event handler.

Entity data models

When you drag an entity or an entity property from the **Data Sources** window to the designer, Visual Studio generates XAML that does the following:

- Adds a new [CollectionViewSource](#) to the resources of the container you dragged the item to. The [CollectionViewSource](#) is an object that can be used to navigate and display the data in the entity.
- Creates a data binding for a control. If you drag the item to an existing control in the designer, the XAML binds the control to the item. If you drag the item to a container, the XAML creates the control that was selected for the dragged item, and it binds the control to the item. The control is created inside a new [Grid](#).

Visual Studio also makes the following changes to the code-behind file:

- Adds a new method that returns a query for the entity that you dragged to the designer (or the entity that contains the property that you dragged to the designer). The new method has the name `Get<EntityName>Query`, where `\<EntityName>` is the name of the entity.
- Creates a [Loaded](#) event handler for the UI element that contains the control. The event handler calls the `Get<EntityName>Query` method to fill the entity with data, retrieves the [CollectionViewSource](#) from the container's resources, and then makes the first data item the current item. If a [Loaded](#) event handler already exists, Visual Studio adds this code to the existing event handler.

Services

When you drag a service object or property from the **Data Sources** window to the designer, Visual Studio generates XAML that creates a data-bound control (or binds an existing control to the object or property).

However, Visual Studio does not generate code that fills the proxy service object with data. You must write this code yourself. For an example that demonstrates how to do this, see [Bind WPF controls to a WCF data service](#).

Visual Studio generates XAML that does the following:

- Adds a new [CollectionViewSource](#) to the resources of the container that you dragged the item to. The [CollectionViewSource](#) is an object that can be used to navigate and display the data in the object that is returned by the service.
- Creates a data binding for a control. If you drag the item to an existing control in the designer, the XAML binds the control to the item. If you drag the item to a container, the XAML creates the control that was selected for the dragged item, and it binds the control to the item. The control is created inside a new [Grid](#).

Objects

When you drag an object or property from the **Data Sources** window to the designer, Visual Studio generates XAML that creates a data-bound control (or binds an existing control to the object or property). However, Visual Studio does not generate code to fill the object with data. You must write this code yourself.

NOTE

Custom classes must be public and, by default, have a constructor without parameters. They can't be nested classes that have a "dot" in their syntax. For more information, see [XAML and custom classes for WPF](#).

Visual Studio generates XAML that does the following:

- Adds a new [CollectionViewSource](#) to the resources of the container that you dragged the item to. The [CollectionViewSource](#) is an object that can be used to navigate and display the data in the object.
- Creates a data binding for a control. If you drag the item to an existing control in the designer, the XAML binds the control to the item. If you drag the item to a container, the XAML creates the control that was selected for the dragged item, and it binds the control to the item. The control is created inside a new [Grid](#).

See also

- [Bind controls to data in Visual Studio](#)

Bind WPF controls to a dataset

8/5/2021 • 8 minutes to read • [Edit Online](#)

In this walkthrough, you create a WPF application that contains data-bound controls. The controls are bound to product records that are encapsulated in a dataset. You also add buttons to browse through products and save changes to product records.

This walkthrough illustrates the following tasks:

- Creating a WPF application and a dataset that is generated from data in the AdventureWorksLT sample database.
- Creating a set of data-bound controls by dragging a data table from the **Data Sources** window to a window in the WPF Designer.
- Creating buttons that navigate forward and backward through product records.
- Creating a button that saves changes that users make to the product records to the data table and the underlying data source.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

Prerequisites

You need the following components to complete this walkthrough:

- Visual Studio
- Access to a running instance of SQL Server or SQL Server Express that has the AdventureWorks Light (AdventureWorksLT) sample database attached to it. You can download the AdventureWorksLT database from the [CodePlex archive](#).

Prior knowledge of the following concepts is also helpful, but not required to complete the walkthrough:

- Datasets and TableAdapters. For more information, see [Dataset tools in Visual Studio](#) and [TableAdapters](#).
- WPF data binding. For more information, see [Data Binding Overview](#).

Create the project

Create a new WPF project to display product records.

1. Open Visual Studio.
2. On the **File** menu, select **New > Project**.
3. Expand **Visual Basic** or **Visual C#**, and then select **Windows**.
4. Select the **WPF App** project template.
5. In the **Name** box, enter **AdventureWorksProductsEditor** and then select **OK**.

1. Open Visual Studio.
2. On the start window, choose **Create a new project**.
3. Search for the C# **WPF App** project template and follow the steps to create the project, naming the project **AdventureWorksProductsEditor**.

Visual Studio creates the AdventureWorksProductsEditor project.

Create a dataset for the application

Before you can create data-bound controls, you must define a data model for your application and add it to the **Data Sources** window. In this walkthrough, you create a dataset to use as the data model.

1. On the **Data** menu, click **Show Data Sources**.

The **Data Sources** window opens.

2. In the **Data Sources** window, click **Add New Data Source**.

The **Data Source Configuration** wizard opens.

3. On the **Choose a Data Source Type** page, select **Database**, and then click **Next**.
4. On the **Choose a Database Model** page, select **Dataset**, and then click **Next**.
5. On the **Choose Your Data Connection** page, select one of the following options:
 - If a data connection to the AdventureWorksLT sample database is available in the drop-down list, select it and then click **Next**.
 - Click **New Connection**, and create a connection to the AdventureWorksLT database.
6. On the **Save the Connection String to the Application Configure File** page, select the **Yes, save the connection as** check box, and then click **Next**.
7. On the **Choose Your Database Objects** page, expand **Tables**, and then select the **Product (SalesLT)** table.
8. Click **Finish**.

Visual Studio adds a new `AdventureWorksLTDataSet.xsd` file to the project, and it adds a corresponding **AdventureWorksLTDataSet** item to the **Data Sources** window. The `AdventureWorksLTDataSet.xsd` file defines a typed dataset named `AdventureWorksLTDataSet` and a TableAdapter named `ProductTableAdapter`. Later in this walkthrough, you will use the `ProductTableAdapter` to fill the dataset with data and save changes back to the database.

9. Build the project.

Edit the default fill method of the TableAdapter

To fill the dataset with data, use the `Fill` method of the `ProductTableAdapter`. By default, the `Fill` method fills the `ProductDataTable` in the `AdventureWorksLTDataSet` with all rows of data from the Product table. You can modify this method to return only a subset of the rows. For this walkthrough, modify the `Fill` method to return only rows for products that have photos.

1. In **Solution Explorer**, double-click the *AdventureWorksLTDataSet.xsd* file.

The Dataset designer opens.

2. In the designer, right-click the **Fill, GetData()** query and select **Configure**.

The **TableAdapter Configuration** wizard opens.

3. In the **Enter a SQL Statement** page, add the following WHERE clause after the `SELECT` statement in the text box.

```
WHERE ThumbnailPhotoFileName <> 'no_image_available_small.gif'
```

4. Click **Finish**.

Define the user interface

Add several buttons to the window by modifying the XAML in the WPF Designer. Later in this walkthrough, you will add code that enables users to scroll through and save changes to products records by using these buttons.

1. In **Solution Explorer**, double-click *MainWindow.xaml*.

The window opens in the **WPF Designer**.

2. In the XAML view of the designer, add the following code between the `<Grid>` tags:

```
<Grid.RowDefinitions>
    <RowDefinition Height="75" />
    <RowDefinition Height="625" />
</Grid.RowDefinitions>
<Button HorizontalAlignment="Left" Margin="22,20,0,24" Name="backButton" Width="75">&lt;</Button>
<Button HorizontalAlignment="Left" Margin="116,20,0,24" Name="nextButton" Width="75">&gt;</Button>
<Button HorizontalAlignment="Right" Margin="0,21,46,24" Name="saveButton" Width="110">Save
changes</Button>
```

3. Build the project.

Create data-bound controls

Create controls that display customer records by dragging the `Product` table from the **Data Sources** window to the WPF Designer.

1. In the **Data Sources** window, click the drop-down menu for the **Product** node and select **Details**.
2. Expand the **Product** node.
3. For this example, some fields will not be displayed, so click the drop-down menu next to the following nodes and select **None**:
 - ProductCategoryID
 - ProductModelID
 - ThumbnailPhotoFileName
 - rowguid
 - ModifiedDate
4. Click the drop-down menu next to the **ThumbNailPhoto** node and select **Image**.

NOTE

By default, items in the **Data Sources** window that represent pictures have their default control set to **None**. This is because pictures are stored as byte arrays in databases, and byte arrays can contain anything from a simple array of bytes to the executable file of a large application.

5. From the **Data Sources** window, drag the **Product** node to the grid row under the row that contains the buttons.

Visual Studio generates XAML that defines a set of controls that are bound to data in the **Products** table. It also generates code that loads the data. For more information about the generated XAML and code, see [Bind WPF controls to data in Visual Studio](#).

6. In the designer, click the text box next to the **Product ID** label.
7. In the **Properties** window, select the check box next to the **IsReadOnly** property.

Navigate product records

Add code that enables users to scroll through product records by using the < and > buttons.

1. In the designer, double-click the < button on the window surface.

Visual Studio opens the code-behind file, and creates a new `backButton_Click` event handler for the [Click](#) event.

2. Modify the `Window_Loaded` event handler, so the `ProductViewSource`, `AdventureWorksLTDataSet`, and `AdventureWorksLTDataSetProductTableAdapter` are outside of the method and accessible to the entire form. Declare only these to be global to the form, and assign them within the `Window_Loaded` event handler similar to the following:

```
private AdventureWorksProductsEditor.AdventureWorksLTDataSet AdventureWorksLTDataSet;
private AdventureWorksProductsEditor.AdventureWorksLTDataSetTableAdapters.ProductTableAdapter
adventureWorksLTDataSetProductTableAdapter;
private System.Windows.Data.CollectionViewSource productViewSource;

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    AdventureWorksLTDataSet = ((AdventureWorksProductsEditor.AdventureWorksLTDataSet)
(this.FindResource("adventureWorksLTDataSet")));
    // Load data into the table Product. You can modify this code as needed.
    adventureWorksLTDataSetProductTableAdapter = new
AdventureWorksProductsEditor.AdventureWorksLTDataSetTableAdapters.ProductTableAdapter();
    adventureWorksLTDataSetProductTableAdapter.Fill(AdventureWorksLTDataSet.Product);
    productViewSource = ((System.Windows.Data.CollectionViewSource)
(this.FindResource("productViewSource")));
    productViewSource.View.MoveCurrentToFirst();
}
```

```

Dim ProductViewSource As System.Windows.Data.CollectionViewSource
Dim AdventureWorksLTDataSet As AdventureWorksProductsEditor.AdventureWorksLTDataSet
Dim AdventureWorksLTDataSetProductTableAdapter As
AdventureWorksProductsEditor.AdventureWorksLTDataSetTableAdapters.ProductTableAdapter

Private Sub Window_Loaded(ByVal sender As System.Object, ByVal e As System.Windows.RoutedEventArgs)
Handles MyBase.Loaded
    AdventureWorksLTDataSet = CType(Me.FindResource("AdventureWorksLTDataSet"),
AdventureWorksProductsEditor.AdventureWorksLTDataSet)
    'Load data into the table Product. You can modify this code as needed.
    AdventureWorksLTDataSetProductTableAdapter = New
AdventureWorksProductsEditor.AdventureWorksLTDataSetTableAdapters.ProductTableAdapter()
    AdventureWorksLTDataSetProductTableAdapter.Fill(AdventureWorksLTDataSet.Product)
    ProductViewSource = CType(Me.FindResource("ProductViewSource"),
System.Windows.Data.CollectionViewSource)
    ProductViewSource.View.MoveCurrentToFirst()
End Sub

```

3. Add the following code to the `backButton_Click` event handler:

```

if (productViewSource.View.CurrentPosition > 0)
{
    productViewSource.View.MoveCurrentToPrevious();
}

```

```

If ProductViewSource.View.CurrentPosition > 0 Then
    ProductViewSource.View.MoveCurrentToPrevious()
End If

```

4. Return to the designer and double-click the > button.
5. Add the following code to the `nextButton_Click` event handler:

```

if (productViewSource.View.CurrentPosition < ((CollectionView)productViewSource.View).Count - 1)
{
    productViewSource.View.MoveCurrentToNext();
}

```

```

If ProductViewSource.View.CurrentPosition < CType(ProductViewSource.View, CollectionView).Count - 1
Then
    ProductViewSource.View.MoveCurrentToNext()
End If

```

Save changes to product records

Add code that enables users to save changes to product records by using the **Save changes** button.

1. In the designer, double-click the **Save changes** button.

Visual Studio opens the code-behind file, and creates a new `saveButton_Click` event handler for the [Click](#) event.

2. Add the following code to the `saveButton_Click` event handler:

```

adventureWorksLTDataSetProductTableAdapter.Update(AdventureWorksLTDataSet.Product);

```



```
AdventureWorksLTDataSetProductTableAdapter.Update(AdventureWorksLTDataSet.Product)
```

NOTE

This example uses the `Save` method of the `TableAdapter` to save the changes. This is appropriate in this walkthrough, because only one data table is being changed. If you need to save changes to multiple data tables, you can alternatively use the `UpdateAll` method of the `TableAdapterManager` that Visual Studio generates with your dataset. For more information, see [TableAdapters](#).

Test the application

Build and run the application. Verify that you can view and update product records.

1. Press **F5**.

The application builds and runs. Verify the following:

- The text boxes display data from the first product record that has a photo. This product has the product ID 713, and the name **Long-Sleeve Logo Jersey, S**.
 - You can click the **>** or **<** buttons to navigate through other product records.
2. In one of the product records, change the **Size** value, and then click **Save changes**.
 3. Close the application, and then restart the application by pressing **F5** in Visual Studio.
 4. Navigate to the product record you changed, and verify that the change persisted.
 5. Close the application.

Next steps

After completing this walkthrough, you might try the following related tasks:

- Learn how to use the **Data Sources** window in Visual Studio to bind WPF controls to other types of data sources. For more information, see [Bind WPF controls to a WCF data service](#).
- Learn how to use the **Data Sources** window in Visual Studio to display related data (that is, data in a parent-child relationship) in WPF controls. For more information, see [Walkthrough: Display related data in a WPF app](#).

See also

- [Bind WPF controls to data in Visual Studio](#)
- [Dataset tools in Visual Studio](#)
- [Data Binding Overview](#)

Bind WPF controls to a WCF data service

8/5/2021 • 9 minutes to read • [Edit Online](#)

In this walkthrough, you will create a WPF application that contains data-bound controls. The controls are bound to customer records that are encapsulated in a WCF Data Service. You will also add buttons that customers can use to view and update records.

This walkthrough illustrates the following tasks:

- Creating an Entity Data Model that is generated from data in the AdventureWorksLT sample database.
- Creating a WCF Data Service that exposes the data in the Entity Data Model to a WPF application.
- Creating a set of data-bound controls by dragging items from the **Data Sources** window to the WPF designer.
- Creating buttons that navigate forward and backward through customer records.
- Creating a button that saves changes to data in the controls to the WCF Data Service and the underlying data source.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

Prerequisites

You need the following components to complete this walkthrough:

- Visual Studio
- Access to a running instance of SQL Server or SQL Server Express that has the AdventureWorksLT sample database attached to it. You can download the AdventureWorksLT database from the [CodePlex website](#).

Prior knowledge of the following concepts is also helpful, but not required to complete the walkthrough:

- [WCF Data Services](#).
- Data models in WCF Data Services.
- Entity Data Models and the ADO.NET Entity Framework. For more information, see [Entity Framework overview](#).
- WPF data binding. For more information, see [Data Binding overview](#).

Create the service project

1. Start this walkthrough by creating a C# or Visual Basic **ASP.NET Web Application** project. Name the project **AdventureWorksService**.
2. In **Solution Explorer**, right-click **Default.aspx** and select **Delete**. This file is not necessary for the

walkthrough.

Create an Entity Data Model for the service

To expose data to an application by using a WCF Data Service, you must define a data model for the service. The WCF Data Service supports two types of data models: Entity Data Models, and custom data models that are defined by using common language runtime (CLR) objects that implement the [IQueryable<T>](#) interface. In this walkthrough, you create an Entity Data Model for the data model.

1. On the **Project** menu, click **Add New Item**.
2. In the Installed Templates list, click **Data**, and then select the **ADO.NET Entity Data Model** project item.
3. Change the name to `AdventureWorksModel.edmx`, and click **Add**.

The **Entity Data Model** wizard opens.

4. On the **Choose Model Contents** page, click **Generate from database**, and click **Next**.
5. On the **Choose Your Data Connection** page, select one of the following options:
 - If a data connection to the AdventureWorksLT sample database is available in the drop-down list, select it.
 - Click **New Connection**, and create a connection to the AdventureWorksLT database.
6. On the **Choose Your Data Connection** page, make sure that the **Save entity connection settings in App.Config** as option is selected, and then click **Next**.
7. On the **Choose Your Database Objects** page, expand **Tables**, and then select the **SalesOrderHeader** table.
8. Click **Finish**.

Create the service

Create a WCF Data Service to expose the data in the Entity Data Model to a WPF application:

1. On the **Project** menu, select **Add New Item**.
2. In the **Installed Templates** list, click **Web**, and then select the **WCF Data Service** project item.
3. In the **Name** box, type `AdventureWorksService.svc`, and click **Add**.

Visual Studio adds the `AdventureWorksService.svc` to the project.

Configure the service

You must configure the service to operate on the Entity Data Model that you created:

1. In the `AdventureWorks.svc` code file, replace the **AdventureWorksService** class declaration with the following code.

```

public class AdventureWorksService : DataService<AdventureWorksLTEntities>
{
    // This method is called only once to initialize service-wide policies.
    public static void InitializeService(IDataServiceConfiguration config)
    {
        config.SetEntitySetAccessRule("SalesOrderHeaders", EntitySetRights.All);
    }
}

```

```

Public Class AdventureWorksService
    Inherits DataService(Of AdventureWorksLTEntities)

    ' This method is called only once to initialize service-wide policies.
    Public Shared Sub InitializeService(ByVal config As IDataServiceConfiguration)
        config.SetEntitySetAccessRule("SalesOrderHeaders", EntitySetRights.All)
        config.UseVerboseErrors = True
    End Sub
End Class

```

This code updates the **AdventureWorksService** class, so that it derives from a [DataService<T>](#) that operates on the `AdventureWorksLTEntities` object context class in your Entity Data Model. It also updates the `InitializeService` method to allow clients of the service full read/write access to the `SalesOrderHeader` entity.

2. Build the project, and verify that it builds without errors.

Create the WPF client application

To display the data from the WCF Data Service, create a new WPF application with a data source that is based on the service. Later in this walkthrough, you will add data-bound controls to the application.

1. In **Solution Explorer**, right-click the solution node, click **Add**, and select **New Project**.
2. In the **New Project** dialog, expand **Visual C#** or **Visual Basic**, and then select **Windows**.
3. Select the **WPF Application** project template.
4. In the **Name** box, type `AdventureWorksSalesEditor`, and click **OK**.

Visual Studio adds the `AdventureWorksSalesEditor` project to the solution.

5. On the **Data** menu, click **Show Data Sources**.

The **Data Sources** window opens.

6. In the **Data Sources** window, click **Add New Data Source**.

The **Data Source Configuration** wizard opens.

7. In the **Choose a Data Source Type** page of the wizard, select **Service**, and then click **Next**.
8. In the **Add Service Reference** dialog box, click **Discover**.

Visual Studio searches the current solution for available services, and adds `AdventureWorksService.svc` to the list of available services in the **Services** box.

9. In the **Namespace** box, type **AdventureWorksService**.
10. In the **Services** box, click **AdventureWorksService.svc**, and then click **OK**.

Visual Studio downloads the service information and then returns to the **Data Source Configuration**

wizard.

11. In the **Add Service Reference** page, click **Finish**.

Visual Studio adds nodes that represent the data returned by the service to the **Data Sources** window.

Define the user interface

Add several buttons to the window by modifying the XAML in the WPF designer. Later in this walkthrough, you will add code that enables users to view and update sales records by using these buttons.

1. In **Solution Explorer**, double-click **MainWindow.xaml**.

The window opens in the WPF designer.

2. In the XAML view of the designer, add the following code between the `<Grid>` tags:

```
<Grid.RowDefinitions>
    <RowDefinition Height="75" />
    <RowDefinition Height="525" />
</Grid.RowDefinitions>
<Button HorizontalAlignment="Left" Margin="22,20,0,24" Name="backButton" Width="75"><</Button>
<Button HorizontalAlignment="Left" Margin="116,20,0,24" Name="nextButton" Width="75">></Button>
<Button HorizontalAlignment="Right" Margin="0,21,46,24" Name="saveButton" Width="110">Save
changes</Button>
```

3. Build the project.

Create the data-bound controls

Create controls that display customer records by dragging the `SalesOrderHeaders` node from the **Data Sources** window to the designer.

1. In the **Data Sources** window, click the drop-down menu for the **SalesOrderHeaders** node, and select **Details**.
2. Expand the **SalesOrderHeaders** node.
3. For this example, some fields will not be displayed, so click the drop-down menu next to the following nodes and select **None**:

- **CreditCardApprovalCode**
- **ModifiedDate**
- **OnlineOrderFlag**
- **RevisionNumber**
- **rowguid**

This action prevents Visual Studio from creating data-bound controls for these nodes in the next step. For this walkthrough, assume that the end user does not need to see this data.

4. From the **Data Sources** window, drag the **SalesOrderHeaders** node to the grid row under the row that contains the buttons.

Visual Studio generates XAML and code that creates a set of controls that are bound to data in the **Product** table. For more information about the generated XAML and code, see [Bind WPF controls to data in Visual Studio](#).

5. In the designer, click the text box next to the **Customer ID** label.
6. In the **Properties** window, select the check box next to the **IsReadOnly** property.
7. Set the **IsReadOnly** property for each of the following text boxes:
 - **Purchase Order Number**
 - **Sales Order ID**
 - **Sales Order Number**

Load the data from the service

Use the service proxy object to load sales data from the service. Then assign the returned data to the data source for the [CollectionViewSource](#) in the WPF window.

1. In the designer, to create the `Window_Loaded` event handler, double-click the text that reads: **MainWindow**.
2. Replace the event handler with the following code. Make sure that you replace the *localhost* address in this code with the local host address on your development computer.

```
private AdventureWorksService.AdventureWorksLTEntities dataServiceClient;
private System.Data.Services.Client.DataServiceQuery<AdventureWorksService.SalesOrderHeader>
salesQuery;
private CollectionViewSource ordersViewSource;

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    // TODO: Modify the port number in the following URI as required.
    dataServiceClient = new AdventureWorksService.AdventureWorksLTEntities(
        new Uri("http://localhost:45899/AdventureWorksService.svc"));
    salesQuery = dataServiceClient.SalesOrderHeaders;

    ordersViewSource = ((CollectionViewSource)(this.FindResource("salesOrderHeadersViewSource")));
    ordersViewSource.Source = salesQuery.Execute();
    ordersViewSource.View.MoveCurrentToFirst();
}
```

```
Private DataServiceClient As AdventureWorksService.AdventureWorksLTEntities
Private SalesQuery As System.Data.Services.Client.DataServiceQuery(Of
AdventureWorksService.SalesOrderHeader)
Private OrdersViewSource As CollectionViewSource

Private Sub Window_Loaded(ByVal Sender As Object, ByVal e As RoutedEventArgs) Handles MyBase.Loaded

    ' TODO: Modify the port number in the following URI as required.
    DataServiceClient = New AdventureWorksService.AdventureWorksLTEntities( _
    New Uri("http://localhost:32415/AdventureWorksService.svc"))
    SalesQuery = DataServiceClient.SalesOrderHeaders

    OrdersViewSource = CType(Me.FindResource("SalesOrderHeadersViewSource"), CollectionViewSource)
    OrdersViewSource.Source = SalesQuery.Execute()
    OrdersViewSource.View.MoveCurrentToFirst()
End Sub
```

Navigate sales records

Add code that enables users to scroll through sales records by using the < and > buttons.

