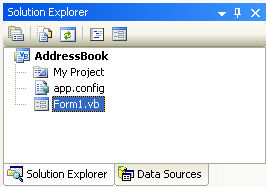
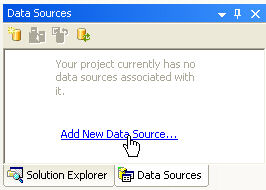
The first part of the tutorial is here: [Database project for Visual Basic .NET Express users](http://www.homeandlearn.co.uk/net/nets12p1.html)

Let's make a start on our Database project. So, once you have your VB software open, do the following:

* Click **File > New Project** from the menu bar
* Select **Windows Application**, and then give it the **Name** AddressBook. Click OK
* Locate the **Solution Explorer**on the right hand side (If you can't see it, click **View > Solution Explorer** from the menu bar in versions 2008 and 2012, or **View > Other Windows > Solution Explorer** in version 2010.)



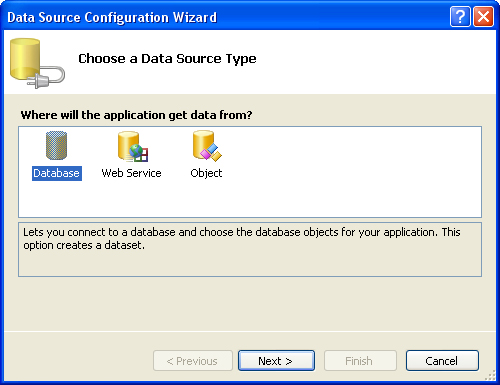
* We need to select a Data Source. So click on **Data Sources** at the bottom of the Solution Explorer in version 2008:



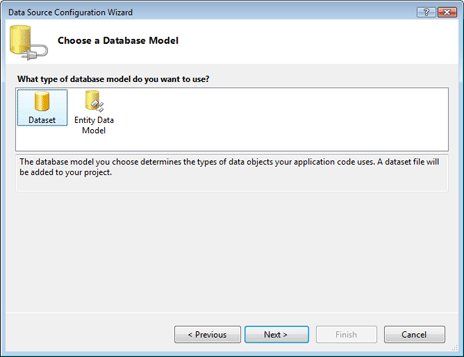
If you have VB NET 2010 or 2012 then the Data Source tab is on the left, just below the Toolbox (If you can't see the tab, click **View > Other Windows > Data Sources**):

Data Source tab in  VB Net 2010

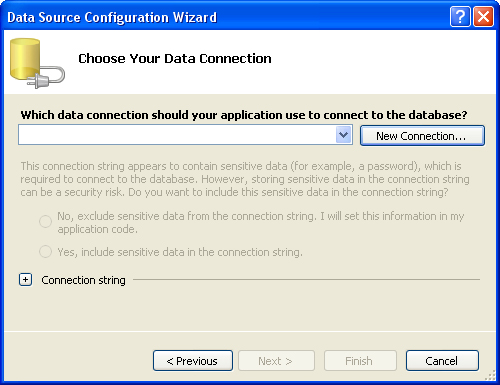
To Add a New Data Source, click on the link. When you do, you'll see a screen welcoming you to the Data Source Configuration Wizard, Just click Next, to get to the screen below:



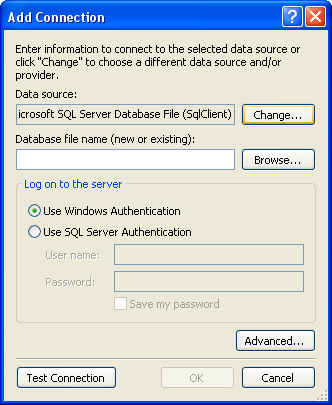
You want to connect to a Database. So select this option, and click Next. In version 2010 and 2012 of VB NET, you'll see this screen appear (you won't see it if you have version 2008):



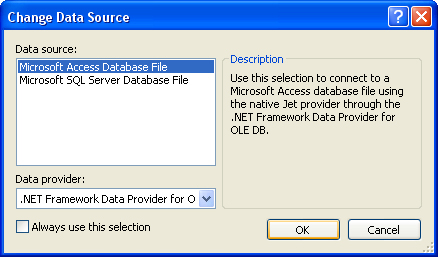
Select DataSet and click Next. You'll then see this screen:



