.NET Framework 4

**Updating Data Sources with DataAdapters (ADO.NET)**

The **Update** method of the [DataAdapter](http://msdn.microsoft.com/en-us/library/system.data.common.dataadapter.aspx) is called to resolve changes from a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) back to the data source. The **Update** method, like the **Fill** method, takes as arguments an instance of a **DataSet**, and an optional [DataTable](http://msdn.microsoft.com/en-us/library/system.data.datatable.aspx) object or **DataTable** name. The **DataSet** instance is the **DataSet** that contains the changes that have been made, and the **DataTable** identifies the table from which to retrieve the changes. If no **DataTable** is specified, the first **DataTable** in the **DataSet** is used.

When you call the **Update** method, the **DataAdapter** analyzes the changes that have been made and executes the appropriate command (INSERT, UPDATE, or DELETE). When the **DataAdapter** encounters a change to a [DataRow](http://msdn.microsoft.com/en-us/library/system.data.datarow.aspx), it uses the [InsertCommand](http://msdn.microsoft.com/en-us/library/system.data.common.dbdataadapter.insertcommand.aspx), [UpdateCommand](http://msdn.microsoft.com/en-us/library/system.data.common.dbdataadapter.updatecommand.aspx), or [DeleteCommand](http://msdn.microsoft.com/en-us/library/system.data.common.dbdataadapter.deletecommand.aspx) to process the change. This allows you to maximize the performance of your ADO.NET application by specifying command syntax at design time and, where possible, through the use of stored procedures. You must explicitly set the commands before calling **Update**. If **Update** is called and the appropriate command does not exist for a particular update (for example, no **DeleteCommand** for deleted rows), an exception is thrown.

|  |
| --- |
| **NoteNote** |
| If you are using SQL Server stored procedures to edit or delete data using a **DataAdapter**, make sure that you do not use SET NOCOUNT ON in the stored procedure definition. This causes the rows affected count returned to be zero, which the **DataAdapter** interprets as a concurrency conflict. In this event, a [DBConcurrencyException](http://msdn.microsoft.com/en-us/library/system.data.dbconcurrencyexception.aspx) will be thrown. |

Command parameters can be used to specify input and output values for an SQL statement or stored procedure for each modified row in a **DataSet**. For more information, see [DataAdapter Parameters (ADO.NET)](http://msdn.microsoft.com/en-us/library/bbw6zyha.aspx).

|  |
| --- |
| **NoteNote** |
| It is important to understand the difference between deleting a row in a [DataTable](http://msdn.microsoft.com/en-us/library/system.data.datatable.aspx) and removing the row. When you call the **Remove** or **RemoveAt** method, the row is removed immediately. Any corresponding rows in the back end data source will not be affected if you then pass the **DataTable** or **DataSet** to a **DataAdapter** and call **Update**. When you use the **Delete** method, the row remains in the **DataTable** and is marked for deletion. If you then pass the **DataTable** or **DataSet** to a **DataAdapter** and call **Update**, the corresponding row in the back end data source is deleted. |

If your **DataTable** maps to or is generated from a single database table, you can take advantage of the [DbCommandBuilder](http://msdn.microsoft.com/en-us/library/system.data.common.dbcommandbuilder.aspx) object to automatically generate the **DeleteCommand**, **InsertCommand**, and **UpdateCommand** objects for the **DataAdapter**. For more information, see [Generating Commands with CommandBuilders (ADO.NET)](http://msdn.microsoft.com/en-us/library/tf579hcz.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Using UpdatedRowSource to Map Values to a DataSet

You can control how the values returned from the data source are mapped back to the **DataTable** following a call to the Update method of a **DataAdapter**, by using the [UpdatedRowSource](http://msdn.microsoft.com/en-us/library/system.data.common.dbcommand.updatedrowsource.aspx) property of a [DbCommand](http://msdn.microsoft.com/en-us/library/system.data.common.dbcommand.aspx) object. By setting the **UpdatedRowSource** property to one of the [UpdateRowSource](http://msdn.microsoft.com/en-us/library/system.data.updaterowsource.aspx) enumeration values, you can control whether output parameters returned by the **DataAdapter** commands are ignored or applied to the changed row in the **DataSet**. You can also specify whether the first returned row (if it exists) is applied to the changed row in the **DataTable**.