1. In the designer, double-click the < button on the window surface.

Visual Studio opens the code-behind file, and creates a new `backButton_Click` event handler for the [Click](#) event.

2. Add the following code to the generated `backButton_Click` event handler:

```
if (ordersViewSource.View.CurrentPosition > 0)
    ordersViewSource.View.MoveCurrentToPrevious();
```

```
If OrdersViewSource.View.CurrentPosition > 0 Then
    OrdersViewSource.View.MoveCurrentToPrevious()
End If
```

3. Return to the designer, and double-click the > button.

Visual Studio opens the code-behind file, and creates a new `nextButton_Click` event handler for the [Click](#) event.

4. Add the following code to the generated `nextButton_Click` event handler:

```
if (ordersViewSource.View.CurrentPosition < ((CollectionView)ordersViewSource.View).Count - 1)
{
    ordersViewSource.View.MoveCurrentToNext();
}
```

```
If OrdersViewSource.View.CurrentPosition < CType(OrdersViewSource.View, CollectionView).Count - 1
Then
    OrdersViewSource.View.MoveCurrentToNext()
End If
```

Save changes to sales records

Add code that enables users to both view and save changes to sales records by using the **Save changes** button:

1. In the designer, double-click the **Save Changes** button.

Visual Studio opens the code-behind file, and creates a new `saveButton_Click` event handler for the [Click](#) event.

2. Add the following code to the `saveButton_Click` event handler.

```
AdventureWorksService.SalesOrderHeader currentOrder =
(AdventureWorksService.SalesOrderHeader)ordersViewSource.View.CurrentItem;
dataServiceClient.UpdateObject(currentOrder);
dataServiceClient.SaveChanges();
```

```
Dim CurrentOrder As AdventureWorksService.SalesOrderHeader = CType(OrdersViewSource.View.CurrentItem,
AdventureWorksService.SalesOrderHeader)
```

```
DataServiceClient.UpdateObject(CurrentOrder)
DataServiceClient.SaveChanges()
```

Test the application

Build and run the application to verify that you can view and update customer records:

1. On **Build** menu, click **Build Solution**. Verify that the solution builds without errors.
2. Press **Ctrl+F5**.

Visual Studio starts the **AdventureWorksService** project, without debugging it.

3. In **Solution Explorer**, right-click the **AdventureWorksSalesEditor** project.
4. On the right-click menu (context menu), under **Debug**, click **Start new instance**.

The application runs. Verify the following:

- The text boxes display different fields of data from the first sales record, which has the sales order ID **71774**.
 - You can click the > or < buttons to navigate through other sales records.
5. In one of the sales records, type some text in the **Comment** box, and then click **Save changes**.
 6. Close the application, and then start the application again from Visual Studio.
 7. Navigate to the sales record that you changed, and verify that the change persists after you close and reopen the application.
 8. Close the application.

Next steps

After completing this walkthrough, you can perform the following related tasks:

- Learn how to use the **Data Sources** window in Visual Studio to bind WPF controls to other types of data sources. For more information, see [Bind WPF controls to a dataset](#).
- Learn how to use the **Data Sources** window in Visual Studio to display related data (that is, data in a parent-child relationship) in WPF controls. For more information, see [Walkthrough: Displaying related data in a WPF application](#).

See also

- [Bind WPF controls to data in Visual Studio](#)
- [Bind WPF controls to a dataset](#)
- [WCF overview \(.NET Framework\)](#)
- [Entity Framework overview \(.NET Framework\)](#)
- [Data Binding overview \(.NET Framework\)](#)

Create lookup tables in WPF applications

8/5/2021 • 4 minutes to read • [Edit Online](#)

The term *lookup table* (sometimes called a *lookup binding*) describes a control that displays information from one data table based on the value of a foreign-key field in another table. You can create a lookup table by dragging the main node of a parent table or object in the **Data Sources** window onto a control that is already bound to a column or property in a related child table.

For example, consider a table of `Orders` in a sales database. Each record in the `Orders` table includes a `CustomerID` that indicates which customer placed the order. The `CustomerID` is a foreign key that points to a customer record in the `Customers` table. When you display a list of orders from the `Orders` table, you may want to display the actual customer name instead of the `CustomerID`. Because the customer name is in the `Customers` table, you need to create a lookup table to display the customer name. The lookup table uses the `CustomerID` value in the `Orders` record to navigate the relationship, and return the customer name.

To create a lookup table

1. Add one of the following types of data sources with related data to your project:

- Dataset or Entity Data Model.
- WCF Data Service, WCF service or web service. For more information, see [How to: Connect to Data in a Service](#).
- Objects. For more information, see [Bind to objects in Visual Studio](#).

NOTE

Before you can create a lookup table, two related tables or objects must exist as a data source for the project.

2. Open the **WPF Designer**, and make sure that the designer contains a container that is a valid drop target for items in the **Data Sources** window.

For more information about valid drop targets, see [Bind WPF controls to data in Visual Studio](#).

3. On the **Data** menu, click **Show Data Sources** to open the **Data Sources** window.

4. Expand the nodes in the **Data Sources** window, until you can see the parent table or object and the related child table or object.

NOTE

The related child table or object is the node that appears as an expandable child node under the parent table or object.

5. Click the drop-down menu for the child node, and select **Details**.

6. Expand the child node.

7. Under the child node, click the drop-down menu for the item that relates the child and parent data. (In the preceding example, this is the **CustomerID** node.) Select one of the following types of controls that support lookup binding:

- **ComboBox**
- **ListBox**
- **ListView**

NOTE

If the **ListBox** or **ListView** control does not appear in the list, you can add these controls to the list. For information, see [Set the control to be created when dragging from the Data Sources window](#).

- Any custom control that derives from [Selector](#).

NOTE

For information about how to add custom controls to the list of controls you can select for items in the **Data Sources** window, see [Add custom controls to the Data Sources window](#).

8. Drag the child node from the **Data Sources** window onto a container in the WPF designer. (In the preceding example, the child node is the **Orders** node.)

Visual Studio generates XAML that creates new data-bound controls for each of the items that you drag. The XAML also adds a new [CollectionViewSource](#) for the child table or object to the resources of the drop target. For some data sources, Visual Studio also generates code to load data into the table or object. For more information, see [Bind WPF controls to data in Visual Studio](#).

9. Drag the parent node from the **Data Sources** window onto the lookup binding control that you created earlier. (In the preceding example, the parent node is the **Customers** node).

Visual Studio sets some properties on the control to configure the lookup binding. The following table lists the properties that Visual Studio modifies. If necessary, you can change these properties in the XAML or in the **Properties** window.

| PROPERTY | EXPLANATION OF SETTING |
|-----------------------------------|--|
| ItemsSource | This property specifies the collection or binding that is used to get the data that is displayed in the control. Visual Studio sets this property to the CollectionViewSource for the parent data you dragged to the control. |
| DisplayMemberPath | <p>This property specifies the path of the data item that is displayed in the control. Visual Studio sets this property to the first column or property in the parent data, after the primary key, that has a string data type.</p> <p>If you want to display a different column or property in the parent data, change this property to the path of a different property.</p> |
| SelectedValue | Visual Studio binds this property to the column or property of the child data that you dragged to the designer. This is the foreign key to the parent data. |
| SelectedValuePath | Visual Studio sets this property to the path of the column or property of the child data that is the foreign key to the parent data. |

See also

- [Bind WPF controls to data in Visual Studio](#)
- [Display related data in WPF applications](#)
- [Walkthrough: Displaying related data in a WPF application](#)

Display related data in WPF applications

8/5/2021 • 2 minutes to read • [Edit Online](#)

In some applications, you might want to work with data that comes from multiple tables or entities that are related to each other in a parent-child relationship. For example, you might want to display a grid that displays customers from a `Customers` table. When the user selects a specific customer, another grid displays the orders for that customer from a related `Orders` table.

You can create data-bound controls that display related data by dragging items from the **Data Sources** window to the WPF Designer.

To create controls that display related records

1. On the **Data** menu, click **Show Data Sources** to open the **Data Sources** window.
2. Click **Add New Data Source**, and complete the **Data Source Configuration** wizard.
3. Open the WPF designer, and make sure that the designer contains a container that is a valid drop target for the items in the **Data Sources** window.

For more information about valid drop targets, see [Bind WPF controls to data in Visual Studio](#).

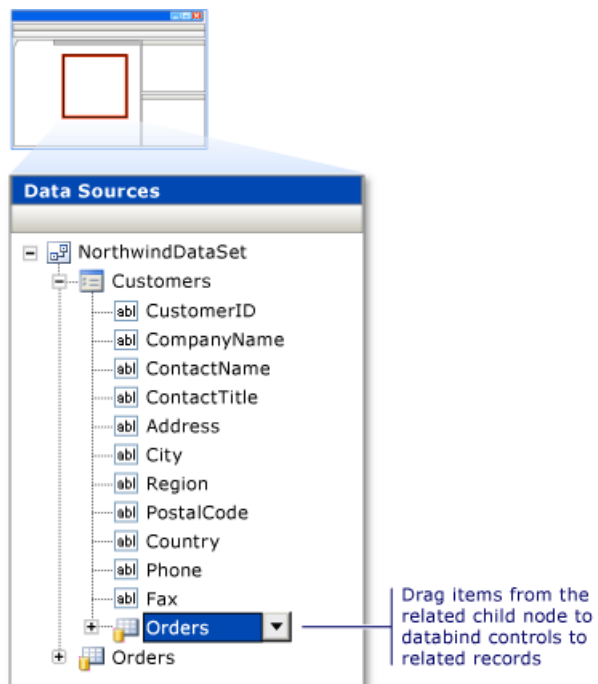
4. In the **Data Sources** window, expand the node that represents the parent table or object in the relationship. The parent table or object is on the "one" side of a one-to-many relationship.
5. Drag the parent node (or any individual items in the parent node) from the **Data Sources** window onto a valid drop target in the designer.

Visual Studio generates XAML that creates new data-bound controls for each item that you drag. The XAML also adds a new `CollectionViewSource` for the parent table or object to the resources of the drop target. For some data sources, Visual Studio also generates code to load the data into the parent table or object. For more information, see [Bind WPF controls to data in Visual Studio](#).

6. In the **Data Sources** window, locate the related child table or object. Related child tables and objects appear as expandable nodes at the bottom of the parent node's list of data.
7. Drag the child node (or any individual items in the child node) from the **Data Sources** window onto a valid drop target in the designer.

Visual Studio generates XAML that creates new data-bound controls for each of the items you drag. The XAML also adds a new `CollectionViewSource` for the child table or object to the resources of the drop target. This new `CollectionViewSource` is bound to the property of the parent table or object that you just dragged to the designer. For some data sources, Visual Studio also generates code to load the data into the child table or object.

The following figure demonstrates the related `Orders` table of the `Customers` table in a dataset in the **Data Sources** window.



See also

- [Bind WPF controls to data in Visual Studio](#)
- [Create lookup tables in WPF applications](#)

Bind controls to pictures from a database

8/5/2021 • 2 minutes to read • [Edit Online](#)

You can use the **Data Sources** window to bind an image in a database to a control in your application. For example, you can bind an image to an [Image](#) control in a WPF application, or to a [PictureBox](#) control in a Windows Forms application.

Pictures in a database are typically stored as byte arrays. Items in the **Data Sources** window that are stored as byte arrays have their control type set to **None** by default, because byte arrays can contain anything from a simple array of bytes to the executable file of a large application. To create a data-bound control for a byte array item in the **Data Sources** window that represents an image, you must select the control to create.

The following procedure assumes that the **Data Sources** window is already populated with an item that is bound to your image.

To bind a picture in a database to a control

1. Make sure that the design surface you want to add the control to is open in the WPF Designer or the Windows Forms Designer.
2. In the **Data Sources** window, expand the desired table or object to display its columns or properties.

TIP

If the **Data Sources** window isn't open, open it by selecting **View > Other Windows > Data Sources**.

3. Select the column or property that contains your image data, and select one of the following controls from its drop-down control list:
 - If the WPF designer is open, select **Image**.
 - If the Windows Forms designer is open, select **PictureBox**.
 - Alternatively, you can select a different control that supports data binding and that can display images. If the control that you want to use is not in the list of available controls, you can add it to the list and then select it. For more information, see [Add custom controls to the Data Sources window](#).

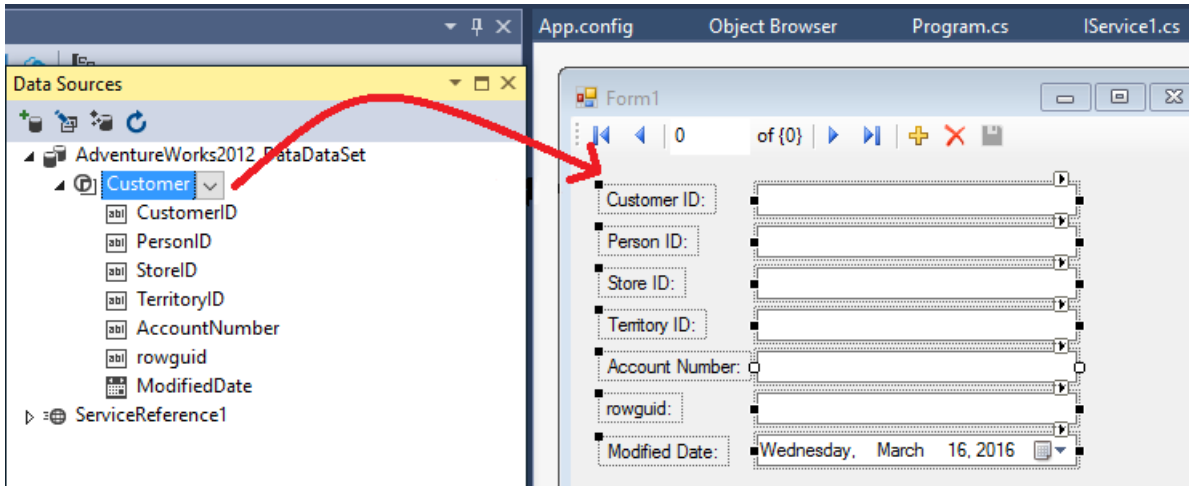
See also

- [Bind WPF controls to data in Visual Studio](#)

Bind Windows Forms controls to data in Visual Studio

8/5/2021 • 3 minutes to read • [Edit Online](#)

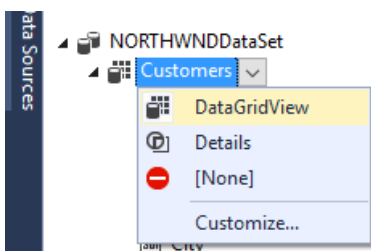
You can display data to users of your application by binding data to Windows Forms. To create these data-bound controls, drag items from the **Data Sources** window onto the Windows Forms Designer in Visual Studio.



TIP

If the **Data Sources** window is not visible, you can open it by choosing **View > Other Windows > Data Sources**, or by pressing **Shift+Alt+D**. You must have a project open in Visual Studio to see the **Data Sources** window.

Before you drag items, you can set the type of control you want to bind to. Different values appear depending on whether you choose the table itself, or an individual column. You can also set custom values. For a table, **Details** means that each column is bound to a separate control.



BindingSource and BindingNavigator controls

The **BindingSource** component serves two purposes. First, it provides a layer of abstraction when binding the controls to data. Controls on the form are bound to the **BindingSource** component instead of directly to a data source. Second, it can manage a collection of objects. Adding a type to the **BindingSource** creates a list of that type.

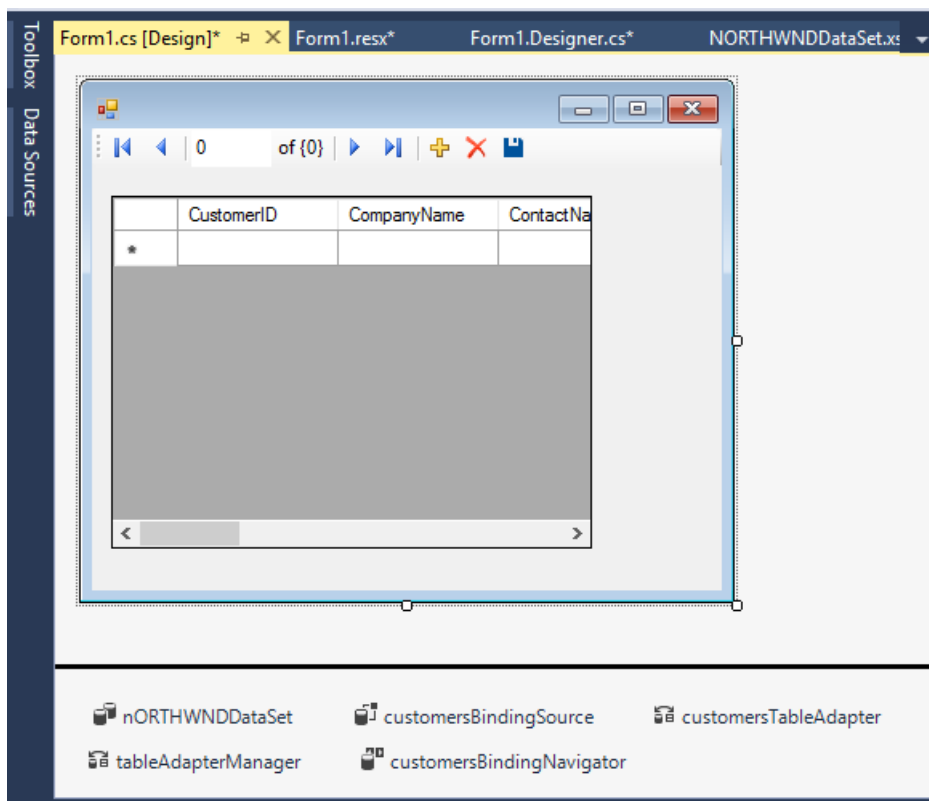
For more information about the **BindingSource** component, see:

- [BindingSource component](#)
- [BindingSource component overview](#)
- [BindingSource component architecture](#)

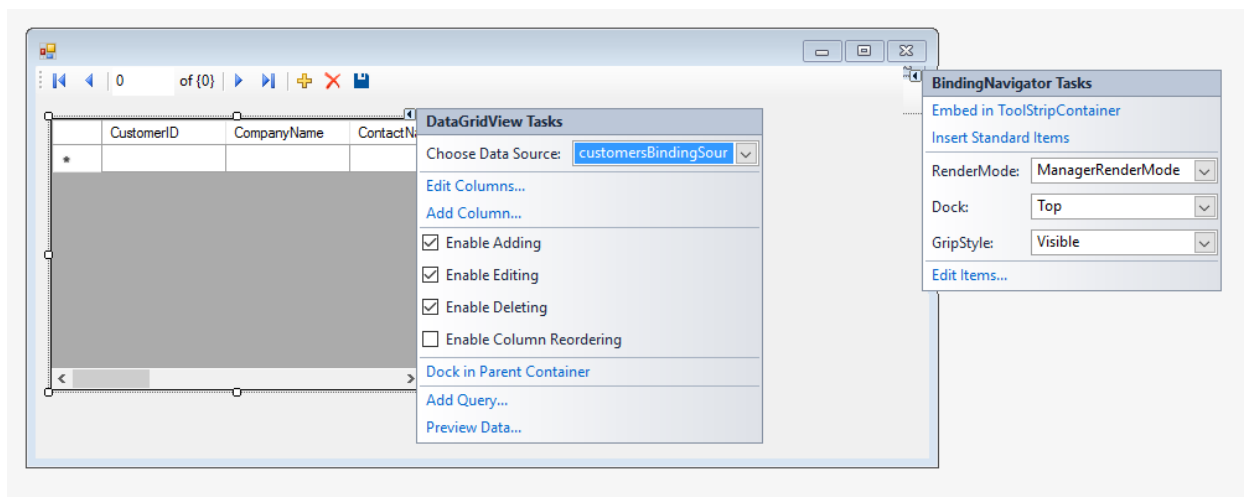
The [BindingNavigator control](#) provides a user interface for navigating through data displayed by a Windows application.

Bind to data in a DataGridView control

For a [DataGridView control](#), the entire table is bound to that single control. When you drag a **DataGridView** to the form, a tool strip for navigating records ([BindingNavigator](#)) also appears. A [DataSet](#), [TableAdapter](#), [BindingSource](#), and [BindingNavigator](#) appear in the component tray. In the following illustration, a [TableAdapterManager](#) is also added because the Customers table has a relation to the Orders table. These variables are all declared in the auto-generated code as private members in the form class. The auto-generated code for filling the **DataGridView** is located in the `Form_Load` event handler. The code for saving the data to update the database is located in the `Save` event handler for the **BindingNavigator**. You can move or modify this code as needed.



You can customize the behavior of the **DataGridView** and the **BindingNavigator** by clicking on the smart tag in the upper-right corner of each:

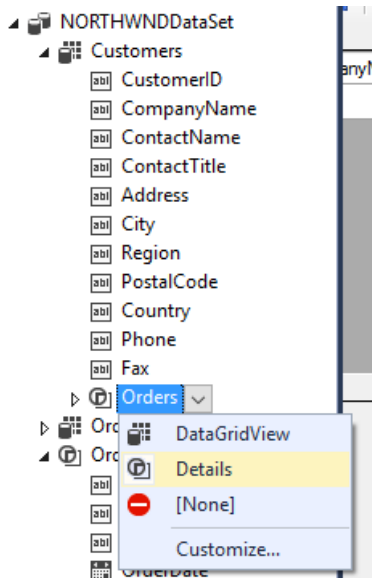


If the controls your application needs are not available from within the **Data Sources** window, you can add controls. For more information, see [Add custom controls to the Data Sources window](#).

You can also drag items from the **Data Sources** window onto controls already on a form to bind the control to data. A control that is already bound to data has its data bindings reset to the item most recently dragged onto it. To be valid drop targets, controls must be capable of displaying the underlying data type of the item dragged onto it from the **Data Sources** window. For example, it's not valid to drag an item that has a data type of [DateTime](#) onto a [CheckBox](#), because the [CheckBox](#) is not capable of displaying a date.

Bind to data in individual controls

When you bind a data source to **Details**, each column in the dataset is bound to a separate control.



IMPORTANT

Note that in the previous illustration, you drag from the Orders property of the Customers table, not from the Orders table. By binding to the `Customer.Orders` property, navigation commands made in the **DataGridView** are reflected immediately in the details controls. If you dragged from the Orders table, the controls would still be bound to the dataset, but not they would not be synchronized with the **DataGridView**.

The following illustration shows the default data-bound controls that are added to the form after the Orders property in the Customers table is bound to **Details** in the **Data Sources** window.

A screenshot of a form generated from the data source binding. The form contains a series of data-bound controls for the 'Orders' table. These include text boxes for 'Order ID', 'Customer ID', 'Employee ID', 'Ship Via', 'Freight', 'Ship Name', 'Ship Address', 'Ship City', 'Ship Region', 'Ship Postal Code', and 'Ship Country'. Date controls are shown for 'Order Date', 'Required Date', and 'Shipped Date', each displaying a date like 'Wednesday, February 24, 2016'. A context menu is open over the 'Order ID' text box, showing options: 'MultiLine' (unchecked), 'Add Query...', and 'Preview Data...'. The 'Order ID' text box also has a small smart tag icon in its top-left corner.

Note also that each control has a smart tag. This tag enables customizations that apply to that control only.

See also

- [Binding controls to data in Visual Studio](#)
- [Data binding in Windows Forms \(.NET Framework\)](#)

Filter and sort data in a Windows Forms application

8/5/2021 • 2 minutes to read • [Edit Online](#)

You filter data by setting the [Filter](#) property to a string expression that returns the desired records.