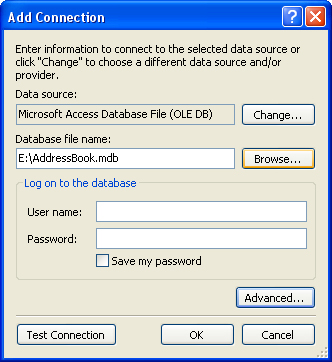
Click the **New Connection** button and another dialogue box pops up in VB NET 2008:



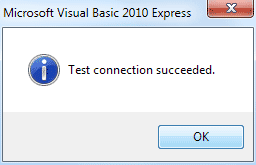
Click the **Change** button, because we want to connect to an Access database. (The default is for a SQL Server database.) When you click **Change**, you'll see this (VB NET versions 2010 and 2012 will see this instead of the dialogu box above when clicking the **New Connection** button):



Select **Microsoft Access Database File**, then click OK. The previous dialogue box will then look like this:



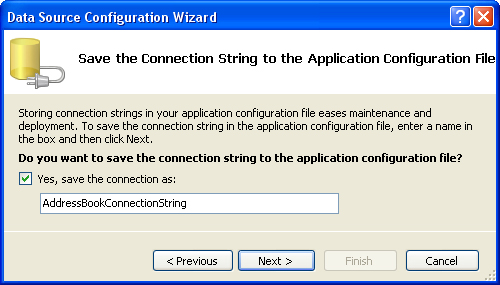
Click the **Browse** button and navigate to where on your computer you downloaded our Access Database called AddressBook.mdb. Click **Test Connection** to see if everything is OK, and you'll hopefully see this:



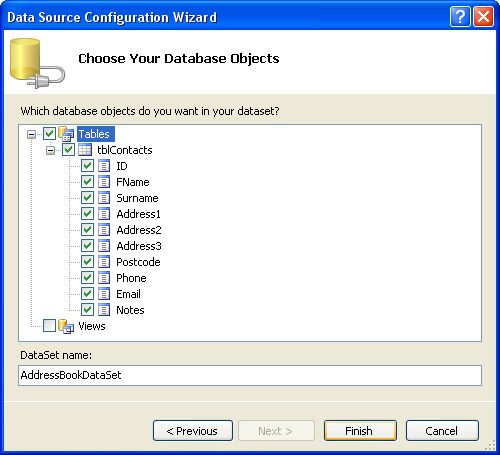
Click the OK button, then click the OK button on the Add Connection dialogue box as well. You will be returned to the Data Source Configuration Wizard, which should now look like this:



Click Next to move to the next step of the Wizard. You may see a message box appear, however. Click No on the message box to stop VB copying the database each time it runs. You should then see this:

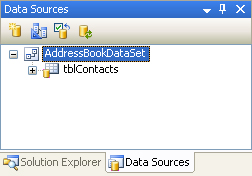


Make sure there's a tick in the box for "Save the connection", and then click Next:

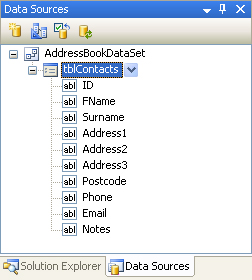


Here, you can select which tables and fields you want. Tick the **Tables** box to include them all. You can give your DataSet a name, if you prefer. Click Finish and you're done.

When you are returned to your form, you should notice your new Data Source has been added:

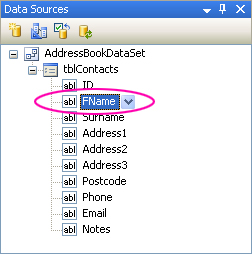


The Data Sources area of the Solution Explorer (or Data Sources tab on the left) now displays information about your database. Click the plus symbol (arrow symbol in version 2012) next to**tblContacts**:



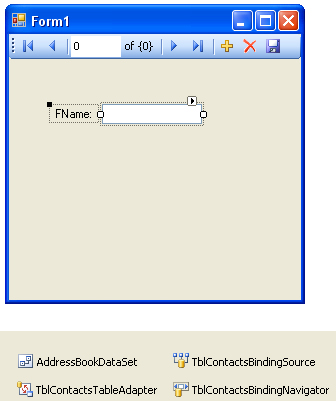
All the Fields in the Address Book database are now showing.

