PDF-XChange
Click to buy NOW!
www.docu-track.com

```
//customer who has orders
var delRow = customersTable.Select("CustomerID='PARIS'")[0];
delRow.Delete();

//send changes to database
da.Update(nwSet, "Customers");
dataGridView2.DataSource = nwSet;
dataGridView2.DataMember = "Customers";
MessageBox.Show("Update Complete");
```

> ***NOTE*** **DON'T EXECUTE TWICE!**
>
> **If you execute this code twice, an exception will be thrown indicating that you are trying to insert duplicate rows. If you stop running the application and then run it again, you will be able to execute again because the Northwind database has the *Copy To Output Directory* property set to *Copy Always*, which means that you get a new clean copy of the database every time you build the project.**

Saving Changes to the Database in Batches

If you have the SQL Profiler tool that is available when you install the full version of SQL Server on your computer, you can use this tool to view the update commands sent to SQL Server. You will notice that individual insert, update, and delete commands are sent to SQL Server on a row-by-row basis. One way to increase update performance is to send the changes to the database server in batches by assigning a value to the *DbDataAdapter* object's *UpdateBatchSize* property. This property defaults to *1*, which causes each change to be sent to the server on a row-by-row basis. Setting the value to *0* instructs the *DbDataAdapter* object to create the largest possible batch size for changes, or you can set the value to the number of changes you want to send to the server in each batch. Setting *UpdateBatchSize* to a number greater than the number of changes that need to be sent is equivalent to setting it to *0*.

> ***NOTE*** **FREE PERFORMANCE UPGRADE!**
>
> **Setting *UpdateBatchSize* to *0* is a quick way to boost the update performance of the *DbDataAdapter* object.**

One way to confirm that the changes are being sent to the database server in batches is to register a handler to the *RowUpdated* event of the *DbDataAdapter*-derived class instances. The event handler method receives the number of rows affected in the last batch. When *UpdateBatchSize* is set to *1*, the *RecordsAffected* property is always *1*. In the following code sample, the Customers table contains 91 rows. *nwDataSet* is filled, and then the CustomerName field is modified on all 91 rows. Before the *Update* method is executed, *UpdateBatchSize* is changed to *40*. When the *Update* method is executed, the changes are sent to the database as a batch of 40 changes, another batch of 40 changes, and, finally, a

batch of 11 changes. This code contains a *RowUpdated* event handler to collect batch information, which is displayed after the *Update* method is executed. In this code, *SqlDataAdapter* is used because this event does not exist on the *DbDataAdapter* base class.

Sample of Visual Basic Code

```
Private WithEvents da As New SqlDataAdapter()
Private sb as New System.Text.StringBuilder()

private sub rowUpdated(byval sender as Object, _
      byval e as SqlRowUpdatedEventArgs) handles da.RowUpdated
   sb.Append("Rows: " & e.RecordsAffected.ToString() & vbCrLf)
End Sub

Private Sub menuUpdateBatch_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles UpdateBatchToolStripMenuItem.Click
    Dim nw = ConfigurationManager.ConnectionStrings("nw")
    Dim connection = New SqlConnection()
    connection.ConnectionString = nw.ConnectionString
    Dim cmd = connection.CreateCommand()
    cmd.CommandType = CommandType.Text
    cmd.CommandText = "SELECT * FROM Customers"
    Dim nwSet As New DataSet("nw")
    da.SelectCommand = cmd
    Dim bldr As New SqlCommandBuilder(da)
    da.Fill(nwSet, "Customers")

    'Modify data here
    For Each dr As DataRow In nwSet.Tables("Customers").Rows
        dr("CompanyName") = dr("CompanyName").ToString().ToUpper()
    Next
    sb.Clear()
    da.UpdateBatchSize = 40
    da.Update(nwSet, "Customers")
    MessageBox.Show(sb.ToString())
End Sub
```

Sample of C# Code

```
private SqlDataAdapter da = new SqlDataAdapter();
private System.Text.StringBuilder sb = new System.Text.StringBuilder();

private void rowUpdated(object sender, SqlRowUpdatedEventArgs e )
{
   sb.Append("Rows: " + e.RecordsAffected.ToString() + "\r\n");
}

private void menuUpdateBatch_Click(object sender, EventArgs e)
{
    //event subscription is normally placed in constructor but is here
    //to encapsulate the sample
    da.RowUpdated += rowUpdated;
    var nw = ConfigurationManager.ConnectionStrings["nw"];
    var connection = new SqlConnection();
    connection.ConnectionString = nw.ConnectionString;
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
    var cmd = connection.CreateCommand();
    cmd.CommandType = CommandType.Text;
    cmd.CommandText = "SELECT * FROM Customers";
    da.SelectCommand = cmd;
    var nwSet = new DataSet("nw");
    var bldr = new SqlCommandBuilder(da);
    da.Fill(nwSet, "Customers");

    //Modify data here
    foreach (DataRow dr in nwSet.Tables["Customers"].Rows)
    {
        dr["CompanyName"] = dr["CompanyName"].ToString().ToUpper();
    }
    sb.Clear();
    da.UpdateBatchSize = 40;
    da.Update(nwSet, "Customers");

    //if event subscription is in the contructor, no need to
    //remove it here....
    da.RowUpdated -= rowUpdated;
    MessageBox.Show(sb.ToString());
}
```

DbProviderFactory Classes

There are many reasons for writing an application that does not require database provider–specific code. A company might want the flexibility to upgrade from one database product to another, such as for moving from Microsoft Access to SQL Server. Or a company might have a retail application that must allow connectivity to any data source. With earlier versions of ADO.NET, you can write a provider-independent application by using generic interfaces. The typical coding might look something like the following:

Sample of Visual Basic Code

```
Public Enum DbProvider
   SqlClient
   OleDb
   Odbc
End Enum

Public Function GetConnection() As IDbConnection
   'Get the provider from the config file
   Dim provider As DbProvider = [Enum].Parse( _
   GetType(DbProvider), _
      ConfigurationSettings.AppSettings("provider").ToString())

   Dim connection As IDbConnection = Nothing
   Select Case (provider)
      Case DbProvider.SqlClient
         connection = New System.Data.SqlClient.SqlConnection()
      Case DbProvider.OleDb
         connection = New System.Data.OleDb.OleDbConnection()
      Case DbProvider.Odbc
```