You sort data by setting the [Sort](#) property to the column name on which you want to sort; append `DESC` to sort in descending order, or append `ASC` to sort in ascending order.

NOTE

If your application does not use [BindingSource](#) components, you can filter and sort data by using [DataView](#) objects. For more information, see [DataViews](#).

To filter data by using a BindingSource component

- Set the [Filter](#) property to the expression you want to return. For example, the following code returns customers with a `CompanyName` that starts with "B":

```
customersBindingSource.Filter = "CompanyName like 'B'";
```

```
CustomersBindingSource.Filter = "CompanyName like 'B'"
```

To sort data by using a BindingSource component

- Set the [Sort](#) property to the column you want to sort on. For example, the following code sorts customers on the `CompanyName` column in descending order:

```
customersBindingSource.Sort = "CompanyName Desc";
```

```
CustomersBindingSource.Sort = "CompanyName Desc"
```

See also

- [Bind controls to data in Visual Studio](#)

Commit in-process edits on data-bound controls before saving data

8/5/2021 • 2 minutes to read • [Edit Online](#)

When editing values in data-bound controls, users must navigate off the current record to commit the updated value to the underlying data source that the control is bound to. When you drag items from the [Data Sources Window](#) onto a form, the first item that you drop generates code into the **Save** button click event of the [BindingNavigator](#). This code calls the [EndEdit](#) method of the [BindingSource](#). Therefore, the call to the [EndEdit](#) method is generated only for the first [BindingSource](#) that is added to the form.

The [EndEdit](#) call commits any changes that are in process, in any data-bound controls that are currently being edited. Therefore, if a data-bound control still has focus and you click the **Save** button, all pending edits in that control are committed before the actual save (the `TableAdapterManager.UpdateAll` method).

You can configure your application to automatically commit changes, even if a user tries to save data without committing the changes, as part of the save process.

NOTE

The designer adds the `BindingSource.EndEdit` code only for the first item dropped onto a form. Therefore, you have to add a line of code to call the [EndEdit](#) method for each [BindingSource](#) on the form. You can manually add a line of code to call the [EndEdit](#) method for each [BindingSource](#). Alternatively, you can add the `EndEditOnAllBindingSources` method to the form and call it before you perform a save.

The following code uses a [LINQ \(Language-Integrated Query\)](#) query to iterate all [BindingSource](#) components and call the [EndEdit](#) method for each [BindingSource](#) on a form.

To call EndEdit for all BindingSource components on a form

1. Add the following code to the form that contains the [BindingSource](#) components.

```
private void EndEditOnAllBindingSources()
{
    var BindingSourcesQuery =
        from Component bindingSources in this.components.Components
        where bindingSources is BindingSource
        select bindingSources;

    foreach (BindingSource bindingSource in BindingSourcesQuery)
    {
        bindingSource.EndEdit();
    }
}
```

```
Private Sub EndEditOnAllBindingSources()  
    Dim BindingSourcesQuery = From bindingsources In Me.components.Components  
        Where (TypeOf bindingsources Is Windows.Forms.BindingSource)  
        Select bindingsources  
  
    For Each bindingSource As Windows.Forms.BindingSource In BindingSourcesQuery  
        bindingSource.EndEdit()  
    Next  
End Sub
```

2. Add the following line of code immediately before any calls to save the form's data (the

`TableAdapterManager.UpdateAll()` method):

```
EndEditOnAllBindingSources();
```

```
Me.EndEditOnAllBindingSources()
```

See also

- [Bind Windows Forms controls to data in Visual Studio](#)
- [Hierarchical update](#)

Create lookup tables in Windows Forms applications

8/5/2021 • 3 minutes to read • [Edit Online](#)

The term *lookup table* describes controls that are bound to two related data tables. These lookup controls display data from the first table based on a value selected in the second table.

You can create lookup tables by dragging the main node of a parent table (from the [Data Sources window](#)) onto a control on your form that is already bound to the column in the related child table.

For example, consider a table of `Orders` in a sales database. Each record in the `Orders` table includes a `CustomerID`, indicating which customer placed the order. The `CustomerID` is a foreign key pointing to a customer record in the `Customers` table. In this scenario, you expand the `Orders` table in the **Data Sources** window and set the main node to **Details**. Then, set the `CustomerID` column to use a [ComboBox](#) (or any other control that supports lookup binding), and drag the `Orders` node onto your form. Finally, drag the `Customers` node onto the control that is bound to the related column — in this case, the [ComboBox](#) bound to the `CustomerID` column.

To databind a lookup control

1. With your project open, open the **Data Sources** window by choosing **View > Other Windows > Data Sources**.

NOTE

Lookup tables require that two related tables or objects are available in the **Data Sources** window. For more information, see [Relationships in datasets](#).

2. Expand the nodes in the **Data Sources** window until you can see the parent table and all of its columns, and the related child table and all of its columns.

NOTE

The child table node is the node that appears as an expandable child node in the parent table.

3. Change the drop type of the child table to **Details** by selecting **Details** from the control list on the child table's node. For more information, see [Set the control to be created when dragging from the Data Sources window](#).
4. Locate the node that relates the two tables (the `CustomerID` node in the previous example). Change its drop type to a [ComboBox](#) by selecting **ComboBox** from the control list.
5. Drag the main child table node from the **Data Sources** window onto your form.

Databound controls (with descriptive labels) and a tool strip ([BindingNavigator](#)) appear on the form. A [DataSet](#), [TableAdapter](#), [BindingSource](#), and [BindingNavigator](#) appear in the component tray.

6. Now, drag the main parent table node from the **Data Sources** window directly onto the lookup control (the [ComboBox](#)).

The lookup bindings are now established. Refer to the following table for the specific properties that were

set on the control.

| PROPERTY | EXPLANATION OF SETTING |
|----------------------|--|
| DataSource | <p>Visual Studio sets this property to the BindingSource, created for the table you drag onto the control (as opposed to the BindingSource, created when the control was created).</p> <p>If you need to make an adjustment, set this to the BindingSource of the table with the column you want to display.</p> |
| DisplayMember | <p>Visual Studio sets this property to the first column after the primary key that has a string data type for the table you drag onto the control.</p> <p>If you need to make an adjustment, set this to the column name you want to display.</p> |
| ValueMember | <p>Visual Studio sets this property to the first column participating in the primary key, or the first column in the table if no key is defined.</p> <p>If you need to make an adjustment, set this to the primary key in the table with the column you want to display.</p> |
| SelectedValue | <p>Visual Studio sets this property to the original column dropped from the Data Sources window.</p> <p>If you need to make an adjustment, set this to the foreign-key column in the related table.</p> |

See also

- [Bind Windows Forms controls to data in Visual Studio](#)

Create a Windows Form to search data

8/5/2021 • 6 minutes to read • [Edit Online](#)

A common application scenario is to display selected data on a form. For example, you might want to display the orders for a specific customer or the details of a specific order. In this scenario, a user enters information into a form, and then a query is executed with the user's input as a parameter; that is, the data is selected based on a parameterized query. The query returns only the data that satisfies the criteria entered by the user. This walkthrough shows how to create a query that returns customers in a specific city, and modify the user interface so that users can enter a city's name and press a button to execute the query.

Using parameterized queries helps make your application efficient by letting the database do the work it is best at — quickly filtering records. In contrast, if you request an entire database table, transfer it over the network, and then use application logic to find the records you want, your application can become slow and inefficient.

You can add parameterized queries to any `TableAdapter` (and controls to accept parameter values and execute the query), using the **Search Criteria Builder** dialog box. Open the dialog box by selecting the **Add Query** command on the **Data** menu (or on any `TableAdapter` smart tag).

Tasks illustrated in this walkthrough include:

- Creating and configuring the data source in your application with the **Data Source Configuration** wizard.
- Setting the drop type of the items in the **Data Sources** window.
- Creating controls that display data by dragging items from the **Data Sources** window onto a form.
- Adding controls to display the data on the form.
- Completing the **Search Criteria Builder** dialog box.
- Entering parameters into the form and executing the parameterized query.

NOTE

The procedures in this article apply only to .NET Framework Windows Forms projects, not to .NET Core Windows Forms projects.

Prerequisites

You must have the **Data storage and processing** workload installed. See [Modify Visual Studio](#).

This walkthrough uses SQL Server Express LocalDB and the Northwind sample database.

1. If you don't have SQL Server Express LocalDB, install it either from the [SQL Server Express download page](#), or through the **Visual Studio Installer**. In the **Visual Studio Installer**, you can install SQL Server Express LocalDB as part of the **Data storage and processing** workload, or as an individual component.
2. Install the Northwind sample database by following these steps:
 - a. In Visual Studio, open the **SQL Server Object Explorer** window. (SQL Server Object Explorer is installed as part of the **Data storage and processing** workload in the **Visual Studio Installer**.) Expand the **SQL Server** node. Right-click on your LocalDB instance and select **New Query**.

A query editor window opens.

- b. Copy the [Northwind Transact-SQL script](#) to your clipboard. This T-SQL script creates the Northwind database from scratch and populates it with data.
- c. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes running and the Northwind database is created.

Create the Windows Forms application

Create a new **Windows Forms App (.NET Framework)** project for either C# or Visual Basic. Name the project **WindowsSearchForm**.

Create the data source

This step creates a data source from a database using the **Data Source Configuration** wizard:

1. To open the **Data Sources** window, on the **Data** menu, click **Show Data Sources**.
2. In the **Data Sources** window, select **Add New Data Source** to start the **Data Source Configuration** wizard.
3. Select **Database** on the **Choose a Data Source Type** page, and then click **Next**.
4. On the **Choose your Data Connection** page do one of the following:
 - If a data connection to the Northwind sample database is available in the drop-down list, select it.
 - Select **New Connection** to launch the **Add/Modify Connection** dialog box.
5. If your database requires a password, select the option to include sensitive data, and then click **Next**.
6. On the **Save connection string to the Application Configuration file** page, click **Next**.
7. On the **Choose your Database Objects** page, expand the **Tables** node.
8. Select the **Customers** table, and then click **Finish**.

The **NorthwindDataSet** is added to your project, and the **Customers** table appears in the **Data Sources** window.

Create a new **Windows Forms App (.NET Framework)** project for either C# or Visual Basic. Name the project **WindowsSearchForm**.

Create the data source

This step creates a data source from a database using the **Data Source Configuration** wizard:

1. To open the **Data Sources** window, use quick search (Ctrl+Q), and search for **Data Sources**.
2. In the **Data Sources** window, select **Add New Data Source** to start the **Data Source Configuration** wizard.
3. Select **Database** on the **Choose a Data Source Type** page, and then click **Next**.
4. On the **Choose a Database Model** screen, choose **Dataset**, and then click **Next**.
5. On the **Choose your Data Connection** page do one of the following:
 - If a data connection to the Northwind sample database is available in the drop-down list, select it.

- Select **New Connection** to launch the **Add/Modify Connection** dialog box.
6. On the **Save connection string to the Application Configuration file** page, click **Next**.
 7. On the **Choose your Database Objects** page, expand the **Tables** node.
 8. Select the **Customers** table, and then click **Finish**.

The **NorthwindDataSet** is added to your project, and the **Customers** table appears in the **Data Sources** window.

Create the form

You can create the data-bound controls by dragging items from the **Data Sources** window onto your form:

1. Make sure the Windows Forms designer has the active focus and the **Data Sources** window is open and pinned.
2. Expand the **Customers** node in the **Data Sources** window.
3. Drag the **Customers** node from the **Data Sources** window to your form.

A **DataGridView** and a tool strip (**BindingNavigator**) for navigating records appear on the form. A **NorthwindDataSet**, **CustomersTableAdapter**, **BindingSource**, and **BindingNavigator** appear in the component tray.

Add parameterization (search functionality) to the query

You can add a WHERE clause to the original query using the **Search Criteria Builder** dialog box:

1. Just below the design surface for your form, select the **customersTableAdapter** button, and then in the **Properties** window, choose **Add Query....**
2. Type **FillByCity** in the **New query name** area on the **Search Criteria Builder** dialog box.
3. Add `WHERE City = @City` to the query in the **Query Text** area.

The query should be similar to the following:

```
SELECT CustomerID, CompanyName, ContactName, ContactTitle,
       Address, City, Region, PostalCode, Country, Phone, Fax
FROM Customers
WHERE City = @City
```

NOTE

Access and OLE DB data sources use the question mark (?) to denote parameters, so the WHERE clause would look like this: `WHERE City = ?`.

4. Click **OK** to close the **Search Criteria Builder** dialog box.

A **FillByCityToolStrip** is added to the form.

Test the application

Running the application opens your form and makes it ready to take the parameter as input:

1. Press **F5** to run the application.

2. Type **London** into the **City** text box, and then click **FillByCity**.

The data grid is populated with customers that meet the criteria. In this example, the data grid only displays customers that have a value of **London** in their **City** column.

Next steps

Depending on your application requirements, there are several steps you may want to perform after creating a parameterized form. Some enhancements you could make to this walkthrough include:

- Adding controls that display related data. For more information, see [Relationships in Datasets](#).
- Editing the dataset to add or remove database objects. For more information, see [Create and configure datasets](#).

See also

- [Bind Windows Forms controls to data in Visual Studio](#)

Create a Windows Forms user control that supports simple data binding

8/5/2021 • 6 minutes to read • [Edit Online](#)

When displaying data on forms in Windows applications, you can choose existing controls from the **Toolbox**, or you can author custom controls if your application requires functionality that is not available in the standard controls. This walkthrough shows how to create a control that implements the [DefaultBindingPropertyAttribute](#). Controls that implement the [DefaultBindingPropertyAttribute](#) can contain one property that can be bound to data. Such controls are similar to a [TextBox](#) or [CheckBox](#).

For more information on control authoring, see [Developing Windows Forms Controls at Design Time](#).

When authoring controls for use in data-binding scenarios, you should implement one of the following data-binding attributes:

DATA-BINDING ATTRIBUTE USAGE

Implement the [DefaultBindingPropertyAttribute](#) on simple controls, like a [TextBox](#), that display a single column (or property) of data. (This process is described in this walkthrough page.)

Implement the [ComplexBindingPropertiesAttribute](#) on controls, like a [DataGridView](#), that display lists (or tables) of data. For more information, see [Create a Windows Forms user control that supports complex data binding](#).

Implement the [LookupBindingPropertiesAttribute](#) on controls, like a [ComboBox](#), that display lists (or tables) of data but also need to present a single column or property. For more information, see [Create a Windows Forms user control that supports lookup data binding](#).

This walkthrough creates a simple control that displays data from a single column in a table. This example uses the `Phone` column of the `Customers` table from the Northwind sample database. The simple user control displays customers' phone numbers in a standard phone-number format, by using a [MaskedTextBox](#) and setting the mask to a phone number.

During this walkthrough, you will learn how to:

- Create a new **Windows Forms Application**.
- Add a new **User Control** to your project.
- Visually design the user control.
- Implement the `DefaultBindingProperty` attribute.
- Create a dataset with the **Data Source Configuration** wizard.
- Set the **Phone** column in the **Data Sources** window to use the new control.
- Create a form to display data in the new control.

Prerequisites

This walkthrough uses SQL Server Express LocalDB and the Northwind sample database.

1. If you don't have SQL Server Express LocalDB, install it either from the [SQL Server Express download page](#), or through the **Visual Studio Installer**. In the **Visual Studio Installer**, you can install SQL

Server Express LocalDB as part of the **Data storage and processing** workload, or as an individual component.

2. Install the Northwind sample database by following these steps:

- a. In Visual Studio, open the **SQL Server Object Explorer** window. (SQL Server Object Explorer is installed as part of the **Data storage and processing** workload in the **Visual Studio Installer**.) Expand the **SQL Server** node. Right-click on your LocalDB instance and select **New Query**.

A query editor window opens.

- b. Copy the [Northwind Transact-SQL script](#) to your clipboard. This T-SQL script creates the Northwind database from scratch and populates it with data.

- c. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes running and the Northwind database is created.

Create a Windows Forms Application

The first step is to create a **Windows Forms Application**:

1. In Visual Studio, on the **File** menu, select **New > Project**.
2. Expand either **Visual C#** or **Visual Basic** in the left-hand pane, then select **Windows Desktop**.
3. In the middle pane, select the **Windows Forms App** project type.
4. Name the project **SimpleControlWalkthrough**, and then choose **OK**.

The **SimpleControlWalkthrough** project is created, and added to **Solution Explorer**.

Add a user control to the project

This walkthrough creates a simple data-bindable control from a **User Control**. Add a **User Control** item to the **SimpleControlWalkthrough** project:

1. From the **Project** menu, choose **Add User Control**.
2. Type **PhoneNumberBox** in the Name area, and click **Add**.

The **PhoneNumberBox** control is added to **Solution Explorer**, and opens in the designer.

Design the PhoneNumberBox control

This walkthrough expands upon the existing [MaskedTextBox](#) to create the **PhoneNumberBox** control:

1. Drag a [MaskedTextBox](#) from the **Toolbox** onto the user control's design surface.
2. Select the smart tag on the [MaskedTextBox](#) you just dragged, and choose **Set Mask**.
3. Select **Phone number** in the **Input Mask** dialog box, and click **OK** to set the mask.

Add the required data-binding attribute

For simple controls that support databinding, implement the [DefaultBindingPropertyAttribute](#):

1. Switch the **PhoneNumberBox** control to code view. (On the **View** menu, choose **Code**.)
2. Replace the code in the **PhoneNumberBox** with the following:

```

using System.Windows.Forms;

namespace CS
{
    [System.ComponentModel.DefaultBindingProperty("PhoneNumber")]
    public partial class PhoneNumberBox : UserControl
    {
        public string PhoneNumber
        {
            get{ return maskedTextBox1.Text; }
            set{ maskedTextBox1.Text = value; }
        }

        public PhoneNumberBox()
        {
            InitializeComponent();
        }
    }
}

```

```

<System.ComponentModel.DefaultBindingProperty("PhoneNumber")>
Public Class PhoneNumberBox

    Public Property PhoneNumber() As String
        Get
            Return MaskedTextBox1.Text
        End Get
        Set(ByVal value As String)
            MaskedTextBox1.Text = value
        End Set
    End Property
End Class

```

3. From the **Build** menu, choose **Build Solution**.

Create a data source from your database

This step uses the **Data Source Configuration** wizard to create a data source based on the **Customers** table in the Northwind sample database. You must have access to the Northwind sample database to create the connection. For information on setting up the Northwind sample database, see [How to: Install sample databases](#).

1. To open the **Data Sources** window, on the **Data** menu, click **Show Data Sources**.
2. In the **Data Sources** window, select **Add New Data Source** to start the **Data Source Configuration** wizard.
3. On the **Choose a Data Source Type** page, select **Database**, and then click **Next**.
4. On the **Choose your Data Connection** page, do one of the following:
 - If a data connection to the Northwind sample database is available in the drop-down list, select it.
 - Select **New Connection** to launch the **Add/Modify Connection** dialog box.
5. If your database requires a password, select the option to include sensitive data, and then click **Next**.
6. On the **Save connection string to the Application Configuration file** page, click **Next**.
7. On the **Choose your Database Objects** page, expand the **Tables** node.
8. Select the **Customers** table, and then click **Finish**.

The **NorthwindDataSet** is added to your project, and the **Customers** table appears in the **Data Sources** window.

Set the phone column to use the PhoneNumberBox control

Within the **Data Sources** window, you can set the control to be created prior to dragging items onto your form:

1. Open **Form1** in the designer.
2. Expand the **Customers** node in the **Data Sources** window.
3. Click the drop-down arrow on the **Customers** node, and choose **Details** from the control list.
4. Click the drop-down arrow on the **Phone** column, and choose **Customize**.
5. Select the **PhoneNumberBox** from the list of **Associated Controls** in the **Data UI Customization Options** dialog box.
6. Click the drop-down arrow on the **Phone** column, and choose **PhoneNumberBox**.

Add controls to the form

You can create the data-bound controls by dragging items from the **Data Sources** window onto the form.

To create data-bound controls on the form, drag the main **Customers** node from the **Data Sources** window onto the form, and verify that the **PhoneNumberBox** control is used to display the data in the **Phone** column.

Data-bound controls with descriptive labels appear on the form, along with a tool strip ([BindingNavigator](#)) for navigating records. A [NorthwindDataSet](#), [CustomersTableAdapter](#), [BindingSource](#), and [BindingNavigator](#) appear in the component tray.

Run the application

Press **F5** to run the application.

Next steps

Depending on your application requirements, there are several steps you may want to perform after creating a control that supports data binding. Some typical next steps include:

- Placing your custom controls in a control library so you can reuse them in other applications.
- Creating controls that support more complex data-binding scenarios. For more information, see [Create a Windows Forms user control that supports complex data binding](#) and [Create a Windows Forms user control that supports lookup data binding](#).

See also

- [Bind Windows Forms controls to data in Visual Studio](#)
- [Set the control to be created when dragging from the Data Sources window](#)

Create a Windows Forms user control that supports complex data binding

8/5/2021 • 5 minutes to read • [Edit Online](#)

When displaying data on forms in Windows applications, you can choose existing controls from the **Toolbox**. Or, you can author custom controls if your application requires functionality that is not available in the standard controls. This walkthrough shows how to create a control that implements the [ComplexBindingPropertiesAttribute](#). Controls that implement the [ComplexBindingPropertiesAttribute](#) contain a `DataSource` and `DataMember` property that can be bound to data. Such controls are similar to a [DataGridView](#) or [ListBox](#).

For more information on control authoring, see [Developing Windows Forms controls at design time](#).

When authoring controls for use in data-binding scenarios you need to implement one of the following data-binding attributes:

DATA-BINDING ATTRIBUTE USAGE

Implement the [DefaultBindingPropertyAttribute](#) on simple controls, like a [TextBox](#), that display a single column (or property) of data. For more information, see [Create a Windows Forms user control that supports simple data binding](#).

Implement the [ComplexBindingPropertiesAttribute](#) on controls, like a [DataGridView](#), that display lists (or tables) of data. (This process is described in this walkthrough page.)

Implement the [LookupBindingPropertiesAttribute](#) on controls, like a [ComboBox](#), that display lists (or tables) of data but also need to present a single column or property. For more information, see [Create a Windows Forms user control that supports lookup data binding](#).

This walkthrough creates a complex control that displays rows of data from a table. This example uses the `Customers` table from the Northwind sample database. The complex user control will display the customers table in a [DataGridView](#) in the custom control.

During this walkthrough, you'll learn how to:

- Add a new **User Control** to your project.
- Visually design the user control.
- Implement the `ComplexBindingProperty` attribute.
- Create a dataset with the [Data Source Configuration Wizard](#).
- Set the **Customers** table in the [Data Sources window](#) to use the new complex control.
- Add the new control by dragging it from the **Data Sources** window onto **Form1**.

Prerequisites

This walkthrough uses SQL Server Express LocalDB and the Northwind sample database.

1. If you don't have SQL Server Express LocalDB, install it either from the [SQL Server Express download page](#), or through the **Visual Studio Installer**. In the **Visual Studio Installer**, you can install SQL Server Express LocalDB as part of the **Data storage and processing** workload, or as an individual

component.

2. Install the Northwind sample database by following these steps:

- a. In Visual Studio, open the **SQL Server Object Explorer** window. (SQL Server Object Explorer is installed as part of the **Data storage and processing** workload in the Visual Studio Installer.) Expand the **SQL Server** node. Right-click on your LocalDB instance and select **New Query**.

A query editor window opens.

- b. Copy the [Northwind Transact-SQL script](#) to your clipboard. This T-SQL script creates the Northwind database from scratch and populates it with data.
- c. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes running and the Northwind database is created.

Create a Windows Forms app project

The first step is to create a **Windows Forms App** project for either C# or Visual Basic. Name the project **ComplexControlWalkthrough**.

Add a user control to the project

Because this walkthrough creates a complex data-bindable control from a **User Control**, add a **User Control** item to the project:

1. From the **Project** menu, choose **Add User Control**.
2. Type **ComplexDataGridView** in the **Name** area, and then click **Add**.