The following table describes the different values of the **UpdateRowSource** enumeration and how they affect the behavior of a command used with a **DataAdapter**.

|  |  |
| --- | --- |
| **UpdatedRowSource Enumeration** | **Description** |
| Both | Both the output parameters and the first row of a returned result set may be mapped to the changed row in the **DataSet**. |
| FirstReturnedRecord | Only the data in the first row of a returned result set may be mapped to the changed row in the **DataSet**. |
| None | Any output parameters or rows of a returned result set are ignored. |
| OutputParameters | Only output parameters may be mapped to the changed row in the **DataSet**. |

The **Update** method resolves your changes back to the data source; however other clients may have modified data at the data source since the last time you filled the **DataSet**. To refresh your **DataSet** with current data, use the **DataAdapter** and **Fill** method. New rows will be added to the table, and updated information will be incorporated into existing rows. The **Fill** method determines whether a new row will be added or an existing row will be updated by examining the primary key values of the rows in the **DataSet** and the rows returned by the **SelectCommand**. If the **Fill** method encounters a primary key value for a row in the **DataSet** that matches a primary key value from a row in the results returned by the **SelectCommand**, it updates the existing row with the information from the row returned by the **SelectCommand** and sets the [RowState](http://msdn.microsoft.com/en-us/library/system.data.datarow.rowstate.aspx) of the existing row to **Unchanged**. If a row returned by the **SelectCommand** has a primary key value that does not match any of the primary key values of the rows in the **DataSet**, the **Fill** method adds a new row with a **RowState** of **Unchanged**.

|  |
| --- |
| **NoteNote** |
| If the **SelectCommand** returns the results of an OUTER JOIN, the **DataAdapter** will not set a **PrimaryKey** value for the resulting **DataTable**. You must define the **PrimaryKey** yourself to ensure that duplicate rows are resolved correctly. For more information, see [Defining Primary Keys (ADO.NET)](http://msdn.microsoft.com/en-us/library/z24kefs8.aspx). |

To handle exceptions that may occur when calling the **Update**method, you can use the **RowUpdated** event to respond to row update errors as they occur (see [Handling DataAdapter Events (ADO.NET)](http://msdn.microsoft.com/en-us/library/6d1wk41s.aspx)), or you can set **DataAdapter.ContinueUpdateOnError** to **true** before calling **Update**, and respond to the error information stored in the **RowError** property of a particular row when the update is complete (see [Row Error Information](http://msdn.microsoft.com/en-us/library/k3877412.aspx)).

**Note**   Calling **AcceptChanges** on the **DataSet**, **DataTable**, or **DataRow** will cause all **Original** values for a **DataRow** to be overwritten with the **Current** values for the **DataRow**. If the field values that identify the row as unique have been modified, after calling **AcceptChanges** the **Original** values will no longer match the values in the data source. **AcceptChanges** is called automatically for each row during a call to the Update method of a **DataAdapter**. You can preserve the original values during a call to the Update method by first setting the **AcceptChangesDuringUpdate** property of the **DataAdapter** to false, or by creating an event handler for the **RowUpdated** event and setting the [Status](http://msdn.microsoft.com/en-us/library/system.data.common.rowupdatedeventargs.status.aspx) to SkipCurrentRow. For more information, see [Merging DataSet Contents (ADO.NET)](http://msdn.microsoft.com/en-us/library/aszytsd8.aspx) and [Handling DataAdapter Events (ADO.NET)](http://msdn.microsoft.com/en-us/library/6d1wk41s.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Example

The following examples demonstrate how to perform updates to modified rows by explicitly setting the **UpdateCommand** of a **DataAdapter** and calling its **Update** method. Notice that the parameter specified in the WHERE clause of the UPDATE statement is set to use the **Original** value of the **SourceColumn**. This is important, because the **Current** value may have been modified and may not match the value in the data source. The **Original** value is the value that was used to populate the **DataTable** from the data source.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl61_ctl00_ctl00_code');" \o "Copy Code)

Private Sub AdapterUpdate(ByVal connectionString As String)

Using connection As SqlConnection = New SqlConnection( \_

connectionString)

Dim adapter As SqlDataAdapter = New SqlDataAdapter( \_

"SELECT CategoryID, CategoryName FROM dbo.Categories", \_

connection)

adapter.UpdateCommand = New SqlCommand( \_

"UPDATE Categories SET CategoryName = @CategoryName " & \_

"WHERE CategoryID = @CategoryID", connection)

adapter.UpdateCommand.Parameters.Add( \_

"@CategoryName", SqlDbType.NVarChar, 15, "CategoryName")

Dim parameter As SqlParameter = \_

adapter.UpdateCommand.Parameters.Add( \_

"@CategoryID", SqlDbType.Int)

parameter.SourceColumn = "CategoryID"

parameter.SourceVersion = DataRowVersion.Original

Dim categoryTable As New DataTable

adapter.Fill(categoryTable)

Dim categoryRow As DataRow = categoryTable.Rows(0)

categoryRow("CategoryName") = "New Beverages"

adapter.Update(categoryTable)

Console.WriteLine("Rows after update.")