To add a Field to your Form, click on one in the list. Hold down your left mouse button, and drag it over to your form:

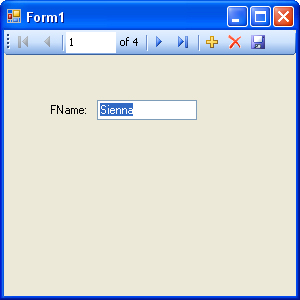


In the image above, the **FName** field is being dragged on the Form. Your mouse cursor will change shape.

When your Field is over the Form, let go of your left mouse button. A textbox and a label will be added. There are two other things to notice: a navigation bar appears at the top of the form, and a lot of strange objects have appeared in the object area at the bottom:

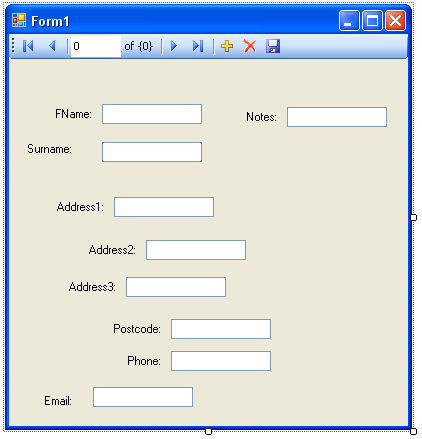


We'll explore the Objects in a later section. But notice the Navigation bar in blue. Run your programme by hitting the F5 key on your keyboard. You should see this:



Click the Navigation arrows to scroll through the database. When you've played around with the controls, stop the form from running, and return to Design View.

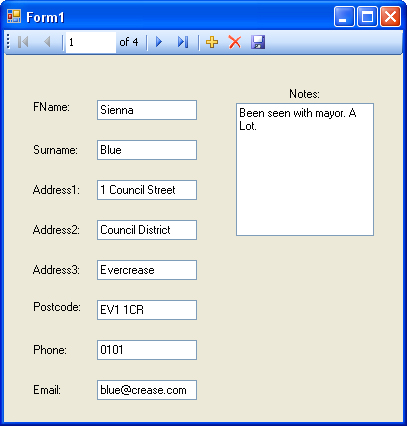
Drag and Drop more Fields to your form. But don't align them yet. We'll see an easy way to do this. But once you've dragged the fields to your form, it might look like this:



I'm sure you'll agree - that's a very untidy form. But there's a very easy way to align all your controls. Try this:

* Click on a Textbox and its label with your left mouse button
* Hold down the Ctrl key on your keyboard, and select a second Textbox and label
* With the Ctrl key still held down, click each Textbox and label in turn
* When all Textbox and labels are selected, click on the **Format** menu at the top
* From the Format menu select **Align > Lefts**. The left edges of the Textboxes will align themselves
* From the Format menu select **Vertical Spacing > Make Equa**l. The space between each textbox will then be the same

For the Notes Textbox, set the **MultiLine** property to **True** and resize the textbox. With your new controls added, and nicely aligned, press F5 to run your form. Your form might then be something like this:



Click the Navigation icons to move backwards and forwards through your database.

In the next part, you'll move away from the Wizards and learn how to add your own programming code to open up and manipulate databases.

In this next section, we'll take a look at the objects that you can use to open and read data from a Database. We'll stick with our Access database, the AddressBook.mdb one, and recreate what the Wizard has done. That way, you'll see for yourself just what is going on behind the scenes.

So close any open projects, and create a new one. Give it whatever name you like, and let's begin.

If you haven't yet downloaded the Address Book database, you can get it here:

[Download the Address Book Database](http://www.homeandlearn.co.uk/net/AddressBook.zip)

**The Connection Object**

The Connection Object is what you need if you want to connect to a database. There are a number of different connection objects, and the one you use depends largely on the type of database you're connecting to. Because we're connecting to an Access database, we'll need something called the OLE DB connection object.