The **ComplexDataGridView** control is added to **Solution Explorer**, and opens in the designer.

Design the ComplexDataGridView control

To add a [DataGridView](#) to the user control, drag a [DataGridView](#) from the **Toolbox** onto the user control's design surface.

Add the required data-binding attribute

For complex controls that support data binding, you can implement the [ComplexBindingPropertiesAttribute](#):

1. Switch the **ComplexDataGridView** control to code view. (On the **View** menu, select **Code**.)
2. Replace the code in the `ComplexDataGridView` with the following:

```

using System.Windows.Forms;

namespace CS
{
    [System.ComponentModel.ComplexBindingProperties("DataSource", "DataMember")]
    public partial class ComplexDataGridView : UserControl
    {
        public object DataSource
        {
            get{ return dataGridView1.DataSource; }
            set{ dataGridView1.DataSource = value; }
        }

        public string DataMember
        {
            get{ return dataGridView1.DataMember; }
            set{ dataGridView1.DataMember = value; }
        }

        public ComplexDataGridView()
        {
            InitializeComponent();
        }
    }
}

```

```

<System.ComponentModel.ComplexBindingProperties("DataSource", "DataMember")>
Public Class ComplexDataGridView

    Public Property DataSource() As Object
        Get
            Return DataGridView1.DataSource
        End Get
        Set(ByVal value As Object)
            DataGridView1.DataSource = value
        End Set
    End Property

    Public Property DataMember() As String
        Get
            Return DataGridView1.DataMember
        End Get
        Set(ByVal value As String)
            DataGridView1.DataMember = value
        End Set
    End Property
End Class

```

3. From the **Build** menu, choose **Build Solution**.

Create a data source from your database

Use the **Data Source Configuration** wizard to create a data source based on the **Customers** table in the Northwind sample database:

1. To open the **Data Sources** window, on the **Data** menu, click **Show Data Sources**.
2. In the **Data Sources** window, select **Add New Data Source** to start the **Data Source Configuration** wizard.
3. Select **Database** on the **Choose a Data Source Type** page, and then click **Next**.
4. On the **Choose your Data Connection** page do one of the following:

- If a data connection to the Northwind sample database is available in the drop-down list, select it.
 - Select **New Connection** to launch the **Add/Modify Connection** dialog box.
5. If your database requires a password, select the option to include sensitive data, and then click **Next**.
 6. On the **Save connection string to the Application Configuration file** page, click **Next**.
 7. On the **Choose your Database Objects** page, expand the **Tables** node.
 8. Select the **Customers** table, and then click **Finish**.

The **NorthwindDataSet** is added to your project, and the **Customers** table appears in the **Data Sources** window.

Set the Customers table to use the ComplexDataGridView control

Within the **Data Sources** window, you can set the control to be created prior to dragging items onto your form:

1. Open **Form1** in the designer.
2. Expand the **Customers** node in the **Data Sources** window.
3. Click the drop-down arrow on the **Customers** node, and choose **Customize**.
4. Select the **ComplexDataGridView** from the list of **Associated Controls** in the **Data UI Customization Options** dialog box.
5. Click the drop-down arrow on the **Customers** table, and choose **ComplexDataGridView** from the control list.

Add controls to the form

You can create the data-bound controls by dragging items from the **Data Sources** window onto your form. Drag the main **Customers** node from the **Data Sources** window onto the form. Verify that the **ComplexDataGridView** control is used to display the table's data.

Run the application

Press F5 to run the application.

Next steps

Depending on your application requirements, there are several steps you may want to perform after creating a control that supports databinding. Some typical next steps include:

- Placing your custom controls in a control library so you can reuse them in other applications.
- Creating controls that support lookup scenarios. For more information, see [Create a Windows Forms user control that supports lookup data binding](#).

See also

- [Bind Windows Forms controls to data in Visual Studio](#)
- [Set the control to be created when dragging from the Data Sources window](#)
- [Windows Forms Controls](#)

Create a Windows Forms user control that supports lookup data binding

8/5/2021 • 6 minutes to read • [Edit Online](#)

When displaying data on Windows Forms, you can choose existing controls from the **Toolbox**, or you can author custom controls if your application requires functionality not available in the standard controls. This walkthrough shows how to create a control that implements the [LookupBindingPropertiesAttribute](#). Controls that implement the [LookupBindingPropertiesAttribute](#) can contain three properties that can be bound to data. Such controls are similar to a [ComboBox](#).

For more information on control authoring, see [Developing Windows Forms controls at design time](#).

When authoring controls for use in data-binding scenarios, you need to implement one of the following data-binding attributes:

DATA-BINDING ATTRIBUTE USAGE

Implement the [DefaultBindingPropertyAttribute](#) on simple controls, like a [TextBox](#), that display a single column (or property) of data. For more information, see [Create a Windows Forms user control that supports simple data binding](#).

Implement the [ComplexBindingPropertiesAttribute](#) on controls, like a [DataGridView](#), that display lists (or tables) of data. For more information, see [Create a Windows Forms user control that supports complex data binding](#).

Implement the [LookupBindingPropertiesAttribute](#) on controls, like a [ComboBox](#), that display lists (or tables) of data, but also need to present a single column or property. (This process is described in this walkthrough page.)

This walkthrough creates a lookup control that binds to data from two tables. This example uses the `Customers` and `Orders` tables from the Northwind sample database. The lookup control is bound to the `CustomerID` field from the `Orders` table. It uses this value to look up the `CompanyName` from the `Customers` table.

During this walkthrough, you'll learn how to:

- Create a new **Windows Forms Application**.
- Add a new **User Control** to your project.
- Visually design the user control.
- Implement the `LookupBindingProperty` attribute.
- Create a dataset with the **Data Source Configuration** wizard.
- Set the `CustomerID` column on the `Orders` table, in the **Data Sources** window, to use the new control.
- Create a form to display data in the new control.

Prerequisites

This walkthrough uses SQL Server Express LocalDB and the Northwind sample database.

1. If you don't have SQL Server Express LocalDB, install it either from the [SQL Server Express download page](#), or through the **Visual Studio Installer**. In the **Visual Studio Installer**, you can install SQL Server Express LocalDB as part of the **Data storage and processing** workload, or as an individual

component.

2. Install the Northwind sample database by following these steps:

- a. In Visual Studio, open the **SQL Server Object Explorer** window. (SQL Server Object Explorer is installed as part of the **Data storage and processing** workload in the Visual Studio Installer.) Expand the **SQL Server** node. Right-click on your LocalDB instance and select **New Query**.

A query editor window opens.

- b. Copy the [Northwind Transact-SQL script](#) to your clipboard. This T-SQL script creates the Northwind database from scratch and populates it with data.
- c. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes running and the Northwind database is created.

Create a Windows Forms app project

The first step is to create a **Windows Forms Application** project.

1. In Visual Studio, on the **File** menu, select **New > Project**.
2. Expand either **Visual C#** or **Visual Basic** in the left-hand pane, then select **Windows Desktop**.
3. In the middle pane, select the **Windows Forms App** project type.
4. Name the project **LookupControlWalkthrough**, and then choose **OK**.

The **LookupControlWalkthrough** project is created, and added to **Solution Explorer**.

Add a user control to the project

This walkthrough creates a lookup control from a **User Control**, so add a **User Control** item to the **LookupControlWalkthrough** project.

1. From the **Project** menu, select **Add User Control**.
2. Type `LookupBox` in the **Name** area, and then click **Add**.

The **LookupBox** control is added to **Solution Explorer**, and opens in the designer.

Design the LookupBox control

To design the **LookupBox** control, drag a **ComboBox** from the **Toolbox** onto the user control's design surface.

Add the required data-binding attribute

For lookup controls that support data binding, you can implement the [LookupBindingPropertiesAttribute](#).

1. Switch the **LookupBox** control to code view. (On the **View** menu, choose **Code**.)
2. Replace the code in the `LookupBox` with the following:

```
<System.ComponentModel.LookupBindingProperties("DataSource", "DisplayMember", "ValueMember",  
"LookupMember")>
```

```
Public Class LookupBox
```

```
    Public Property DataSource() As Object
```

```
        Get
```

```
            Return ComboBox1.DataSource
```

```
        End Get
```

```
        Set(ByVal value As Object)
```

```
            ComboBox1.DataSource = value
```

```
        End Set
```

```
    End Property
```

```
    Public Property DisplayMember() As String
```

```
        Get
```

```
            Return ComboBox1.DisplayMember
```

```
        End Get
```

```
        Set(ByVal value As String)
```

```
            ComboBox1.DisplayMember = value
```

```
        End Set
```

```
    End Property
```

```
    Public Property ValueMember() As String
```

```
        Get
```

```
            Return ComboBox1.ValueMember
```

```
        End Get
```

```
        Set(ByVal value As String)
```

```
            ComboBox1.ValueMember = value
```

```
        End Set
```

```
    End Property
```

```
    Public Property LookupMember() As String
```

```
        Get
```

```
            Return ComboBox1.SelectedValue.ToString()
```

```
        End Get
```

```
        Set(ByVal value As String)
```

```
            ComboBox1.SelectedValue = value
```

```
        End Set
```

```
    End Property
```

```
End Class
```

```

using System.Windows.Forms;

namespace CS
{
    [System.ComponentModel.LookupBindingProperties("DataSource", "DisplayMember", "ValueMember",
    "LookupMember")]
    public partial class LookupBox : UserControl
    {
        public object DataSource
        {
            get{ return comboBox1.DataSource; }
            set{ comboBox1.DataSource = value; }
        }

        public string DisplayMember
        {
            get{ return comboBox1.DisplayMember; }
            set{ comboBox1.DisplayMember = value; }
        }

        public string ValueMember
        {
            get{ return comboBox1.ValueMember; }
            set{ comboBox1.ValueMember = value; }
        }

        public string LookupMember
        {
            get{ return comboBox1.SelectedValue.ToString(); }
            set{ comboBox1.SelectedValue = value; }
        }

        public LookupBox()
        {
            InitializeComponent();
        }
    }
}

```

3. From the **Build** menu, choose **Build Solution**.

Create a data source from your database

This step creates a data source using the **Data Source Configuration** wizard, based on the **Customers** and **Orders** tables in the Northwind sample database.

1. To open the **Data Sources** window, on the **Data** menu, click **Show Data Sources**.
2. In the **Data Sources** window, select **Add New Data Source** to start the **Data Source Configuration** wizard.
3. Select **Database** on the **Choose a Data Source Type** page, and then click **Next**.
4. On the **Choose your Data Connection** page do one of the following:
 - If a data connection to the Northwind sample database is available in the drop-down list, select it.
 - Select **New Connection** to launch the **Add/Modify Connection** dialog box.
5. If your database requires a password, select the option to include sensitive data, and then click **Next**.
6. On the **Save connection string to the Application Configuration file** page, click **Next**.

7. On the **Choose your Database Objects** page, expand the **Tables** node.

8. Select the **Customers** and **Orders** tables, and then click **Finish**.

The **NorthwindDataSet** is added to your project, and the **Customers** and **orders** tables appear in the **Data Sources** window.

Set the CustomerID column of the Orders table to use the LookupBox control

Within the **Data Sources** window, you can set the control to be created prior to dragging items onto your form.

1. Open **Form1** in the designer.
2. Expand the **Customers** node in the **Data Sources** window.
3. Expand the **Orders** node (the one in the **Customers** node below the **Fax** column).
4. Click the drop-down arrow on the **Orders** node, and choose **Details** from the control list.
5. Click the drop-down arrow on the **CustomerID** column (in the **Orders** node), and choose **Customize**.
6. Select the **LookupBox** from the list of **Associated Controls** in the **Data UI Customization Options** dialog box.
7. Click **OK**.
8. Click the drop-down arrow on the **CustomerID** column, and choose **LookupBox**.

Add controls to the form

You can create the data-bound controls by dragging items from the **Data Sources** window onto **Form1**.

To create data-bound controls on the Windows Form, drag the **Orders** node from the **Data Sources** window onto the Windows Form, and verify that the **LookupBox** control is used to display the data in the **CustomerID** column.

Bind the control to look up CompanyName from the Customers table

To set up the lookup bindings, select the main **Customers** node in the **Data Sources** window, and drag it onto the combo box in the **CustomerIDLookupBox** on **Form1**.

This sets up the data binding to display the **CompanyName** from the **Customers** table, while maintaining the **CustomerID** value from the **Orders** table.

Run the application

- Press **F5** to run the application.
- Navigate through some records, and verify that the **CompanyName** appears in the **LookupBox** control.

See also

- [Bind Windows Forms controls to data in Visual Studio](#)

Pass data between forms

8/5/2021 • 5 minutes to read • [Edit Online](#)

This walkthrough provides step-by-step instructions for passing data from one form to another. Using the customers and orders tables from Northwind, one form allows users to select a customer, and a second form displays the selected customer's orders. This walkthrough shows how to create a method on the second form that receives data from the first form.

NOTE

This walkthrough demonstrates only one way to pass data between forms. There are other options for passing data to a form, including creating a second constructor to receive data, or creating a public property that can be set with data from the first form.

Tasks illustrated in this walkthrough include:

- Creating a new **Windows Forms Application** project.
- Creating and configuring a dataset with the [Data Source Configuration Wizard](#).
- Selecting the control to be created on the form when dragging items from the **Data Sources** window.
For more information, see [Set the control to be created when dragging from the Data Sources window](#).
- Creating a data-bound control by dragging items from the **Data Sources** window onto a form.
- Creating a second form with a grid to display data.
- Creating a TableAdapter query to fetch orders for a specific customer.
- Passing data between forms.

Prerequisites

This walkthrough uses SQL Server Express LocalDB and the Northwind sample database.

1. If you don't have SQL Server Express LocalDB, install it either from the [SQL Server Express download page](#), or through the **Visual Studio Installer**. In the Visual Studio Installer, SQL Server Express LocalDB can be installed as part of the **Data storage and processing** workload, or as an individual component.
2. Install the Northwind sample database by following these steps:
 - a. In Visual Studio, open the **SQL Server Object Explorer** window. (SQL Server Object Explorer is installed as part of the **Data storage and processing** workload in the Visual Studio Installer.) Expand the **SQL Server** node. Right-click on your LocalDB instance and select **New Query**.

A query editor window opens.
 - b. Copy the [Northwind Transact-SQL script](#) to your clipboard. This T-SQL script creates the Northwind database from scratch and populates it with data.
 - c. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes running and the Northwind database is created.

Create the Windows Forms app project

1. In Visual Studio, on the **File** menu, select **New > Project**.
2. Expand either **Visual C#** or **Visual Basic** in the left-hand pane, then select **Windows Desktop**.
3. In the middle pane, select the **Windows Forms App** project type.
4. Name the project **PassingDataBetweenForms**, and then choose **OK**.

The **PassingDataBetweenForms** project is created, and added to **Solution Explorer**.

Create the data source

1. To open the **Data Sources** window, on the **Data** menu, click **Show Data Sources**.
2. In the **Data Sources** window, select **Add New Data Source** to start the **Data Source Configuration** wizard.
3. Select **Database** on the **Choose a Data Source Type** page, and then click **Next**.
4. On the **Choose a database model** page, verify that **Dataset** is specified, and then click **Next**.
5. On the **Choose your Data Connection** page, do one of the following:
 - If a data connection to the Northwind sample database is available in the drop-down list, select it.
 - Select **New Connection** to launch the **Add/Modify Connection** dialog box.
6. If your database requires a password and if the option to include sensitive data is enabled, select the option and then click **Next**.
7. On the **Save connection string to the Application Configuration file** page, click **Next**.
8. On the **Choose your Database Objects** page, expand the **Tables** node.
9. Select the **Customers** and **Orders** tables, and then click **Finish**.

The **NorthwindDataSet** is added to your project, and the **Customers** and **Orders** tables appear in the **Data Sources** window.

Create the first form (Form1)

You can create a data-bound grid (a [DataGridView](#) control), by dragging the **Customers** node from the **Data Sources** window onto the form.

To create a data-bound grid on the form

- Drag the main **Customers** node from the **Data Sources** window onto **Form1**.

A [DataGridView](#) and a tool strip ([BindingNavigator](#)) for navigating records appear on **Form1**. A [NorthwindDataSet](#), [CustomersTableAdapter](#), [BindingSource](#), and [BindingNavigator](#) appear in the component tray.

Create the second form

Create a second form to pass data to.

1. From the **Project** menu, choose **Add Windows Form**.
2. Leave the default name of **Form2**, and click **Add**.

3. Drag the main **Orders** node from the **Data Sources** window onto **Form2**.

A [DataGridView](#) and a tool strip ([BindingNavigator](#)) for navigating records appear on **Form2**. A [NorthwindDataSet](#), [CustomersTableAdapter](#), [BindingSource](#), and [BindingNavigator](#) appear in the component tray.

4. Delete the **OrdersBindingNavigator** from the component tray.

The **OrdersBindingNavigator** disappears from **Form2**.

Add a TableAdapter query

Add a TableAdapter query to **Form2** to load orders for the selected customer on **Form1**.

1. Double-click the **NorthwindDataSet.xsd** file in **Solution Explorer**.
2. Right-click the **OrdersTableAdapter**, and select **Add Query**.
3. Leave the default option of **Use SQL statements**, and then click **Next**.
4. Leave the default option of **SELECT which returns rows**, and then click **Next**.
5. Add a WHERE clause to the query, to return **Orders** based on the **CustomerID**. The query should be similar to the following:

```
SELECT OrderID, CustomerID, EmployeeID, OrderDate, RequiredDate, ShippedDate, ShipVia, Freight,
ShipName, ShipAddress, ShipCity, ShipRegion, ShipPostalCode, ShipCountry
FROM Orders
WHERE CustomerID = @CustomerID
```

NOTE

Verify the correct parameter syntax for your database. For example, in Microsoft Access, the WHERE clause would look like: `WHERE CustomerID = ?`.

6. Click **Next**.
7. For the **Fill a DataTableMethod Name**, type **FillByCustomerID**.
8. Clear the **Return a DataTable** option, and then click **Next**.
9. Click **Finish**.

Create a method on Form2 to pass data to

1. Right-click **Form2**, and select **View Code** to open **Form2** in the **Code Editor**.
2. Add the following code to **Form2** after the **Form2_Load** method:

```
Friend Sub LoadOrders(ByVal CustomerID As String)
    OrdersTableAdapter.FillByCustomerID(NorthwindDataSet.Orders, CustomerID)
End Sub
```

```
internal void LoadOrders(String CustomerID)
{
    ordersTableAdapter.FillByCustomerID(northwindDataSet.Orders, CustomerID);
}
```

Create a method on Form1 to pass data and display Form2

1. In **Form1**, right-click the Customer data grid, and then click **Properties**.
2. In the **Properties** window, click **Events**.
3. Double-click the **CellDoubleClick** event.

The code editor appears.

4. Update the method definition to match the following sample:

```
private void customersDataGridView_DoubleClick(object sender, EventArgs e)
{
    System.Data.DataRowView SelectedRowView;
    NorthwindDataSet.CustomersRow SelectedRow;

    SelectedRowView = (System.Data.DataRowView)customersBindingSource.Current;
    SelectedRow = (NorthwindDataSet.CustomersRow)SelectedRowView.Row;

    Form2 OrdersForm = new Form2();
    OrdersForm.LoadOrders(SelectedRow.CustomerID);
    OrdersForm.Show();
}
```

```
Private Sub CustomersDataGridView_DoubleClick() Handles CustomersDataGridView.DoubleClick

    Dim SelectedRowView As Data.DataRowView
    Dim SelectedRow As NorthwindDataSet.CustomersRow

    SelectedRowView = CType(CustomersBindingSource.Current, System.Data.DataRowView)
    SelectedRow = CType(SelectedRowView.Row, NorthwindDataSet.CustomersRow)

    Dim OrdersForm As New Form2
    OrdersForm.LoadOrders(SelectedRow.CustomerID)
    OrdersForm.Show()
End Sub
```

Run the app

- Press **F5** to run the application.
- Double-click a customer record in **Form1** to open **Form2** with that customer's orders.

Next steps

Depending on your application requirements, there are several steps you may want to perform after passing data between forms. Some enhancements you could make to this walkthrough include:

- Editing the dataset to add or remove database objects. For more information, see [Create and configure datasets](#).
- Adding functionality to save data back to the database. For more information, see [Save data back to the database](#).

See also

- [Bind Windows Forms controls to data in Visual Studio](#)

Bind objects as data sources in Visual Studio

8/5/2021 • 7 minutes to read • [Edit Online](#)

Visual Studio provides design-time tools for working with custom objects as the data source in your application. When you want to store data from a database in an object that you bind to UI controls, the recommended approach is to use Entity Framework to generate the class or classes. Entity Framework auto-generates all the boilerplate change-tracking code, which means that any changes to the local objects are automatically persisted to the database when you call `AcceptChanges` on the `DbSet` object. For more information, see [Entity Framework Documentation](#).

TIP

The approaches to object binding in this article should only be considered if your application is already based on datasets. You can also use these approaches if you are already familiar with datasets, and the data you will be processing is tabular and not too complex or too big. For an even simpler example, involving loading data directly into objects by using a `DataReader` and manually updating the UI without databinding, see [Create a simple data application by using ADO.NET](#).

Object requirements

The only requirement for custom objects to work with the data design tools in Visual Studio is that the object needs at least one public property.

Generally, custom objects do not require any specific interfaces, constructors, or attributes to act as a data source for an application. However, if you want to drag the object from the **Data Sources** window to a design surface to create a data-bound control, and if the object implements the `ITypeedList` or `IListSource` interface, the object must have a default constructor. Otherwise, Visual Studio cannot instantiate the data source object, and it displays an error when you drag the item to the design surface.

Examples of using custom objects as data sources

While there are countless ways to implement your application logic when working with objects as a data source, for SQL databases there are a few standard operations that can be simplified by using the Visual Studio-generated `TableAdapter` objects. This page explains how to implement these standard processes using `TableAdapters`. It is not intended as a guide for creating your custom objects. For example, you will typically perform the following standard operations regardless of the specific implementation of your objects, or application's logic:

- Loading data into objects (typically from a database).
- Creating a typed collection of objects.
- Adding objects to and removing objects from a collection.
- Displaying the object data to users on a form.
- Changing/editing the data in an object.
- Saving data from objects back to the database.

Load data into objects

For this example, you load data into your objects by using `TableAdapters`. By default, `TableAdapters` are created with two kinds of methods that fetch data from a database and populate data tables.

- The `TableAdapter.Fill` method fills an existing data table with the data returned.
- The `TableAdapter.GetData` method returns a new data table populated with data.

The easiest way to load your custom objects with data is to call the `TableAdapter.GetData` method, loop through the collection of rows in the returned data table, and populate each object with the values in each row. You can create a `GetData` method that returns a populated data table for any query added to a `TableAdapter`.

NOTE

Visual Studio names the `TableAdapter` queries `Fill` and `GetData` by default, but you can change those names to any valid method name.