Dim row As DataRow

For Each row In categoryTable.Rows

Console.WriteLine("{0}: {1}", row(0), row(1))

Next

End Using

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl61_ctl00_ctl01_code');" \o "Copy Code)

private static void AdapterUpdate(string connectionString)

{

using (SqlConnection connection =

new SqlConnection(connectionString))

{

SqlDataAdapter dataAdpater = new SqlDataAdapter(

"SELECT CategoryID, CategoryName FROM Categories",

connection);

dataAdpater.UpdateCommand = new SqlCommand(

"UPDATE Categories SET CategoryName = @CategoryName " +

"WHERE CategoryID = @CategoryID", connection);

dataAdpater.UpdateCommand.Parameters.Add(

"@CategoryName", SqlDbType.NVarChar, 15, "CategoryName");

SqlParameter parameter = dataAdpater.UpdateCommand.Parameters.Add(

"@CategoryID", SqlDbType.Int);

parameter.SourceColumn = "CategoryID";

parameter.SourceVersion = DataRowVersion.Original;

DataTable categoryTable = new DataTable();

dataAdpater.Fill(categoryTable);

DataRow categoryRow = categoryTable.Rows[0];

categoryRow["CategoryName"] = "New Beverages";

dataAdpater.Update(categoryTable);

Console.WriteLine("Rows after update.");

foreach (DataRow row in categoryTable.Rows)

{

{

Console.WriteLine("{0}: {1}", row[0], row[1]);

}

}

}

}

http://i.msdn.microsoft.com/Global/Images/clear.gif AutoIncrement Columns

If the tables from your data source have auto-incrementing columns, you can fill the columns in your **DataSet** either by returning the auto-increment value as an output parameter of a stored procedure and mapping that to a column in a table, by returning the auto-increment value in the first row of a result set returned by a stored procedure or SQL statement, or by using the **RowUpdated** event of the **DataAdapter** to execute an additional SELECT statement. For more information and an example, see [Retrieving Identity or Autonumber Values (ADO.NET)](http://msdn.microsoft.com/en-us/library/ks9f57t0.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Ordering of Inserts, Updates, and Deletes

In many circumstances, the order in which changes made through the **DataSet** are sent to the data source is important. For example, if a primary key value for an existing row is updated, and a new row has been added with the new primary key value as a foreign key, it is important to process the update before the insert.

You can use the **Select** method of the **DataTable** to return a **DataRow** array that only references rows with a particular **RowState**. You can then pass the returned **DataRow** array to the **Update** method of the **DataAdapter** to process the modified rows. By specifying a subset of rows to be updated, you can control the order in which inserts, updates, and deletes are processed.

http://i.msdn.microsoft.com/Global/Images/clear.gif Example

For example, the following code ensures that the deleted rows of the table are processed first, then the updated rows, and then the inserted rows.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl64_ctl00_ctl00_code');" \o "Copy Code)

Dim table As DataTable = dataSet.Tables("Customers")

' First process deletes.

dataSet.Update(table.Select(Nothing, Nothing, \_

DataViewRowState.Deleted))

' Next process updates.

adapter.Update(table.Select(Nothing, Nothing, \_

DataViewRowState.ModifiedCurrent))

' Finally, process inserts.

dataAdpater.Update(table.Select(Nothing, Nothing, \_

DataViewRowState.Added))

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl64_ctl00_ctl01_code');" \o "Copy Code)

DataTable table = dataSet.Tables["Customers"];

// First process deletes.

adapter.Update(table.Select(null, null, DataViewRowState.Deleted));

// Next process updates.

adapter.Update(table.Select(null, null,

DataViewRowState.ModifiedCurrent));

// Finally, process inserts.

adapter.Update(table.Select(null, null, DataViewRowState.Added));