Title	Use a DataGrid in VB.NET
Keywords	VB.NET, NET, DataGrid, ADO.NET
Categories	Database, VB.NET

If you use the VB.NET Data Form Wizard to display data in a grid, you get an amazingly complicated and confusing result. Perhaps even more amazing is the fact that the form doesn't do everything. The user still needs to push a button to load data and there is no code at all to save changes.

After you dig into the objects for a while, however, you discover that this is actually a very simple problem. The program only uses three interesting types of objects: SqlDataAdapter, DataSet, and SqlCommandBuilder.

In the form's Load event handler, create a new SqlDataAdapter using the SQL SELECT statement that fetches the data and a connection string for connecting to the database. The details of this string will vary depending on your database.

Optionally add a TableMapping to map the (stupid) default table name Table returned by the connection to a more meaningful name (like Contacts for the Contacts table).

Create a new DataSet object to hold the data. Use the data adapter's Fill method to pull data from the database into the DataSet.

Now bind the DataGrid control to the DataSet. This makes the control automatically display all of the data in rows and columns. The user can add, edit, and delete records using the DataGrid with no more work from you. You can use the DataGrid's properties if you want to restrict access. For example, you can make the DataGrid disallow editing.

```
Private Const SELECT_STRING As String = _
    "SELECT * FROM Contacts ORDER BY LastName, FirstName"
Private Const CONNECT_STRING As String = _
    "Data Source=Bender\NETSDK;Initial " & _
        "Catalog=Contacts;User Id=sa"

' The DataSet that holds the data.
Private m_DataSet As DataSet

' Load the data.
Private Sub Form1_Load(ByVal sender As Object, ByVal e As _
    System.EventArgs) Handles MyBase.Load
    Dim data_adapter As SqlDataAdapter

    ' Create the SqlDataAdapter.
    data_adapter = New SqlDataAdapter(SELECT_STRING, _
        CONNECT_STRING)

    ' Map Table to Contacts.
    data_adapter.TableMappings.Add("Table", "Contacts")

    ' Fill the DataSet.
    m_DataSet = New DataSet()
    data_adapter.Fill(m_DataSet)

    ' Bind the DataGrid control to the Contacts DataTable.
    dgContacts.SetDataBinding(m_DataSet, "Contacts")
End Sub
```

When the user modifies the data, the changes are only saved to the DataSet, not to the database. To save the changes, give the form a Closing event handler. In this routine, make a SqlDataAdapter as before. Add a TableMapping if you did in the Load event handler.

Because you have only defined the data adapter's SQL SELECT statement, you still need to define the INSERT, UPDATE, and DELETE statements it will use to update the data. The easy way to do that is to create a SqlCommandBuilder attached to the data adapter. It automatically creates those statements from the SELECT statement. Uncomment the Debug.WriteLine statements in the following code to see the statements the SqlCommandBuilder creates.

Now use the SqlDataAdapter's Update method to update the database.

```
' Save any changes to the data.
Private Sub Form1_Closing(ByVal sender As Object, ByVal e _
    As System.ComponentModel.CancelEventArgs) Handles _
    MyBase.Closing
    If m_DataSet.HasChanges() Then
        Dim data_adapter As SqlDataAdapter
        Dim command_builder As SqlCommandBuilder

        ' Create the DataAdapter.
        data_adapter = New SqlDataAdapter(SELECT_STRING, _
            CONNECT_STRING)
```

```
        ' Map Table to Contacts.
        data_adapter.TableMappings.Add("Table", "Contacts")

        ' Make the CommandBuilder generate the
        ' insert, update, and delete commands.
        command_builder = New _
            SqlCommandBuilder(data_adapter)

        ' Uncomment this code to see the INSERT,
        ' UPDATE, and DELETE commands.
        'Debug.WriteLine("*** INSERT ***")
        'Debug.WriteLine(command_builder.GetInsertCommand.CommandText)
        'Debug.WriteLine("*** UPDATE ***")
        'Debug.WriteLine(command_builder.GetUpdateCommand.CommandText)
        'Debug.WriteLine("*** DELETE ***")
        'Debug.WriteLine(command_builder.GetDeleteCommand.CommandText)

        ' Save the changes.
        data_adapter.Update(m_DataSet)
    End If
End Sub
```

There are a few simple enhancements you can make. For example, you can check the DataSet's HasChanges method to see if there are any changes before you go to the trouble of creating a new SqlDataAdapter and SqlCommandBuilder. It may also be more efficient to save changes grouped by type: all inserts, all updates, all deletes. The basic program, however, is trivial and is perfectly adequate if the user won't make a huge number of changes.

For more information on using databases in Visual Basic .NET, see my book Visual Basic .NET Database Programming.

Note that this example uses an MSDE database. Click here for information on MSDE including instructions for installing it for free.

You will also need build a database for the program to use and change the connect code shown here to match. This example uses an MSDE server named NETSDK on the computer Bender and takes data from the Contacts database.

Download