The following example shows how to loop through the rows in a data table, and populate an object with data:

```

private void LoadCustomers()
{
    NorthwindDataSet.CustomersDataTable customerData =
        customersTableAdapter1.GetTop5Customers();

    foreach (NorthwindDataSet.CustomersRow customerRow in customerData)
    {
        Customer currentCustomer = new Customer();
        currentCustomer.CustomerID = customerRow.CustomerID;
        currentCustomer.CompanyName = customerRow.CompanyName;

        if (customerRow.IsAddressNull() == false)
        {
            currentCustomer.Address = customerRow.Address;
        }

        if (customerRow.IsCityNull() == false)
        {
            currentCustomer.City = customerRow.City;
        }

        if (customerRow.IsContactNameNull() == false)
        {
            currentCustomer.ContactName = customerRow.ContactName;
        }

        if (customerRow.IsContactTitleNull() == false)
        {
            currentCustomer.ContactTitle = customerRow.ContactTitle;
        }

        if (customerRow.IsCountryNull() == false)
        {
            currentCustomer.Country = customerRow.Country;
        }

        if (customerRow.IsFaxNull() == false)
        {
            currentCustomer.Fax = customerRow.Fax;
        }

        if (customerRow.IsPhoneNull() == false)
        {
            currentCustomer.Phone = customerRow.Phone;
        }

        if (customerRow.IsPostalCodeNull() == false)
        {
            currentCustomer.PostalCode = customerRow.PostalCode;
        }

        if (customerRow.IsRegionNull() == false)
        {
            currentCustomer.Region = customerRow.Region;
        }

        LoadOrders(currentCustomer);
        customerBindingSource.Add(currentCustomer);
    }
}

```

```

Private Sub LoadCustomers()
    Dim customerData As NorthwindDataSet.CustomersDataTable =
        CustomersTableAdapter1.GetTop5Customers()

    Dim customerRow As NorthwindDataSet.CustomersRow

    For Each customerRow In customerData
        Dim currentCustomer As New Customer()
        With currentCustomer

            .CustomerID = customerRow.CustomerID
            .CompanyName = customerRow.CompanyName

            If Not customerRow.IsAddressNull Then
                .Address = customerRow.Address
            End If

            If Not customerRow.IsCityNull Then
                .City = customerRow.City
            End If

            If Not customerRow.IsContactNameNull Then
                .ContactName = customerRow.ContactName
            End If

            If Not customerRow.IsContactTitleNull Then
                .ContactTitle = customerRow.ContactTitle
            End If

            If Not customerRow.IsCountryNull Then
                .Country = customerRow.Country
            End If

            If Not customerRow.IsFaxNull Then
                .Fax = customerRow.Fax
            End If

            If Not customerRow.IsPhoneNull Then
                .Phone = customerRow.Phone
            End If

            If Not customerRow.IsPostalCodeNull Then
                .PostalCode = customerRow.PostalCode
            End If

            If Not customerRow.Is_RegionNull Then
                .Region = customerRow._Region
            End If

        End With

        LoadOrders(currentCustomer)
        CustomerBindingSource.Add(currentCustomer)
    Next
End Sub

```

Create a typed collection of objects

You can create collection classes for your objects, or use the typed collections that are automatically provided by the [BindingSource](#) component.

When you're creating a custom collection class for objects, we suggest that you inherit from [BindingList<T>](#). This generic class provides functionality to administer your collection, as well as the ability to raise events that send notifications to the data-binding infrastructure in Windows Forms.

The automatically generated collection in the [BindingSource](#) uses a [BindingList<T>](#) for its typed collection. If

your application does not require additional functionality, you can maintain your collection within the [BindingSource](#). For more information, see the [List](#) property of the [BindingSource](#) class.

NOTE

If your collection requires functionality not provided by the base implementation of the [BindingList<T>](#), you should create a custom collection so you can add to the class as needed.

The following code shows how to create the class for a strongly-typed collection of `Order` objects:

```
/// <summary>
/// A collection of Orders
/// </summary>
public class Orders: System.ComponentModel.BindingList<Order>
{
    // Add any additional functionality required by your collection.
}
```

```
''' <summary>
''' A collection of Orders
''' </summary>
Public Class Orders
    Inherits System.ComponentModel.BindingList(Of Order)

    ' Add any additional functionality required by your collection.

End Class
```

Add objects to a collection

You add objects to a collection by calling the `Add` method of your custom collection class or of the [BindingSource](#).

NOTE

The `Add` method is automatically provided for your custom collection when you inherit from [BindingList<T>](#).

The following code shows how to add objects to the typed collection in a [BindingSource](#):

```
Customer currentCustomer = new Customer();
customerBindingSource.Add(currentCustomer);
```

```
Dim currentCustomer As New Customer()
CustomerBindingSource.Add(currentCustomer)
```

The following code shows how to add objects to a typed collection that inherits from [BindingList<T>](#):

NOTE

In this example, the `Orders` collection is a property of the `Customer` object.

```
Order currentOrder = new Order();
currentCustomer.Orders.Add(currentOrder);
```

```
Dim currentOrder As New Order()
currentCustomer.Orders.Add(currentOrder)
```

Remove objects from a collection

You remove objects from a collection by calling the `Remove` or `RemoveAt` method of your custom collection class or of [BindingSource](#).

NOTE

The `Remove` and `RemoveAt` methods are automatically provided for your custom collection when you inherit from [BindingList<T>](#).

The following code shows how to locate and remove objects from the typed collection in a [BindingSource](#) with the `RemoveAt` method:

```
int customerIndex = customerBindingSource.Find("CustomerID", "ALFKI");
customerBindingSource.RemoveAt(customerIndex);
```

```
Dim customerIndex As Integer = CustomerBindingSource.Find("CustomerID", "ALFKI")
CustomerBindingSource.RemoveAt(customerIndex)
```

Display object data to users

To display the data in objects to users, create an object data source using the **Data Source Configuration** wizard, and then drag the entire object or individual properties onto your form from the **Data Sources** window.

Modify the data in objects

To edit data in custom objects that are data-bound to Windows Forms controls, simply edit the data in the bound control (or directly in the object's properties). Data-binding architecture updates the data in the object.

If your application requires the tracking of changes and the rolling back of proposed changes to their original values, then you must implement this functionality in your object model. For examples of how data tables keep track of proposed changes, see [DataRowState](#), [HasChanges](#), and [GetChanges](#).

Save data in objects back to the database

Save data back to the database by passing the values from your object to the TableAdapter's DBDirect methods.

Visual Studio creates DBDirect methods that can be executed directly against the database. These methods do not require DataSet or DataTable objects.

| TABLEADAPTER DBDIRECT METHOD | DESCRIPTION |
|----------------------------------|--|
| <code>TableAdapter.Insert</code> | Adds new records to a database, allowing you to pass in individual column values as method parameters. |

| TABLEADAPTER DBDIRECT METHOD | DESCRIPTION |
|----------------------------------|--|
| <code>TableAdapter.Update</code> | <p>Updates existing records in a database. The Update method takes original and new column values as method parameters. The original values are used to locate the original record, and the new values are used to update that record.</p> <p>The <code>TableAdapter.Update</code> method is also used to reconcile changes in a dataset back to the database, by taking a DataSet, DataTable, DataRow, or array of DataRows as method parameters.</p> |
| <code>TableAdapter.Delete</code> | Deletes existing records from the database based on the original column values passed in as method parameters. |

To save data from a collection of objects, loop through the collection of objects (for example, using a for-next loop). Send the values for each object to the database by using the TableAdapter's DBDirect methods.

The following example shows how to use the `TableAdapter.Insert` DBDirect method to add a new customer directly into the database:

```
private void AddNewCustomers(Customer currentCustomer)
{
    customersTableAdapter.Insert(
        currentCustomer.CustomerID,
        currentCustomer.CompanyName,
        currentCustomer.ContactName,
        currentCustomer.ContactTitle,
        currentCustomer.Address,
        currentCustomer.City,
        currentCustomer.Region,
        currentCustomer.PostalCode,
        currentCustomer.Country,
        currentCustomer.Phone,
        currentCustomer.Fax);
}
```

```
Private Sub AddNewCustomer(ByVal currentCustomer As Customer)

    CustomersTableAdapter.Insert(
        currentCustomer.CustomerID,
        currentCustomer.CompanyName,
        currentCustomer.ContactName,
        currentCustomer.ContactTitle,
        currentCustomer.Address,
        currentCustomer.City,
        currentCustomer.Region,
        currentCustomer.PostalCode,
        currentCustomer.Country,
        currentCustomer.Phone,
        currentCustomer.Fax)

End Sub
```

See also

- [Bind controls to data in Visual Studio](#)

Customize how Visual Studio creates captions for data-bound controls

8/5/2021 • 3 minutes to read • [Edit Online](#)

When you drag items from the [Data Sources window](#) onto a designer, a special consideration comes into play: the column names in the caption labels are reformatted into a more readable string when two or more words are found to be concatenated together.

You can customize the way in which these labels are created by setting the **SmartCaptionExpression**, **SmartCaptionReplacement**, and **SmartCaptionSuffix** values in the `HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\15.0\Data Designers` registry key.

You can customize the way in which these labels are created by setting the **SmartCaptionExpression**, **SmartCaptionReplacement**, and **SmartCaptionSuffix** values in the `HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\16.0\Data Designers` registry key.

NOTE

This registry key does not exist until you create it.

Smart captioning is controlled by the regular expression entered into the value of the **SmartCaptionExpression** value. Adding the **Data Designers** registry key overrides the default regular expression that controls caption labels. For more information about regular expressions, see [Using regular expressions in Visual Studio](#).

The following table describes the registry values that control caption labels.

| REGISTRY ITEM | DESCRIPTION |
|--------------------------------|---|
| SmartCaptionExpression | The regular expression you use to match your patterns. |
| SmartCaptionReplacement | The format to display any groups matched in the SmartCaptionExpression . |
| SmartCaptionSuffix | An optional string to append to the end of the caption. |

The following table lists the internal default settings for these registry values.

| REGISTRY ITEM | DEFAULT VALUE | EXPLANATION |
|--------------------------------|-----------------------------------|--|
| SmartCaptionExpression | <code>(\p{Ll})(\p{Lu})[_]+</code> | Matches a lowercase character followed by an uppercase character or an underscore. |
| SmartCaptionReplacement | <code>\$1 \$2</code> | The \$1 represents any characters matched in the first parentheses of the expression, and the \$2 represents any characters matched in the second parentheses. The replacement is the first match, a space, and then the second match. |

| REGISTRY ITEM | DEFAULT VALUE | EXPLANATION |
|---------------------------|---------------|---|
| SmartCaptionSuffix | : | Represents a character appended to the returned string. For example, if the caption is <code>Company Name</code> , the suffix makes it <code>Company Name:</code> |

Caution

Be very careful when doing anything in the Registry Editor. Back up the registry before editing it. If you use the Registry Editor incorrectly, you can cause serious problems that may require you to reinstall your operating system. Microsoft does not guarantee that problems that you cause by using the Registry Editor incorrectly can be resolved. Use the Registry Editor at your own risk.

For information about backing up, editing, and restoring the registry, see [Windows registry information for advanced users](#).

Modify the smart captioning behavior of the Data Sources window

1. Open a command window by clicking **Start** and then **Run**.
2. Type `regedit` in the **Run** dialog box, and click **OK**.
3. Expand the **HKEY_CURRENT_USER > Software > Microsoft > VisualStudio** node.
4. Right-click the **15.0** node, and create a new **Key** named `Data Designers`.
4. Right-click the **16.0** node, and create a new **Key** named `Data Designers`.
5. Right-click the **Data Designers** node, and create three new string values:
 - `SmartCaptionExpression`
 - `SmartCaptionReplacement`
 - `SmartCaptionSuffix`
6. Right-click the **SmartCaptionExpression** value, and select **Modify**.
7. Enter the regular expression you want the **Data Sources** window to use.
8. Right-click the **SmartCaptionReplacement** value, and select **Modify**.
9. Enter the replacement string formatted the way you want to display the patterns matched in your regular expression.
10. Right-click the **SmartCaptionSuffix** value, and select **Modify**.
11. Enter any characters you want to appear at the end of the caption.

The next time you drag items from the **Data Sources** window, the caption labels are created using the new registry values provided.

Turn off the smart captioning feature

1. Open a command window by clicking **Start** and then **Run**.
2. Type `regedit` in the **Run** dialog box, and click **OK**.
3. Expand the **HKEY_CURRENT_USER > Software > Microsoft > VisualStudio** node.
4. Right-click the **15.0** node, and create a new **Key** named `Data Designers`.

4. Right-click the **16.0** node, and create a new **Key** named `Data Designers`.
5. Right-click the **Data Designers** node, and create three new string values:
 - `SmartCaptionExpression`
 - `SmartCaptionReplacement`
 - `SmartCaptionSuffix`
6. Right-click the **SmartCaptionExpression** item, and select **Modify**.
7. Enter `(.*)` for the value. This will match the entire string.
8. Right-click the **SmartCaptionReplacement** item, and select **Modify**.
9. Enter `$1` for the value. This replaces the string with the matched value, which is the entire string so that it will remain unchanged.

The next time you drag items from the **Data Sources** window, the caption labels are created with unmodified captions.

See also

- [Bind controls to data in Visual Studio](#)

Windows Communication Foundation Services and WCF Data Services in Visual Studio

8/5/2021 • 12 minutes to read • [Edit Online](#)

Visual Studio provides tools for working with Windows Communication Foundation (WCF) and WCF Data Services, Microsoft technologies for creating distributed applications. This topic provides an introduction to services from a Visual Studio perspective. For the full documentation, see [WCF Data Services 4.5](#).

What Is WCF?

Windows Communication Foundation (WCF) is a unified framework for creating secure, reliable, transacted, and interoperable distributed applications. It replaces older interprocess communication technologies such as ASMX web services, .NET Remoting, Enterprise Services (DCOM), and MSMQ. WCF brings together the functionality of all those technologies under a unified programming model. This simplifies the experience of developing distributed applications.

What are WCF Data Services

WCF Data Services is an implementation of the Open Data (OData) Protocol standard. WCF Data Services lets you expose tabular data as a set of REST APIs, allowing you to return data using standard HTTP verbs such as GET, POST, PUT, or DELETE. On the server side, WCF Data Services are being superseded by [ASP.NET Web API](#) for creating new OData services. The WCF Data Services client library continues to be a good choice for consuming OData services in a .NET application from Visual Studio (**Project > Add Service Reference**). For more information, see [WCF Data Services 4.5](#).

WCF programming model

The WCF programming model is based on communication between two entities: a WCF service and a WCF client. The programming model is encapsulated in the [System.ServiceModel](#) namespace in .NET.

WCF Service

A WCF service is based on an interface that defines a contract between the service and the client. It is marked with a [ServiceContractAttribute](#) attribute, as shown in the following code:

```
[ServiceContract]
public interface IService1
```

```
<ServiceContract(>>
Public Interface IService1
```

You define functions or methods that are exposed by a WCF service by marking them with a [OperationContractAttribute](#) attribute.

```
[OperationContract]
string GetData(string value);
```

```
<OperationContract(>>
Function GetData(ByVal value As String) As String
```

In addition, you can expose serialized data by marking a composite type with a [DataContractAttribute](#) attribute. This enables data binding in a client.

After an interface and its methods are defined, they are encapsulated in a class that implements the interface. A single WCF service class can implement multiple service contracts.

A WCF service is exposed for consumption through what is known as an *endpoint*. The endpoint provides the only way to communicate with the service; you cannot access the service through a direct reference as you would with other classes.

An endpoint consists of an address, a binding, and a contract. The address defines where the service is located; this could be a URL, an FTP address, or a network or local path. A binding defines the way that you communicate with the service. WCF bindings provide a versatile model for specifying a protocol such as HTTP or FTP, a security mechanism such as Windows Authentication or user names and passwords, and much more. A contract includes the operations that are exposed by the WCF service class.

Multiple endpoints can be exposed for a single WCF service. This enables different clients to communicate with the same service in different ways. For example, a banking service might provide one endpoint for employees and another for external customers, each using a different address, binding, and/or contract.

WCF client

A WCF client consists of a *proxy* that enables an application to communicate with a WCF service, and an endpoint that matches an endpoint defined for the service. The proxy is generated on the client side in the *app.config* file and includes information about the types and methods that are exposed by the service. For services that expose multiple endpoints, the client can select the one that best fits its needs, for example, to communicate over HTTP and use Windows Authentication.

After a WCF client has been created, you reference the service in your code just as you would any other object. For example, to call the `GetData` method shown earlier, you would write code that resembles the following:

```
private void button1_Click(System.Object sender, System.EventArgs e)
{
    ServiceReference1.Service1Client client = new
        ServiceReference1.Service1Client();
    string returnString;

    returnString = client.GetData(textBox1.Text);
    label1.Text = returnString;
}
```

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    Dim client As New ServiceReference1.Service1Client
    Dim returnString As String

    returnString = client.GetData(TextBox1.Text)
    Label1.Text = returnString
End Sub
```

WCF tools in Visual Studio

Visual Studio provides tools to help you create both WCF services and WCF clients. For a walkthrough that demonstrates the tools, see [Walkthrough: Creating a simple WCF service in Windows Forms](#).

Create and test WCF services

You can use the WCF Visual Studio templates as a foundation to quickly create your own service. You can then use WCF Service Auto Host and WCF Test Client to debug and test the service. These tools together provide a

fast and convenient debug and testing cycle, and eliminate the requirement to commit to a hosting model at an early stage.

WCF Templates

WCF Visual Studio templates provide a basic class structure for service development. Several WCF templates are available in the **Add New Project** dialog box. These include WCF service library projects, WCF service websites, and WCF Service item templates.

When you select a template, files are added for a service contract, a service implementation, and a service configuration. All necessary attributes are already added, creating a simple "Hello World" type of service, and you did not have to write any code. You will, of course, want to add code to provide functions and methods for your real world service, but the templates provide the basic foundation.

To learn more about WCF templates, see [WCF Visual Studio templates](#).

WCF service host

When you start the Visual Studio debugger (by pressing **F5**) for a WCF service project, the WCF Service Host tool is automatically started to host the service locally. WCF Service Host enumerates the services in a WCF service project, loads the project's configuration, and instantiates a host for each service that it finds.

By using WCF Service Host, you can test a WCF service without writing extra code or committing to a specific host during development.

To learn more about WCF Service Host, see [WCF service host \(WcfSvcHost.exe\)](#).

WCF test client

The WCF Test Client tool enables you to input test parameters, submit that input to a WCF service, and view the response that the service sends back. It provides a convenient service testing experience when you combine it with WCF Service Host. Find the tool in the *%ProgramFiles(x86)%\Microsoft Visual Studio\2017\Enterprise\Common7\IDE* folder.

When you press **F5** to debug a WCF service project, WCF Test Client opens and displays a list of service endpoints that are defined in the configuration file. You can test the parameters and start the service, and repeat this process to continuously test and validate your service.

To learn more about WCF Test Client, see [WCF test client \(WcfTestClient.exe\)](#).

Accessing WCF services in Visual Studio

Visual Studio simplifies the task of creating WCF clients, automatically generating a proxy and an endpoint for services that you add by using the **Add Service Reference** dialog box. All necessary configuration information is added to the *app.config* file. Most of the time, all that you have to do is instantiate the service in order to use it.

The **Add Service Reference** dialog box enables you to enter the address for a service or to search for a service that is defined in your solution. The dialog box returns a list of services and the operations provided by those services. It also enables you to define the namespace by which you will reference the services in code.

The **Configure Service References** dialog box enables you to customize the configuration for a service. You can change the address for a service, specify access level, asynchronous behavior, and message contract types, and configure type reuse.

How to: Select a service endpoint

Some Windows Communication Foundation (WCF) services expose multiple endpoints through which a client may communicate with the service. For example, a service might expose one endpoint that uses an HTTP binding and user name and password security and a second endpoint that uses FTP and Windows Authentication. The first endpoint might be used by applications that access the service from outside a firewall, whereas the second might be used on an intranet.

In such a case, you can specify the `endpointConfigurationName` as a parameter to the constructor for a service reference.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To select a service endpoint

1. Add a reference to a WCF service by right-clicking the project node in **Solution Explorer** and choosing **Add service reference**.
2. In the Code Editor, add a constructor for the service reference:

```
Dim proxy As New ServiceReference.Service1Client(
```

```
ServiceReference.Service1Client proxy = new ServiceReference.Service1Client(
```

NOTE

Replace *ServiceReference* with the namespace for the service reference and replace *Service1Client* with the name of the service.

3. An IntelliSense list displays that includes the overloads for the constructor. Select the `endpointConfigurationName As String` overload.
4. Following the overload, type `=` *ConfigurationName*, where *ConfigurationName* is the name of the endpoint that you want to use.

NOTE

If you do not know the names of the available endpoints, you can find them in the *app.config* file.

To find the available endpoints for a WCF service

1. In **Solution Explorer**, right-click the **app.config** file for the project that contains the service reference and then click **Open**. The file appears in the Code Editor.
2. Search for the `<Client>` tag in the file.
3. Search underneath the `<Client>` tag for a tag that starts with `<Endpoint>`.

If the service reference provides multiple endpoints, there will be two or more `<Endpoint>` tags.

4. Inside the `<EndPoint>` tag, you will find a `name="SomeService"` parameter (where *SomeService* represents an endpoint name). This is the name for the endpoint that can be passed to the `endpointConfigurationName As String` overload of a constructor for a service reference.

How to: Call a service method asynchronously

Most methods in Windows Communication Foundation (WCF) services may be called either synchronously or asynchronously. Calling a method asynchronously enables your application to continue to work while the

method is being called when it operates over a slow connection.

By default, when a service reference is added to a project, it is configured to call methods synchronously. You can change the behavior to call methods asynchronously by changing a setting in the **Configure Service Reference** dialog box.

NOTE

This option is set on a per-service basis. If one method for a service is called asynchronously, all methods must be called asynchronously.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To call a service method asynchronously

1. In **Solution Explorer**, select the service reference.
2. On the **Project** menu, click **Configure Service Reference**.
3. In the **Configure Service Reference** dialog box, select the **Generate asynchronous operations** check box.

How to: Bind data returned by a service

You can bind data returned by a Windows Communication Foundation (WCF) service to a control just as you can bind any other data source to a control. When you add a reference to a WCF service, if the service contains composite types that return data, they are automatically added to the **Data Sources** window.

To bind a control to single data field returned by a WCF service

1. On the **Data** menu, click **Show Data Sources**.

The **Data Sources** window appears.
2. In the **Data Sources** window, expand the node for your service reference. Any composite types returned by the service display.
3. Expand a node for a type. The data fields for that type appear.
4. Select a field and click the drop-down arrow to display a list of controls that are available for the data type.
5. Click the type of control to which you want to bind.
6. Drag the field onto a form. The control is added to the form, together with a [BindingSource](#) component and a [BindingNavigator](#) component.
7. Repeat steps 4 through 6 for any other fields that you want to bind.

To bind a control to composite type returned by a WCF service

1. On the **Data** menu, select **Show Data Sources**. The **Data Sources** window appears.
2. In the **Data Sources** window, expand the node for your service reference. Any composite types returned by the service display.

3. Select a node for a type and click the drop-down arrow to display a list of available options.
4. Click either **DataGridView** to display the data in a grid or **Details** to display the data in individual controls.
5. Drag the node onto the form. The controls are added to the form, together with a [BindingSource](#) component and a [BindingNavigator](#) component.

How to: Configure a service to reuse existing types

When a service reference is added to a project, any types defined in the service are generated in the local project. In many cases, this creates duplicate types when a service uses common .NET types or when types are defined in a shared library.

To avoid this problem, types in referenced assemblies are shared by default. If you want to disable type sharing for one or more assemblies, you can do so in the **Configure Service References** dialog box.

To disable type sharing in a single assembly

1. In **Solution Explorer**, select the service reference.
2. On the **Project** menu, click **Configure Service Reference**.
3. In the **Configure Service References** dialog box, select **Reuse types in specified referenced assemblies**.
4. Select the check box for each assembly in which you want to enable type sharing. To disable type sharing for an assembly, leave the check box cleared.

To disable type sharing in all assemblies

1. In **Solution Explorer**, select the service reference.
2. On the **Project** menu, click **Configure Service Reference**.
3. In the **Configure Service References** dialog box, clear the **Reuse types in referenced assemblies** check box.

Related topics

| TITLE | DESCRIPTION |
|--|---|
| Walkthrough: Creating a simple WCF Service in Windows Forms | Provides a step-by-step demonstration of creating and using WCF services in Visual Studio. |
| Walkthrough: Creating a WCF data service with WPF and Entity Framework | Provides a step-by-step demonstration of how to create and use WCF Data Services in Visual Studio. |
| Using the WCF development tools | Discusses how to create and test WCF services in Visual Studio. |
| | How to: Add, update, or remove a WCF Data Service reference |
| Troubleshooting service references | Presents some common errors that can occur with service references and how to prevent them. |
| Debugging WCF services | Describes common debugging problems and techniques you might encounter when debugging WCF services. |

| TITLE | DESCRIPTION |
|--|--|
| Walkthrough: Creating an n-tier data application | Provides step-by-step instructions for creating a typed dataset and separating the TableAdapter and dataset code into multiple projects. |
| Configure Service Reference dialog box | Describes the user interface elements of the Configure Service Reference dialog box. |

Reference

- [System.ServiceModel](#)
- [System.Data.Services](#)

See also

- [Visual Studio data tools for .NET](#)

Work with a Conceptual Model (WCF Data Services)

8/5/2021 • 2 minutes to read • [Edit Online](#)

When you use a conceptual model to describe the data in a database, you can query data through your objects instead of having to translate back and forth between a database schema and an object model.

You can use conceptual models with WCF Data Services applications. The following topics show how to query data through a conceptual model.

| TOPIC | DESCRIPTION |
|--|---|
| How to: Execute Data Service queries | Shows how to query a data service from a .NET application. |
| How to: Project query results | Shows how to reduce the amount of data returned through a data service query. |

When you use a conceptual model, you can define what kind of data is valid in the language that matches your domain. You can define valid data in the model, or you can add validation to operations that you perform on an entity or data service.

The following topics show how to add validation to WCF Data Services applications.

| TOPIC | DESCRIPTION |
|---|--|
| How to: Intercept Data Service messages | Shows how to add validation to a data service operation. |

The following topics show how to create, update, and delete data by performing operations on entities.

| TOPIC | DESCRIPTION |
|--|--|
| How to: Add, modify, and delete entities | Shows how to create, update, and delete entity data in a data service. |
| How to: Define entity relationships | Shows how to create or change relationships in a data service. |

See also

- [Windows Communication Foundation Services and WCF Data Services in Visual Studio](#)
- [Querying the Data Service](#)

How to: Connect to data in a service

8/5/2021 • 2 minutes to read • [Edit Online](#)

You connect your application to the data returned from a service by running the [Data Source Configuration Wizard](#) and selecting **Service** on the **Choose a Data Source Type** page.

Upon completion of the wizard, a service reference is added to your project and is immediately available in the [Data Sources](#) window.

NOTE

The items that appear in the **Data Sources** window are dependent on the information that the service returns. Some services might not provide enough information for the **Data Source Configuration Wizard** to create bindable objects. For example, if the service returns an untyped dataset, no items appear in the **Data Sources** window upon completing the wizard. This is because untyped datasets do not provide schema, so the wizard does not have enough information to create the data source.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To connect your application to a service

1. On the **Data** menu, click **Add New Data Source**.
2. Select **Service** on the **Choose a Data Source Type** page, and then click **Next**.
3. Enter the address of the service you want to use, or click **Discover** to locate services in the current solution, and then click **Go**.
4. Optionally, you can type a new **Namespace** in place of the default value.

NOTE

Click **Advanced** to open the [Configure Service Reference](#) dialog box.

5. Click **OK** to add a service reference to your project.
6. Click **Finish**.

The data source is added to the **Data Sources** window.

Next steps

To add functionality to your application, select an item in the **Data Sources** window and drag it onto a form to create bound controls. For more information, see [Bind controls to data in Visual Studio](#).

See also

- [Bind WPF controls to a WCF data service](#)
- [Windows Communication Foundation Services and WCF data services in Visual Studio](#)

Walkthrough: Create a simple WCF service in Windows Forms

8/5/2021 • 3 minutes to read • [Edit Online](#)

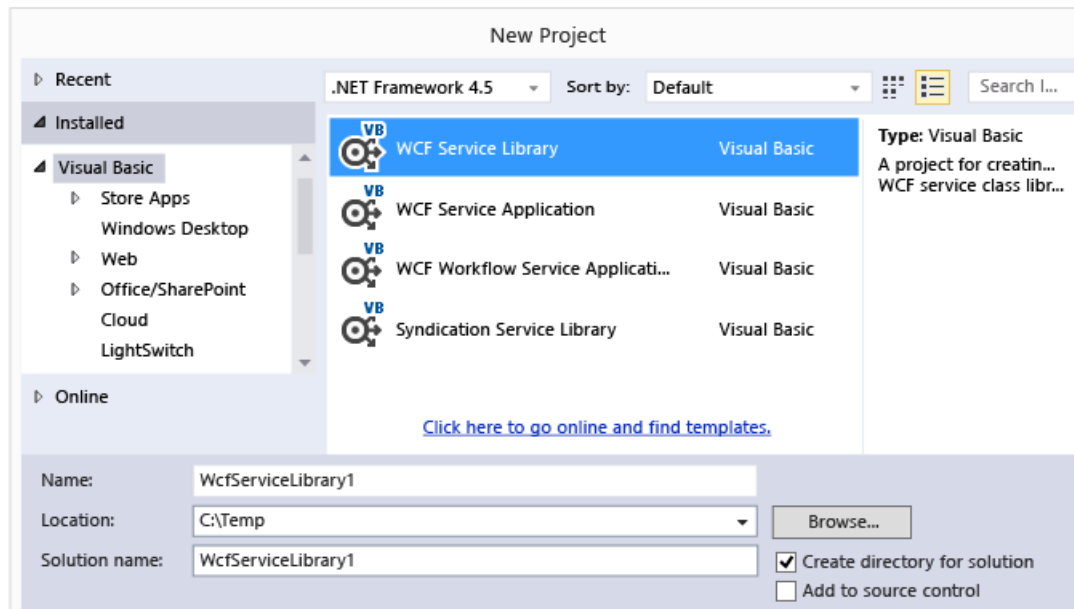
This walkthrough demonstrates how to create a simple Windows Communication Foundation (WCF) service, test it, and then access it from a Windows Forms application.

NOTE

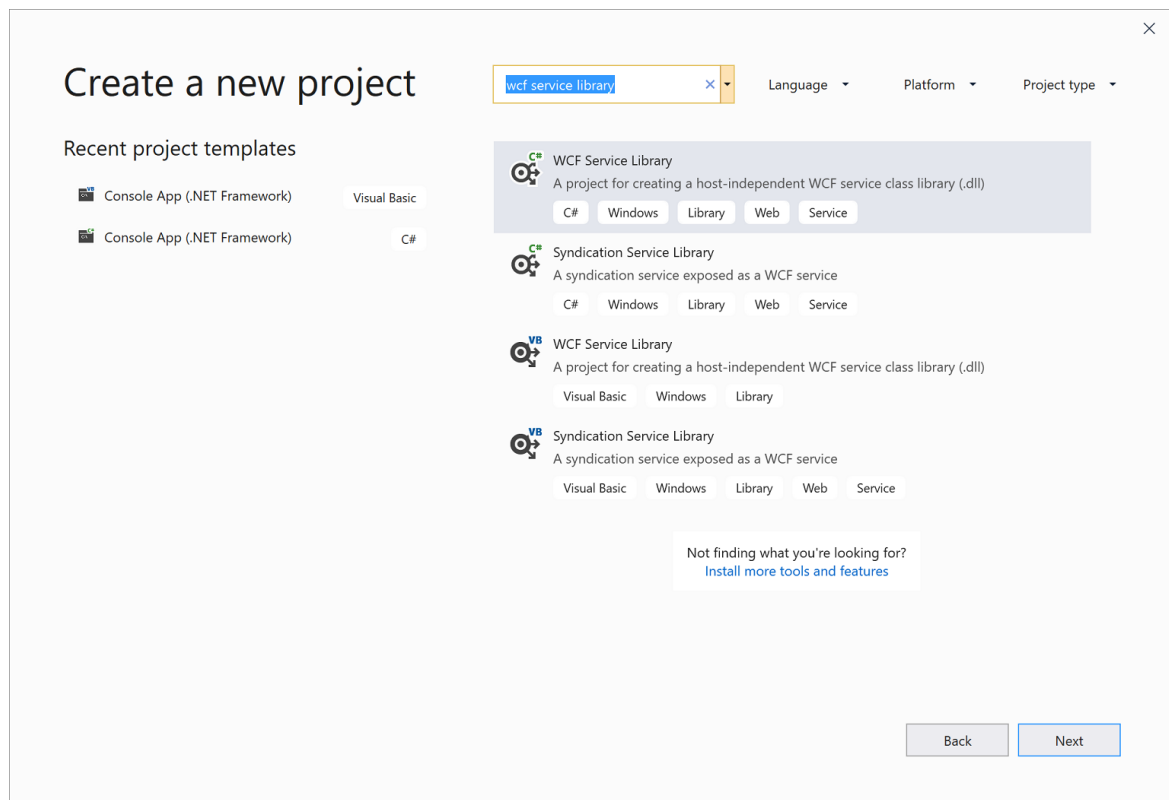
Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

Create a service

1. Open Visual Studio.
2. On the **File** menu, choose **New > Project**.
3. In the **New Project** dialog box, expand the **Visual Basic** or **Visual C#** node and choose **WCF**, followed by **WCF Service Library**.
4. Click **OK** to create the project.



2. On the start window, choose **Create a new project**.
3. Type **wcf service library** in the search box on the **Create a new project** page. Select either the C# or Visual Basic template for **WCF Service Library**, and then click **Next**.



TIP

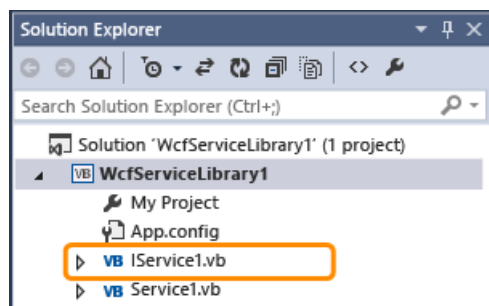
If you don't see any templates, you may need to install the **Windows Communication Foundation** component of Visual Studio. Choose **Install more tools and features** to open Visual Studio Installer. Choose the **Individual components** tab, scroll down to **Development activities**, and then select **Windows Communication Foundation**. Click **Modify**.

4. On the **Configure your new project** page, click **Create**.

NOTE

This creates a working service that can be tested and accessed. The following two steps demonstrate how you might modify the default method to use a different data type. In a real application, you would also add your own functions to the service.

5. In **Solution Explorer**, double-click **IService1.vb** or **IService1.cs**.



Find the following line:

```
[OperationContract]  
string GetData(int value);
```

```
<OperationContract(>>  
Function GetData(ByVal value As Integer) As String
```

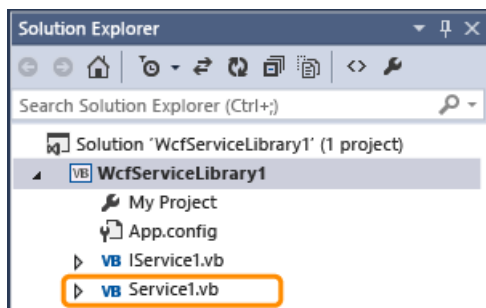
Change the type for the `value` parameter to string:

```
[OperationContract]  
string GetData(string value);
```

```
<OperationContract(>>  
Function GetData(ByVal value As String) As String
```

In the above code, note the `<OperationContract(>>` or `[OperationContract]` attributes. These attributes are required for any method exposed by the service.

6. In **Solution Explorer**, double-click **Service1.vb** or **Service1.cs**.



Find the following line:

```
Public Function GetData(ByVal value As Integer) As String Implements IService1.GetData  
    Return String.Format("You entered: {0}", value)  
End Function
```

```
public string GetData(int value)  
{  
    return string.Format("You entered: {0}", value);  
}
```

Change the type for the `value` parameter to string:

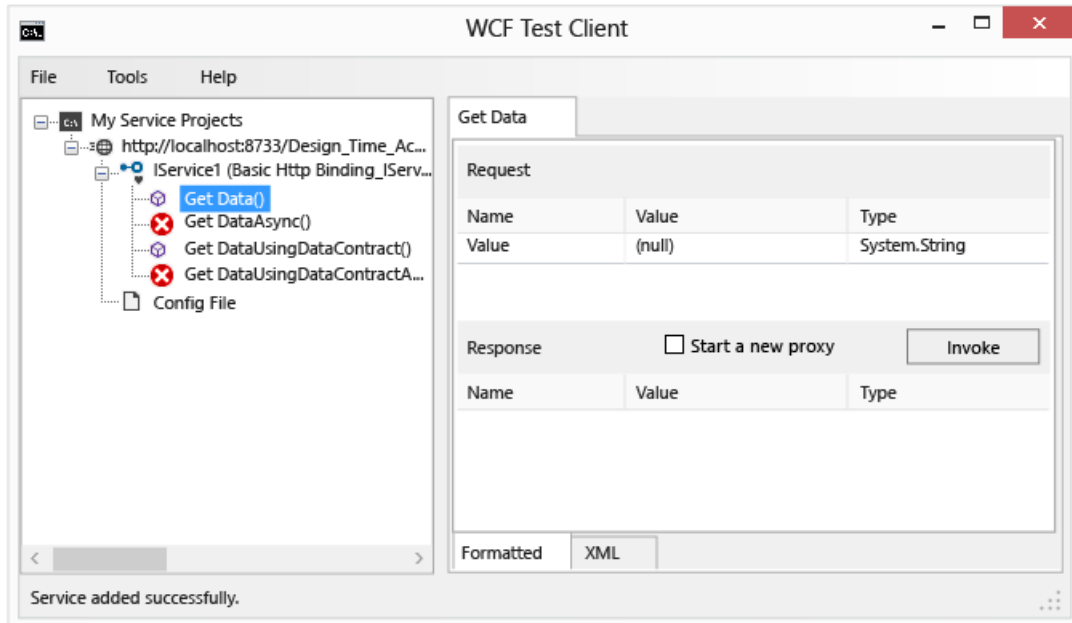
```
public string GetData(string value)  
{  
    return string.Format("You entered: {0}", value);  
}
```

```
Public Function GetData(ByVal value As String) As String Implements IService1.GetData  
    Return String.Format("You entered: {0}", value)  
End Function
```

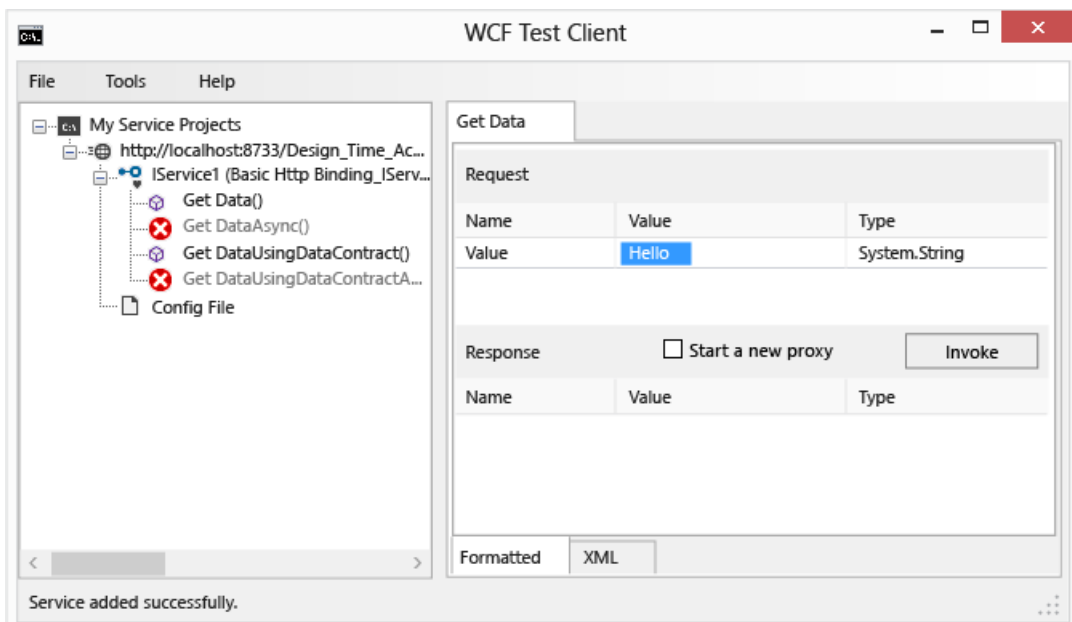
Test the service

1. Press **F5** to run the service. A **WCF Test Client** form appears and loads the service.
2. In the **WCF Test Client** form, double-click the **GetData()** method under **IService1**. The **GetData** tab

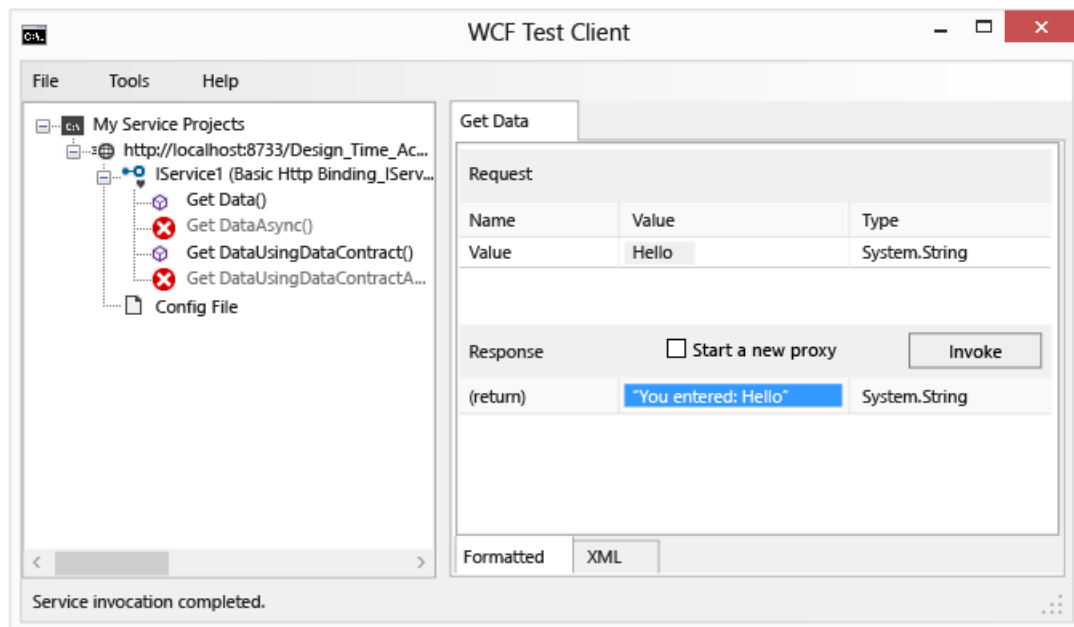
appears.



3. In the **Request** box, select the **Value** field and type `Hello`.



4. Click the **Invoke** button. If a **Security Warning** dialog box appears, click **OK**. The result displays in the **Response** box.

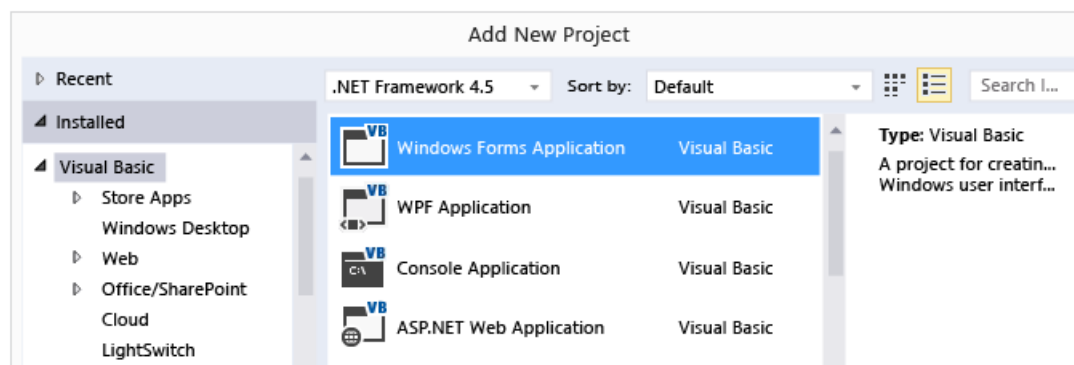


5. On the **File** menu, click **Exit** to close the test form.

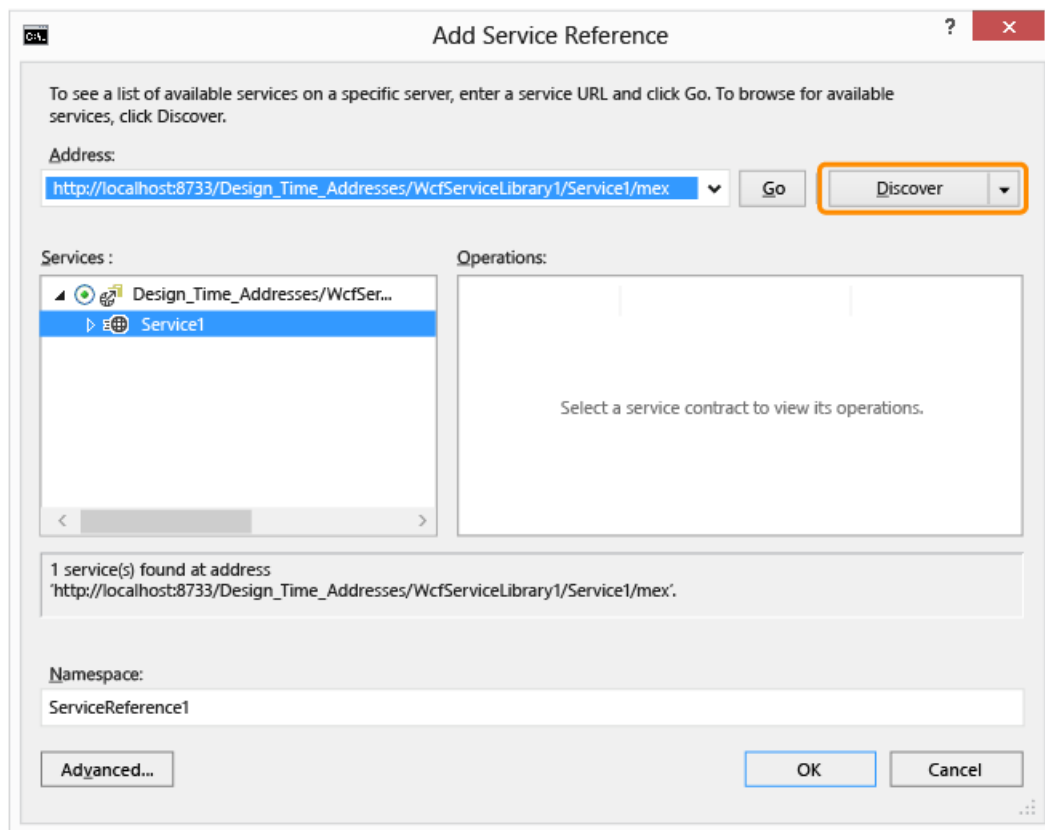
Access the Service

Reference the WCF service

1. On the **File** menu, point to **Add** and then click **New Project**.
2. In the **New Project** dialog box, expand the **Visual Basic** or **Visual C#** node, select **Windows**, and then select **Windows Forms Application**. Click **OK** to open the project.



3. Right-click **WindowsApplication1** and click **Add Service Reference**. The **Add Service Reference** dialog box appears.
4. In the **Add Service Reference** dialog box, click **Discover**.

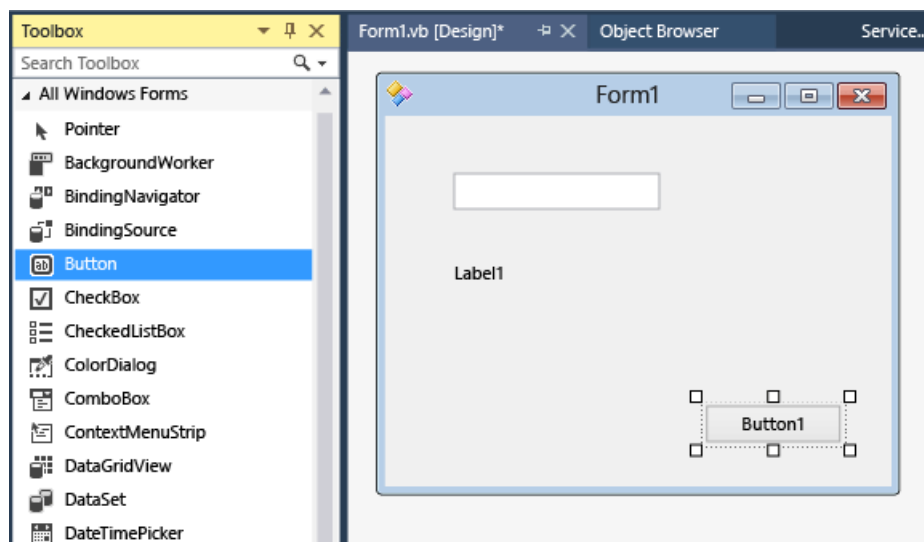


Service1 displays in the Services pane.

5. Click OK to add the service reference.

Build a client application

1. In Solution Explorer, double-click Form1.vb or Form1.cs to open the Windows Forms Designer if it is not already open.
2. From the Toolbox, drag a `TextBox` control, a `Label1` control, and a `Button` control onto the form.



3. Double-click the `Button`, and add the following code in the `Click` event handler:

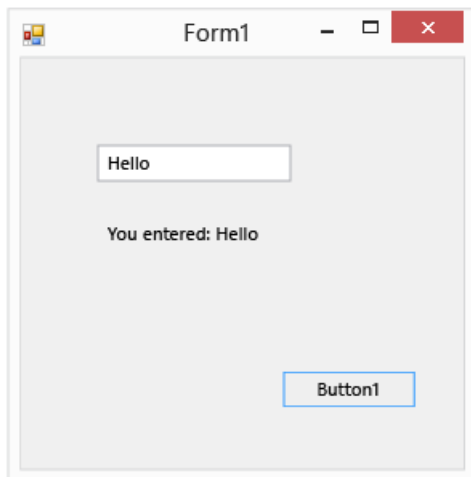
```
private void button1_Click(System.Object sender, System.EventArgs e)
{
    ServiceReference1.Service1Client client = new
        ServiceReference1.Service1Client();
    string returnString;

    returnString = client.GetData(textBox1.Text);
    label1.Text = returnString;
}
```

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    Dim client As New ServiceReference1.Service1Client
    Dim returnString As String

    returnString = client.GetData(TextBox1.Text)
    Label1.Text = returnString
End Sub
```

4. In **Solution Explorer**, right-click **WindowsApplication1** and click **Set as StartUp Project**.
5. Press **F5** to run the project. Enter some text and click the button. The label displays "You entered:" and shows the text that you entered.



See also

- [Windows Communication Foundation Services and WCF Data Services in Visual Studio](#)

Walkthrough: Creating a WCF Data Service with WPF and Entity Framework

8/5/2021 • 8 minutes to read • [Edit Online](#)

This walkthrough demonstrates how to create a simple WCF Data Service that is hosted in an ASP.NET web application and then access it from a Windows Forms application.

In this walkthrough you:

- Create a web application to host a WCF Data Service.
- Create an Entity Data Model that represents the `Customers` table in the Northwind database.
- Create a WCF Data Service.
- Create a client application and add a reference to the WCF Data Service.
- Enable data binding to the service and generate the user interface.
- Optionally add filtering capabilities to the application.

Prerequisites

This walkthrough uses SQL Server Express LocalDB and the Northwind sample database.

1. If you don't have SQL Server Express LocalDB, install it either from the [SQL Server Express download page](#), or through the **Visual Studio Installer**. In the **Visual Studio Installer**, you can install SQL Server Express LocalDB as part of the **Data storage and processing** workload, or as an individual component.
2. Install the Northwind sample database by following these steps:
 - a. In Visual Studio, open the **SQL Server Object Explorer** window. (**SQL Server Object Explorer** is installed as part of the **Data storage and processing** workload in the Visual Studio Installer.) Expand the **SQL Server** node. Right-click on your LocalDB instance and select **New Query**.

A query editor window opens.
 - b. Copy the [Northwind Transact-SQL script](#) to your clipboard. This T-SQL script creates the Northwind database from scratch and populates it with data.
 - c. Paste the T-SQL script into the query editor, and then choose the **Execute** button.

After a short time, the query finishes running and the Northwind database is created.

Creating the Service

To create a WCF Data Service, you will add a web project, create an Entity Data Model, and then create the service from the model.

In the first step, you add a web project to host the service.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

To create the web project

1. On the menu bar, choose **File > New > Project**.
2. In the **New Project** dialog box, expand the **Visual Basic** or **Visual C#** and **Web** nodes, and then choose the **ASP.NET Web Application** template.
3. In the **Name** text box, enter **NorthwindWeb**, and then choose the **OK** button.
4. In the **New ASP.NET Project** dialog box, in the **Select a template** list, choose **Empty**, and then choose the **OK** button.

In the next step, you create an Entity Data Model that represents the `Customers` table in the Northwind database.

To create the Entity Data Model

1. On the menu bar, choose **Project > Add New Item**.
2. In the **Add New Item** dialog box, choose the **Data** node, and then choose the **ADO.NET Entity Data Model** item.
3. In the **Name** text box, enter `NorthwindModel`, and then choose the **Add** button.

The Entity Data Model Wizard appears.

4. In the Entity Data Model Wizard, on the **Choose Model Contents** page, choose the **EF Designer from database** item, and then choose the **Next** button.
5. On the **Choose Your Data Connection** page, perform one of the following steps:
 - If a data connection to the Northwind sample database is available in the drop-down list, choose it.
 - or-
 - Choose the **New Connection** button to configure a new data connection. For more information, see [Add new connections](#).
6. If the database requires a password, choose the **Yes, include sensitive data in the connection string** option button, and then choose the **Next** button.

NOTE

If a dialog box appears, choose **Yes** to save the file to your project.

7. On the **Choose your version** page, choose the **Entity Framework 5.0** option button, and then choose the **Next** button.

NOTE

In order to use the latest version of the Entity Framework 6 with WCF Services, you'll need to install the WCF Data Services Entity Framework Provider NuGet package. See [Using WCF Data Services 5.6.0 with Entity Framework 6+](#).

- On the **Choose Your Database Objects** page, expand the **Tables** node, select the **Customers** check box, and then choose the **Finish** button.

The entity model diagram displays, and a *NorthwindModel.edmx* file is added to your project.

In the next step, you create and test the data service.

To create the data service

- On the menu bar, choose **Project > Add New Item**.
- In the **Add New Item** dialog box, choose the **Web** node, and then choose the **WCF Data Service 5.6** item.
- In the **Name** text box, enter `NorthwindCustomers`, and then choose the **Add** button.

The **NorthwindCustomers.svc** file appears in the **Code Editor**.

- In the **Code Editor**, locate the first `TODO:` comment and replace the code with the following:

```
Inherits DataService(Of northwindEntities)
```

```
public class NorthwindCustomers : DataService<northwindEntities>
```

- Replace the comments in the `InitializeService` event handler with the following code:

```
config.SetEntitySetAccessRule("*", EntitySetRights.All)
```

```
config.SetEntitySetAccessRule("*", EntitySetRights.All);
```

- On the menu bar, choose **Debug > Start Without Debugging** to run the service. A browser window opens and the XML schema for the service displays.
- In the **Address** bar, enter `Customers` at the end of the URL for **NorthwindCustomers.svc**, and then choose the **Enter** key.

An XML representation of the data in the `Customers` table appears.

NOTE

In some cases, Internet Explorer will misinterpret the data as an RSS feed. You must make sure that the option to display RSS feeds is disabled. For more information, see [Troubleshooting service references](#).

- Close the browser window.

In the next steps, you create a Windows Forms client application to consume the service.

Creating the Client Application

To create the client application, you add a second project, add a service reference to the project, configure a data source, and create a user interface to display the data from the service.

In the first step, you add a Windows Forms project to the solution and set it as the startup project.

To create the client application

1. On the menu bar, choose File, **Add > New Project**.
2. In the **New Project** dialog box, expand the **Visual Basic** or **Visual C#** node, choose the **Windows** node, and then choose **Windows Forms Application**.
3. In the **Name** text box, enter `NorthwindClient`, and then choose the **OK** button.
4. In **Solution Explorer**, choose the **NorthwindClient** project node.
5. On the menu bar, choose **Project, Set as StartUp Project**.

In the next step, you add a service reference to the WCF Data Service in the web project.

To add a service reference

1. On the menu bar, choose **Project > Add Service Reference**.
2. In the **Add Service Reference** dialog box, choose the **Discover** button.

The URL for the NorthwindCustomers service appears in the **Address** field.

3. Choose the **OK** button to add the service reference.

In the next step, you configure a data source to enable data binding to the service.

To enable data binding to the service

1. On the menu bar, choose **View > Other Windows > Data Sources**.

The **Data Sources** window opens.

2. In the **Data Sources** window, choose the **Add New Data Source** button.
3. On the **Choose a Data Source Type** page of the **Data Source Configuration Wizard**, choose **Object**, and then choose the **Next** button.
4. On the **Select the Data Objects** page, expand the **NorthwindClient** node, and then expand the **NorthwindClient.ServiceReference1** node.
5. Select **Customer** check box, and then choose the **Finish** button.

In the next step, you create the user interface that displays the data from the service.

To create the user interface

1. In the **Data Sources** window, open the shortcut menu for the **Customers** node and choose **Copy**.
2. In the **Form1.vb** or **Form1.cs** form designer, open the shortcut menu and choose **Paste**.

A **DataGridView** control, a **BindingSource** component, and a **BindingNavigator** component are added to the form.

3. Choose the **CustomersDataGridView** control, and then in the **Properties** window set the **Dock** property to **Fill**.
4. In **Solution Explorer**, open the shortcut menu for the **Form1** node and choose **View Code** to open the Code Editor, and add the following `Imports` or `Using` statement at the top of the file:

```
Imports NorthwindClient.ServiceReference1
```

```
using NorthwindClient.ServiceReference1;
```

5. Add the following code to the `Form1_Load` event handler:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Dim proxy As New NorthwindEntities _
(New Uri("http://localhost:53161/NorthwindCustomers.svc/"))
    Me.CustomersBindingSource.DataSource = proxy.Customers
End Sub
```

```
private void Form1_Load(object sender, EventArgs e)
{
    NorthwindEntities proxy = new NorthwindEntities(new
Uri("http://localhost:53161/NorthwindCustomers.svc/"));
    this.CustomersBindingSource.DataSource = proxy.Customers;
}
```

6. In **Solution Explorer**, open the shortcut menu for the **NorthwindCustomers.svc** file and choose **View in Browser**. Internet Explorer opens and the XML schema for the service displays.
7. Copy the URL from the Internet Explorer address bar.
8. In the code that you added in step 4, select `http://localhost:53161/NorthwindCustomers.svc/` and replace it with the URL that you just copied.
9. On the menu bar, choose **Debug > Start Debugging** to run the application. The customer information is shown.

You now have a working application that displays a list of customers from the NorthwindCustomers service. If you want to expose additional data through the service, you can modify the Entity Data Model to include additional tables from the Northwind database.

In the next optional step, you learn how to filter the data that is returned by the service.

Adding Filtering Capabilities

In this step, you customize the application to filter the data by the customer's city.

To add filtering by city

1. In **Solution Explorer**, open the shortcut menu for the **Form1.vb** or **Form1.cs** node and choose **Open**.
2. Add a **TextBox** control and a **Button** control from the **Toolbox** to the form.
3. Open the shortcut menu for the **Button** control, choose **View Code**, and then add the following code in the `Button1_Click` event handler:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim proxy As New northwindEntities _
(New Uri("http://localhost:53161/NorthwindCustomers.svc"))
    Dim city As String = TextBox1.Text

    If city <> "" Then
        Me.CustomersBindingSource.DataSource = From c In _
        proxy.Customers Where c.City = city
    End If

End Sub
```

```
private void Button1_Click(object sender, EventArgs e)
{
    ServiceReference1.northwindModel.northwindEntities proxy = new northwindEntities(new
    Uri("http://localhost:53161/NorthwindCustomers.svc"));
    string city = TextBox1.Text;

    if (!string.IsNullOrEmpty(city)) {
        this.CustomersBindingSource.DataSource = from c in proxy.Customers where c.City == city;
    }

}
```

4. In the previous code, replace `http://localhost:53161/NorthwindCustomers.svc` with the URL from the `Form1_Load` event handler.
5. On the menu bar, choose **Debug > Start Debugging** to run the application.
6. In the text box, enter **London**, and then choose the button. Only the customers from London are displayed.

See also

- [Windows Communication Foundation Services and WCF Data Services in Visual Studio](#)
- [How to: Add, update, or remove a WCF Data Service reference](#)

Troubleshoot service references

8/5/2021 • 3 minutes to read • [Edit Online](#)

This topic lists common issues that may occur when you are working with Windows Communication Foundation (WCF) or WCF Data Services references in Visual Studio.

Error returning data from a service

When you return a `DataSet` or `DataTable` from a service, you may receive a "The maximum size quota for incoming messages has been exceeded" exception. By default, the `MaxReceivedMessageSize` property for some bindings is set to a relatively small value to limit exposure to denial-of-service attacks. You can increase this value to prevent the exception. For more information, see [MaxReceivedMessageSize](#).

To fix this error:

1. In **Solution Explorer**, double-click the *app.config* file to open it.
2. Locate the `MaxReceivedMessageSize` property and change it to a larger value.

Cannot find a service in my solution

When you click the **Discover** button in the **Add Service References** dialog box, one or more WCF Service Library projects in the solution do not appear in the services list. This can occur if a Service Library has been added to the solution but has not yet been compiled.

To fix this error:

- In **Solution Explorer**, right-click the WCF Service Library project and click **Build**.

Error accessing a service over a remote desktop

When a user accesses a Web-hosted WCF service over a remote desktop connection and the user does not have administrative permissions, NTLM authentication is used. If the user does not have administrative permissions, the user may receive the following error message: "The HTTP request is unauthorized with client authentication scheme 'Anonymous'. The authentication header received from the server was 'NTLM'."

To fix this error:

1. In the website project, open the **Properties** pages.
2. On the **Start Options** tab, clear the **NTLM Authentication** check box.

NOTE

You should turn off NTLM authentication only for websites that exclusively contain WCF services. Security for WCF services is managed through the configuration in the *web.config* file. This makes NTLM authentication unnecessary.

Access level for generated classes setting has no effect

Setting the **Access level for generated classes** option in the **Configure Service References** dialog box to **Internal** or **Friend** may not always work. Even though the option appears to be set in the dialog box, the

resulting support classes are generated with an access level of `Public`.

This is a known limitation of certain types, such as those serialized using the [XmlSerializer](#).

Error debugging service code

When you step into the code for a WCF service from client code, you may receive an error related to missing symbols. This can occur when a service that was part of your solution was moved or removed from the solution.

When you first add a reference to a WCF service that is part of the current solution, an explicit build dependency is added between the service project and the service client project. This guarantees that the client always accesses up-to-date service binaries, which is especially important for debugging scenarios such as stepping from client code into service code.

If the service project is removed from the solution, this explicit build dependency is invalidated. Visual Studio can no longer guarantee that the service project is rebuilt as necessary.

To fix this error, you have to manually rebuild the service project:

1. On the **Tools** menu, click **Options**.
2. In the **Options** dialog box, expand **Projects and Solutions**, and then select **General**.
3. Make sure that the **Show advanced build configurations** check box is selected, and then click **OK**.
4. Load the WCF service project.
5. In the **Configuration Manager** dialog box, set the **Active solution configuration** to **Debug**. For more information, see [How to: Create and edit configurations](#).
6. In **Solution Explorer**, select the WCF service project.
7. On the **Build** menu, click **Rebuild** to rebuild the WCF service project.

WCF Data Services do not display in the browser

When it attempts to view an XML representation of data in a WCF Data Service, Internet Explorer may misinterpret the data as an RSS feed. Make sure that the option to display RSS feeds is disabled.

To fix this error, disable RSS feeds:

1. In Internet Explorer, on the **Tools** menu, click **Internet Options**.
2. On the **Content** tab, in the **Feeds** section, click **Settings**.
3. In the **Feed Settings** dialog box, clear the **Turn on feed reading view** check box, and then click **OK**.
4. Click **OK** to close the **Internet Options** dialog box.

See also

- [Windows Communication Foundation Services and WCF Data Services in Visual Studio](#)

Configure Service Reference dialog box

8/5/2021 • 2 minutes to read • [Edit Online](#)

The **Configure Service Reference** dialog box enables you to configure the behavior of Windows Communication Foundation (WCF) services.

To access the **Configure Service Reference** dialog box, right-click a service reference in **Solution Explorer** and choose **Configure Service Reference**. You can also access the dialog box by clicking the **Advanced** button in the **Add Service Reference Dialog Box**.

Task list

- To change the address where a WCF service is hosted, enter the new address in the **Address** field.
- To change the access level for classes in a WCF client, select an access-level keyword in the **Access level for generated classes** list.
- To call the methods of a WCF service asynchronously, select the **Generate asynchronous operations** check box.
- To generate message contract types in a WCF client, select the **Always generate message contracts** check box.
- To specify list or dictionary collection types for a WCF client, select the types from the **Collection type** and **Dictionary collection type** lists.
- To disable type sharing, clear the **Reuse types in referenced assemblies** check box. To enable type sharing for a subset of referenced assemblies, select the **Reuse types in referenced assemblies** check box, select **Reuse types in specified referenced assemblies**, and select the desired references in the **Referenced assemblies** list.

UIElement list

Address

Updates the web address where a service reference looks for a service. For example, during development, the service may be hosted on a development server and then later moved to a production server, necessitating an address change.

NOTE

The Address element is not available when the **Configure Service Reference** dialog box is displayed from the **Add Service Reference Dialog Box**.

Access level for generated classes

Determines the code access level for WCF client classes.

NOTE

For Website projects, this option is always set to `Public` and cannot be changed. For more information, see [Troubleshooting service references](#).

Generate asynchronous operations

Determines whether WCF service methods is called synchronously (the default) or asynchronously.

Generate task-based operations

When writing async code, this option lets you take advantage of the Task Parallel Library (TPL) that was introduced with .NET 4. See [Task Parallel Library \(TPL\)](#).

Always generate message contracts

Determines whether message contract types are generated for a WCF client. For more information about message contracts, see [Using message contracts](#).

Collection type

Specifies the list collection type for a WCF client. The default type is [Array](#).

Dictionary collection type

Specifies the dictionary collection type for a WCF client. The default type is [Dictionary<TKey,TValue>](#).

Reuse types in referenced assemblies

Determines whether a WCF client tries to reuse what already exists in referenced assemblies instead of generating new types when a service is added or updated. By default, this option is checked.

Reuse types in all referenced assemblies

When selected, all types in the **Referenced assemblies** list are reused if possible. By default, this option is selected.

Reuse types in specified referenced assemblies

When selected, only the selected types in the **Referenced assemblies** list are reused.

Referenced assemblies list

Contains a list of referenced assemblies for the project or website. When you select **Reuse types in specified referenced assemblies**, you can select or clear individual assemblies.

Add Web Reference

Displays the **Add Web Reference** dialog box.

NOTE

This option should only be used for projects that target version 2.0 of the .NET Framework.

NOTE

The **Add Web Reference** button is only available when the **Configure Service Reference** dialog box is displayed from the **Add Service Reference Dialog Box**.

See also

- [How to: Add a reference to a web service](#)
- [Windows Communication Foundation Services and WCF Data Services](#)

How to: Add, update, or remove a WCF data service reference

8/5/2021 • 4 minutes to read • [Edit Online](#)

A *service reference* enables a project to access one or more WCF Data Services. Use the **Add Service Reference** dialog box to search for WCF Data Services in the current solution, locally, on a local area network, or on the Internet.

You can use the **Connected Services** node in **Solution Explorer** to access the **Microsoft WCF Web Service Reference Provider**, which lets you manage Windows Communication Foundation (WCF) data service references.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in this article. You may be using a different edition of Visual Studio or different environment settings. For more information, see [Personalize the IDE](#).

Add a WCF service reference

To add a reference to an external service

1. In **Solution Explorer**, right-click the name of the project to which you want to add the service, and then click **Add Service Reference**.

The **Add Service Reference** dialog box appears.

2. In the **Address** box, enter the URL for the service, and then click **Go** to search for the service. If the service implements user name and password security, you may be prompted for a user name and password.

NOTE

You should only reference services from a trusted source. Adding references from an untrusted source may compromise security.

You can also select the URL from the **Address** list, which stores the previous 15 URLs at which valid service metadata was found.

A progress bar displays when the search is being performed. You can stop the search at any time by clicking **Stop**.

3. In the **Services** list, expand the node for the service that you want to use and select an entity set.
4. In the **Namespace** box, enter the namespace that you want to use for the reference.
5. Click **OK** to add the reference to the project.

A service client (proxy) is generated, and metadata that describes the service is added to the *app.config* file.

1. In **Solution Explorer**, double-click or tap the **Connected Services** node.

The **Configure Services** tab opens.

2. Choose **Microsoft WCF Web Service Reference Provider**.

The **Configure WCF Web Service Reference** dialog appears.

Configure WCF Web Service Reference
Specify the service to add

Service Endpoint
Data Type Options
Client Options

To see a list of available services on a specific server, enter a service URL and select Go. To find available services in the solution, select Discover. To load a service metadata from a WSDL file, select Browse.

URI:

Services: Operations:

Status:

Namespace:

[More Information](#)

3. In the **URI** box, enter the URL for the service, and then click **Go** to search for the service. If the service implements user name and password security, you may be prompted for a user name and password.

NOTE

You should only reference services from a trusted source. Adding references from an untrusted source may compromise security.

You can also select the URL from the **URI** list, which stores the previous 15 URLs at which valid service metadata was found.

A progress bar displays when the search is being performed. You can stop the search at any time by clicking **Stop**.

4. In the **Services** list, expand the node for the service that you want to use and select an entity set.
5. In the **Namespace** box, enter the namespace that you want to use for the reference.
6. Click **Finish** to add the reference to the project.

A service client (proxy) is generated, and metadata that describes the service is added to the *app.config* file.

To add a reference to a service in the current solution

1. In **Solution Explorer**, right-click the name of the project to which you want to add the service, and then click **Add Service Reference**.

The **Add Service Reference** dialog box appears.

2. Click **Discover**.

All services (both WCF Data Services and WCF services) in the current solution are added to the **Services** list.

3. In the **Services** list, expand the node for the service that you want to use and select an entity set.
4. In the **Namespace** box, enter the namespace that you want to use for the reference.
5. Click **OK** to add the reference to the project.

A service client (proxy) generates, and metadata that describes the service is added to the *app.config* file.

1. In **Solution Explorer**, double-click or tap the **Connected Services** node.

The **Configure Services** tab opens.

2. Choose **Microsoft WCF Web Service Reference Provider**.

The **Configure WCF Web Service Reference** dialog appears.

3. Click **Discover**.

All services (both WCF Data Services and WCF services) in the current solution are added to the **Services** list.

4. In the **Services** list, expand the node for the service that you want to use and select an entity set.
5. In the **Namespace** box, enter the namespace that you want to use for the reference.
6. Click **Finish** to add the reference to the project.

A service client (proxy) generates, and metadata that describes the service is added to the *app.config* file.

Update a service reference

The Entity Data Model for a WCF Data Services sometimes changes. When this happens, you must update the service reference.

To update a service reference

- In **Solution Explorer**, right-click the service reference and then click **Update Service Reference**.

A progress dialog box displays while the reference is updated from its original location, and the service client is regenerated to reflect any changes in the metadata.

Remove a service reference

If a service reference is no longer being used, you can remove it from your solution.

To remove a service reference

- In **Solution Explorer**, right-click the service reference and then click **Delete**.

The service client will be removed from the solution, and the metadata that describes the service will be removed from the *app.config* file.

NOTE

Any code that references the service reference must be removed manually.

See also

- [Windows Communication Foundation Services and WCF data services in Visual Studio](#)

Upgrade .mdf files

8/5/2021 • 3 minutes to read • [Edit Online](#)

This topic describes the options for upgrading a database file (.mdf) after you install a newer version of Visual Studio. It includes instructions for the following tasks:

- Upgrade a database file to use a newer version of SQL Server Express LocalDB
- Upgrade a database file to use a newer version of SQL Server Express
- Work with a database file in Visual Studio but retain compatibility with an older version of SQL Server Express or LocalDB
- Make SQL Server Express the default database engine

You can use Visual Studio to open a project that contains a database file (.mdf) that was created by using an older version of SQL Server Express or LocalDB. However, to continue to develop your project in Visual Studio, you must have that version of SQL Server Express or LocalDB installed on the same machine as Visual Studio, or you must upgrade the database file. If you upgrade the database file, you won't be able to access it by using older versions of SQL Server Express or LocalDB.

You may also be prompted to upgrade a database file that was created through an earlier version of SQL Server Express or LocalDB if the version of the file isn't compatible with the instance of SQL Server Express or LocalDB that's currently installed. To resolve the issue, Visual Studio will prompt you to upgrade the file.

IMPORTANT

We recommend that you back up the database file before you upgrade it.

WARNING

If you upgrade an .mdf file that was created in LocalDB 2014 (V12) 32 bit to LocalDB 2016 (V13) or later, you will not be able to open the file again in the 32-bit version of LocalDB.

Before you upgrade a database, consider the following criteria:

- Don't upgrade if you want to work on your project in both an older version and a newer version of Visual Studio.
- Don't upgrade if your application will be used in environments that use SQL Server Express rather than LocalDB.
- Don't upgrade if your application uses remote connections, because LocalDB doesn't accept them.
- Don't upgrade if your application relies on Internet Information Services (IIS).
- Consider upgrading if you want to test database applications in a sandbox environment but don't want to administer a database.

To upgrade a database file to use the LocalDB version

1. In **Server Explorer**, select the **Connect to Database** button.
2. In the **Add Connection** dialog box, specify the following information:

- **Data Source:** `Microsoft SQL Server (SqlClient)`
- **Server Name:**
 - To use the default version: `(localdb)\MSSQLLocalDB`. This will specify either ProjectV12 or ProjectV13, depending on which version of Visual Studio is installed and when the first LocalDB instance was created. The **MSSQLLocalDB** node in **SQL Server Object Explorer** shows which version it is pointing to.
 - To use a specific version: `(localdb)\ProjectsV12` or `(localdb)\ProjectsV13`, where V12 is LocalDB 2014 and V13 is LocalDB 2016.
- **Attach a database file:** The physical path of the primary *.mdf* file.
- **Logical Name:** The name that you want to use with the file.

3. Select the **OK** button.

4. When you're prompted, select the **Yes** button to upgrade the file.

The database is upgraded, is attached to the LocalDB database engine, and is no longer compatible with the older version of LocalDB.

You can also modify a SQL Server Express connection to use LocalDB by opening the shortcut menu for the connection and then selecting **Modify Connection**. In the **Modify Connection** dialog box, change the server name to `(LocalDB)\MSSQLLocalDB`. In the **Advanced Properties** dialog box, make sure that **User Instance** is set to **False**.

To upgrade a database file to use the SQL Server Express version

1. On the shortcut menu for the connection to the database, select **Modify Connection**.
2. In the **Modify Connection** dialog box, select the **Advanced** button.
3. In the **Advanced Properties** dialog box, select the **OK** button without changing the server name.

The database file is upgraded to match the current version of SQL Server Express.

To work with the database in Visual Studio but retain compatibility with SQL Server Express

- In Visual Studio, open the project without upgrading it.
 - To run the project, select the **F5** key.
 - To edit the database, open the *.mdf* file in **Solution Explorer**, and expand the node in **Server Explorer** to work with your database.

To make SQL Server Express the default database engine

1. On the menu bar, select **Tools > Options**.
2. In the **Options** dialog box, expand the **Database Tools** options, and then select **Data Connections**.
3. In the **SQL Server Instance Name** text box, specify the name of the instance of SQL Server Express or LocalDB that you want to use. If the instance isn't named, specify `.\SQLEXPRESS` or `(LocalDB)\MSSQLLocalDB`.
4. Select the **OK** button.

SQL Server Express will be the default database engine for your applications.

See also

- [Accessing data in Visual Studio](#)

DataContext Methods (O/R Designer)

8/5/2021 • 4 minutes to read • [Edit Online](#)

DataContext methods (in the context of the [LINQ to SQL Tools in Visual Studio](#)) are methods of the **DataContext** class that run stored procedures and functions in a database.

The **DataContext** class is a LINQ to SQL class that acts as a conduit between a SQL Server database and the LINQ to SQL entity classes mapped to that database. The **DataContext** class contains the connection string information and the methods for connecting to a database and manipulating the data in the database. By default, the **DataContext** class contains several methods that you can call, such as the [SubmitChanges](#) method that sends updated data from LINQ to SQL classes to the database. You can also create additional **DataContext** methods that map to stored procedures and functions. In other words, calling these custom methods runs the stored procedure or function in the database to which the **DataContext** method is mapped. You can add new methods to the **DataContext** class just as you would add methods to extend any class. However, in discussions about **DataContext** methods in the context of the **O/R Designer**, it is the **DataContext** methods that map to stored procedures and functions that are being discussed.

Methods pane

DataContext methods that map to stored procedures and functions are displayed in the **Methods** pane of the **O/R Designer**. The **Methods** pane is the pane along the side of the **Entities** pane (the main design surface). The **Methods** pane lists all **DataContext** methods that you created by using the **O/R Designer**. By default, the **Methods** pane is empty; drag stored procedures or functions from **Server Explorer** or **Database Explorer** onto the **O/R Designer** to create **DataContext** methods and populate the **Methods** pane. For more information, see [How to: Create DataContext methods mapped to stored procedures and functions \(O/R Designer\)](#).

NOTE

Open and close the methods pane by right-clicking the **O/R Designer** and then clicking **Hide Methods Pane** or **Show Methods Pane**, or use the keyboard shortcut **CTRL+1**.

Two types of DataContext methods

DataContext methods are those methods that map to stored procedures and functions in the database. You can create and add **DataContext** methods on the **Methods** pane of the **O/R Designer**. There are two distinct types of **DataContext** methods; those that return one or more result sets, and those that do not:

- **DataContext** methods that return one or more result sets:

Create this kind of **DataContext** method when your application just needs to run stored procedures and functions in the database and return the results. For more information, see [How to: Create DataContext methods mapped to stored procedures and functions \(O/R Designer\)](#), [System.Data.Linq.ISingleResult<T>](#), and [IMultipleResults](#).

- **DataContext** methods that do not return result sets: such as Inserts, Updates, and Deletes for a specific entity class.

Create this kind of **DataContext** method when your application has to run stored procedures instead of using the default LINQ to SQL behavior for saving modified data between an entity class and the database. For more information, see [How to: Assign stored procedures to perform updates, inserts, and](#)

[deletes \(O/R Designer\)](#).

Return Types of DataContext Methods

When you drag stored procedures and functions from **Server Explorer** or **Database Explorer** onto the **O/R Designer**, the return type of the generated [DataContext](#) method differs depending on where you drop the item. Dropping the items directly onto an existing entity class creates a [DataContext](#) method with the return type of the entity class; dropping items onto an empty area of the **O/R Designer** (in either pane) creates a [DataContext](#) method that returns an automatically generated type. The automatically generated type has the name that matches the stored procedure or function name and properties, which map to the fields returned by the stored procedure or function.

NOTE

You can change the return type of a [DataContext](#) method after you add it to the methods pane. To inspect or change the return type of a [DataContext](#) method, select it and inspect the **Return Type** property in the **Properties** window. For more information, see [How to: Change the return type of a DataContext method \(O/R Designer\)](#).

Objects you drag from the database onto the O/R Designer surface are named automatically, based on the name of the objects in the database. If you drag the same object more than once, a number is added to the end of the new name that differentiates the names. When database object names contain spaces, or characters not-supported in Visual Basic or C#, the space or invalid character is replaced with an underscore.

See also

- [LINQ to SQL tools in Visual Studio](#)
- [LINQ to SQL](#)
- [Stored procedures](#)
- [How to: Create DataContext methods mapped to stored procedures and functions \(O/R Designer\)](#)
- [How to: Assign stored procedures to perform updates, inserts, and deletes \(O/R Designer\)](#)
- [Walkthrough: Customizing the insert, update, and delete behavior of entity classes](#)
- [Walkthrough: Creating LINQ to SQL classes \(O-R Designer\)](#)

Data class inheritance (O/R Designer)

8/5/2021 • 2 minutes to read • [Edit Online](#)

Like other objects, LINQ to SQL classes can use inheritance and be derived from other classes. In code, you can specify inheritance relationships between objects by declaring that one class inherits from another. In a database, inheritance relationships are created in several ways. The **Object Relational Designer (O/R Designer)** supports the concept of single-table inheritance as it is often implemented in relational systems.

In single-table inheritance, there is a single database table that contains columns for both base and derived classes. With relational data, a discriminator column contains the value that determines which class any given record belongs to. For example, consider a `Persons` table that contains everyone employed by a company. Some people are employees and some people are managers. The `Persons` table contains a column named `Type` that has a value of 1 for managers and a value of 2 for employees. The `Type` column is the discriminator column. In this scenario, you can create a subclass of employees and populate the class with only records that have a `Type` value of 2.

When you configure inheritance in entity classes by using the O/R Designer, drag the single table that contains the inheritance data onto the designer two times: one time for each class in the inheritance hierarchy. After you add the tables to the designer, connect them with an Inheritance item from the **Object Relational Designer** toolbox and then set the four inheritance properties in the **Properties** window.

Inheritance properties

The following table lists the inheritance properties and their descriptions:

| PROPERTY | DESCRIPTION |
|--|--|
| Discriminator Property | The property (mapped to the column) that determines which class the current record belongs to. |
| Base Class Discriminator Value | The value (in the column designated as the Discriminator Property) that determines that a record is of the base class. |
| Derived Class Discriminator Value | The value (in the property designated as the Discriminator Property) that determines that a record is of the derived class. |
| Inheritance Default | The class that is populated when the value in the property designated as the Discriminator Property does not match either the Base Class Discriminator Value or the Derived Class Discriminator Value . |

Creating an object model that uses inheritance and corresponds to relational data can be somewhat confusing. This topic provides information about the basic concepts and individual properties that are required for configuring inheritance. The following topics provide a clearer explanation of how to configure inheritance with the O/R Designer.

| TOPIC | DESCRIPTION |
|---|--|
| How to: Configure inheritance by using the O/R Designer | Describes how to configure entity classes that use single-table inheritance by using the O/R Designer . |

| TOPIC | DESCRIPTION |
|--|--|
| Walkthrough: Creating LINQ to SQL classes by using single-table inheritance (O/R Designer) | Provides step-by-step instructions on how to configure entity classes that use single-table inheritance by using the O/R Designer . |

See also

- [LINQ to SQL tools in Visual Studio](#)
- [Walkthrough: Creating LINQ to SQL classes \(O-R Designer\)](#)
- [Walkthrough: Creating LINQ to SQL classes by using single-table inheritance \(O/R Designer\)](#)
- [Getting started](#)

The selected class cannot be deleted because it is used as a return type for one or more DataContext methods

8/5/2021 • 2 minutes to read • [Edit Online](#)

The return type of one or more [DataContext](#) methods is the selected entity class. Deleting an entity class that is used as the return type for a [DataContext](#) method causes the compilation of the project to fail. To delete the selected entity class, identify the [DataContext](#) methods that use it and set their return types to a different entity class.

To revert the return types of [DataContext](#) methods to their original auto-generated types, first delete the [DataContext](#) method from the **Methods** pane and then drag the object from **Server Explorer/Database Explorer** onto the **O/R Designer** again.

To correct this error

1. Identify [DataContext](#) methods that use the entity class as a return type by selecting a [DataContext](#) method in the **Methods** pane and inspecting the **Return Type** property in the **Properties** window.
2. Set the **Return Type** to a different entity class or delete the [DataContext](#) method from the methods pane.

See also

- [LINQ to SQL tools in Visual Studio](#)

The connection string contains credentials with a clear text password and is not using integrated security

8/5/2021 • 2 minutes to read • [Edit Online](#)

Do you want to save the connection string to the current DBML file and application configuration files with this sensitive information? Click **No** to save the connection string without the sensitive information.

When working with data connections that include sensitive information (passwords that are included in the connection string), you are given the option of saving the connection string into a project's DBML file and application configuration file with or without the sensitive information.

WARNING

Explicitly setting the **Connection** properties **Application Settings** property to **False** will add the password to the DBML file.

Save options

- To save the connection string with the sensitive information, choose **Yes**.

The connection string is stored as an application setting. The connection string includes the sensitive information in plain text. The DBML file does not contain the sensitive information.

- To save the connection string without the sensitive information, choose **No**.

The connection string is stored as an application setting, but the password is not included.

See also

- [LINQ to SQL tools in Visual Studio](#)

The property <property name> cannot be deleted because it is participating in the association <association name>

8/5/2021 • 2 minutes to read • [Edit Online](#)

The selected property is set as the **Association Property** for the association between the classes indicated in the error message. Properties cannot be deleted if they are participating in an association between data classes.

Set the **Association Property** to a different property of the data class to enable successful deletion of the desired property.

To correct this error

1. Select the association line on the **O/R Designer** that connects the data classes indicated in the error message.
2. Double-click the line to open the **Association Editor** dialog box.
3. Remove the property from the **Association Properties**.
4. Try to delete the property again.

See also

- [LINQ to SQL tools in Visual Studio](#)

The property <property name> cannot be deleted

8/5/2021 • 2 minutes to read • [Edit Online](#)

The property <property name> cannot be deleted because it is set as the **Discriminator Property** for the inheritance between <class name> and <class name>

The selected property is set as the **Discriminator Property** for the inheritance between the classes indicated in the error message. Properties cannot be deleted if they are participating in the inheritance configuration between data classes.

Set the **Discriminator Property** to a different property of the data class to enable successful deletion of the desired property.

To correct this error

1. In the **O/R Designer**, select the inheritance line that connects the data classes indicated in the error message.
2. Set the **Discriminator Property** to a different property.
3. Try to delete the property again.

See also

- [LINQ to SQL tools in Visual Studio](#)

The connection property in the Application Settings file is missing or incorrect

8/5/2021 • 2 minutes to read • [Edit Online](#)

The connection property in the Application Settings file is missing or incorrect. The connection string from the *.dbm/* file has been used in its place.

The *.dbm/* file contains a reference to a connection string in the application settings file that cannot be found. This message is informational; the connection string setting will be created when **OK** is clicked.

To respond to this message, select **OK**. The connection information that is contained in the *.dbm/* file is added to application settings.

See also

- [LINQ to SQL tools in Visual Studio](#)

Cannot create an association <association name> - property types do not match

8/5/2021 • 2 minutes to read • [Edit Online](#)

Cannot create an association <association name> - property types do not match. Properties do not have matching types: <property names>.

Associations are defined by the selected **Association Properties** in the **Association Editor** dialog box. Properties on each side of the association must be of the same data type.

The properties listed in the message do not have the same data types.

To correct this error

1. Examine the message and note the properties called out in the message.
2. Click **OK** to dismiss the dialog box.
3. Inspect the **Association Properties** and select properties of the same data type.
4. Click **OK**.

See also

- [LINQ to SQL tools in Visual Studio](#)
- [How to: Create an association between LINQ to SQL classes \(O/R Designer\)](#)

Warning: Changes have been made to the Configure Behavior dialog box that have not been applied

8/5/2021 • 2 minutes to read • [Edit Online](#)

Warning. Changes have been made to the Configure Behavior dialog box that have not been applied. Do you want to apply your changes?

The **Configure Behavior** dialog box enables you to configure , , and behavior for all classes available. This message appears when you select a new **Class** and **Behavior** combination and the previous change has not yet been applied.

Change options

- To apply the change and continue, click **Yes**. The change is applied to the selected **Class** and **Behavior**.
- To cancel the previous change and continue, click **No**.

See also

- [LINQ to SQL tools in Visual Studio](#)

You have selected a database object from an unsupported database provider

8/5/2021 • 2 minutes to read • [Edit Online](#)

The **O/R Designer** supports only the .NET Framework Data Provider for SQL Server ([System.Data.SqlClient](#)). Although you can click **OK** and continue to work with objects from unsupported database providers, you may experience unexpected behavior at run time.

NOTE

Only data connections that use the .NET Framework Data Provider for SQL Server are supported.

Options

- Click **OK** to continue designing the entity classes that map to the connection that uses the unsupported database provider. You might experience unexpected behavior when you use unsupported database providers.
- Click **Cancel** to stop the action. Create or use a different data connection that uses the .NET Framework Provider for SQL Server.

See also

- [LINQ to SQL tools in Visual Studio](#)

This related method is the backing method for the following default insert, update, or delete methods

8/5/2021 • 2 minutes to read • [Edit Online](#)

This related method is the backing method for the following default `Insert`, `Update`, or `Delete` methods. If it is deleted, these methods will be deleted as well. Do you wish to continue?

The selected `DataContext` method is currently used as one of the `Insert`, `Update`, or `Delete` methods for one of the entity classes on the **O/R Designer**. Deleting the selected method causes the entity class that was using this method to revert to the default run-time behavior for performing the insert, update, or delete during an update.

Selected method options

- To delete the selected method, causing the entity class to use runtime updates, click **Yes**.

The selected method is deleted and any classes that used this method for overriding update behavior are reverted to using the default LINQ to SQL run-time behavior.

- To close the message box, leaving the selected method unchanged, click **No**.

The message box closes and no changes are made.

See also

- [LINQ to SQL tools in Visual Studio](#)

One or more selected items contain a data type that is not supported by the designer

8/5/2021 • 2 minutes to read • [Edit Online](#)

One or more of the items dragged from **Server Explorer** or **Database Explorer** onto the **O/R Designer** contains a data type that is not supported by the **O/R Designer**, for example, [CLR user-defined types](#).

To correct this error

1. Create a view that is based on the desired table and that does not include the unsupported data type.
2. Drag the view from **Server Explorer** or **Database Explorer** onto the designer.

See also

- [LINQ to SQL tools in Visual Studio](#)

Changing the return type of a DataContext method cannot be undone

8/5/2021 • 2 minutes to read • [Edit Online](#)

Changing the return type of a DataContext method cannot be undone. To revert back to the automatically generated type, you must drag the item from **Server Explorer** or **Database Explorer** onto the O/R Designer again. Are you sure you want to change the return type?

The return type of a [DataContext](#) method differs depending on where you drop the item in the O/R Designer. If you drop an item directly onto an existing entity class, a [DataContext](#) method that has the return type of the entity class is created. If you drop an item onto an empty area of the O/R Designer, a [DataContext](#) method that returns an automatically generated type is created. You can change the return type of a [DataContext](#) method after you add it to the methods pane. To inspect or change the return type of a [DataContext](#) method, select it and click the **Return Type** property in the **Properties** window.

To change the return type of a DataContext

- Click **Yes**.

To exit the message box and leave the return type unchanged

- Click **No**.

To revert to the original return type after changing the return type

1. Select the [DataContext](#) method on the O/R Designer and delete it.
2. Locate the item in **Server Explorer/Database Explorer** and drag it onto the O/R Designer.

A [DataContext](#) method is created with the original default return type.

See also

- [LINQ to SQL tools in Visual Studio](#)

The designer cannot be modified while debugging

8/5/2021 • 2 minutes to read • [Edit Online](#)

This message appears when an attempt is made to modify items on the **O/R Designer** when the application is running in debug mode. When the application is running in debug mode, the **O/R Designer** is read-only.

To correct this error, select **Stop Debugging** on the **Debug** menu. The application stops debugging, and you can modify items in the **O/R Designer**.

See also

- [LINQ to SQL tools in Visual Studio](#)

The selected connection uses an unsupported database provider

8/5/2021 • 2 minutes to read • [Edit Online](#)

This message appears when you drag items that do not use the .NET Framework Data Provider for SQL Server from **Server Explorer** or **Database Explorer** onto the [LINQ to SQL tools in Visual Studio](#).

The **O/R Designer** supports only data connections that use the .NET Framework Provider for SQL Server. Only connections to Microsoft SQL Server or Microsoft SQL Server Database File are valid.

To correct this error, add only items from data connections that use the .NET Framework Data Provider for SQL Server to the **O/R Designer**.

See also

- [System.Data.SqlClient](#)
- [LINQ to SQL tools in Visual Studio](#)

The objects you are adding to the designer use a different data connection than the designer

8/5/2021 • 2 minutes to read • [Edit Online](#)

The objects you are adding to the designer use a different data connection than the designer is currently using. Do you want to replace the connection used by the designer?

When you add items to the **Object Relational Designer (O/R Designer)**, all items use one shared data connection. (The design surface represents the [DataContext](#), which uses a single connection for all objects on the surface.) If you add an object to the designer that uses a data connection that differs from the data connection currently being used by the designer, this message appears. To resolve this error, you can choose to maintain the existing connection. If you make this choice, the selected object will not be added. Alternatively, you can choose to add the object and reset the [DataContext](#) connection to the new connection.

Connection options

- To replace the existing connection with the connection used by the selected object, click **Yes**.

The selected object is added to the **O/R Designer**, and the *DataContext.Connection* is set to the new connection.

NOTE

If you click **Yes**, all entity classes on the **O/R Designer** are mapped to the new connection.

- To continue to use the existing connection and cancel adding the selected object, click **No**.

The action is canceled. The *DataContext.Connection* remains set to the existing connection.

See also

- [LINQ to SQL tools in Visual Studio](#)

Cannot create an association <association name> - property listed twice

8/5/2021 • 2 minutes to read • [Edit Online](#)

Cannot create an association <association name>. The same property is listed more than once: <property name>.

Associations are defined by the selected **Association Properties** in the **Association Editor** dialog box. Properties can be listed only one time for each class in the association.

The property in the message appears more than one time in either the Parent or Child class's **Association Properties**.

To resolve this condition

- Examine the message and note the property specified in the message.
- Click **OK** to dismiss the message box.
- Inspect the **Association Properties** and remove the duplicate entries.
- Click **OK**.

See also

- [LINQ to SQL tools in Visual Studio](#)
- [How to: Create an association between LINQ to SQL classes \(O/R Designer\)](#)

Could not retrieve schema information for database object <object name>

8/5/2021 • 2 minutes to read • [Edit Online](#)

This message typically appears when an object in **Server Explorer** or **Database Explorer** is copied to the clipboard, deleted from the database, and then pasted onto the designer. Because the database object no longer exists, this message appears.

See also

- [LINQ to SQL tools in Visual Studio](#)

One or more selected database objects return a schema that does not match the schema of the target class

8/5/2021 • 2 minutes to read • [Edit Online](#)

One or more selected database objects return a schema that does not match the schema of the target class. Nothing has been added to the designer.

When you drag database objects onto existing entity classes, the data returned by the database object must match the schema of the target entity class. Verify that the correct database object is selected and that the correct entity class is being targeted.

To correct this error

1. Click **OK** to dismiss the dialog box.
2. Select a database object that returns data that matches the schema of the target class (the class the database object is being dropped onto in the **O/R Designer**).

See also

- [LINQ to SQL tools in Visual Studio](#)