OLE stands for Object Linking and Embedding, and its basically a lot of objects (COM objects) bundled together that allow you to connect to data sources in general, and not just databases. You can use it, for example, to connect to text files, SQL Server, email, and a whole lot more.

There are a number of different OLE DB objects (called data providers), but the one we'll use is called "**Jet**". Others are SQL Server and Oracle.

So place a button on your form. Change the **Name** property to **btnLoad**. Double click your button to open up the code window. Add the following line:

**Dim con As New OleDb.OleDbConnection**

The variable **con** will now hold the **Connection Objec**t. Notice that there is a full stop after the OleDB part. You'll then get a pop up box from where you can select OleDbConnection. We're also creating a**New** object on this line.This is the object that you use to connect to an Access database.

[No more reading these lessons online - get the eBook here!](http://www.homeandlearn.co.uk/bookshop/net.html)

**Setting a Connection String**

There are Properties and Methods associated with the Connection Object, of course. We want to start with the ConnectionString property. This can take MANY parameters . Fortunately, we only need a few of these.

We need to pass two things to our new **Connection Object**: the technology we want to use to do the connecting to our database; and where the database is. (If your database was password and user name protected, you would add these two parameters as well. Ours isn't, so we only need the two.)

The technology is called the **Provider**; and you use **Data Source** to specify where your database is. So add this to your code:

**Dim dbProvider As String  
Dim dbSource As String**

**dbProvider = "PROVIDER=Microsoft.Jet.OLEDB.4.0;"  
dbSource = "Data Source = C:/AddressBook.mdb"**

**con.ConnectionString = dbProvider & dbSource**

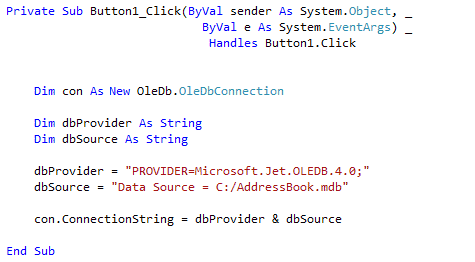
The first part specifies which provider technology we want to use to do the connecting (JET). The second part, typed after a semi-colon, points to where the database is. In the above code, the database is on the C drive, in the root folder. The name of the Access file we want to connect to is called AddressBook.mdb. (Note that "Data Source" is two words, and not one.)

If you prefer, you can have the provider and source on one line, as below (it's on two here because it won't all fit on one line):

**con.ConnectionString = "PROVIDER=Microsoft.Jet.OLEDB.4.0;Data Source = C:\AddressBook.mdb"**

The first part specifies which provider technology we want to use to do the connecting (**JET**). The second part, typed after a semi-colon, points to where the database is. In the above code, the database is on the C drive, in the root folder. The name of the Access file we want to connect to is called **AddressBook.mdb.** (Note that "**Data Source**" is two words, and not one.)

But your coding window should now look like this:



This assumes that you have copied the AddressBook database over to the root folder of your C Drive. If you've copied it to another folder, change the "Data Source" part to match. For example, if you copied it to a folder called "databases" you'd put this:

**Data Source = C:\databases\AddressBook.mdb**

You can also specify a folder such as MyDocuments (or Documents in Vista and Windows 7). You do it like this:

**dbSource = "Data Source = C:\Users\Owner\Documents\AddressBook.mdb"**

Another way to specify a file path is this:

**Dim fldr As String  
fldr = Environment.GetFolderPath( Environment.SpecialFolder.MyDocuments ) & "/AddressBook.mdb"**

**dbSource = "Data Source = " & fldr**

On the second line, spread over two lines in the code above, we have this:

**Environment.GetFolderPath()**

The folder path you're getting goes between the round brackets of GetFolderPath:

**Environment.SpecialFolder.MyDocuments**

The Special Folder in this case is the MyDocuments folder.

But back to our connection code. **ConnectionString** is a property of the **con** variable. The con variable holds our Connection Object. We're passing the Connection String the name of a data provider, and a path to the database.

**Opening the Connection**

Now that we have a ConnectionString, we can go ahead and open the datatbase. This is quite easy - just use the **Open** method of the Connection Object:

**con.Open()**

Once open, the connection has to be closed again. This time, just use the Close method:

**con.Close()**

Add the following four lines to your code:

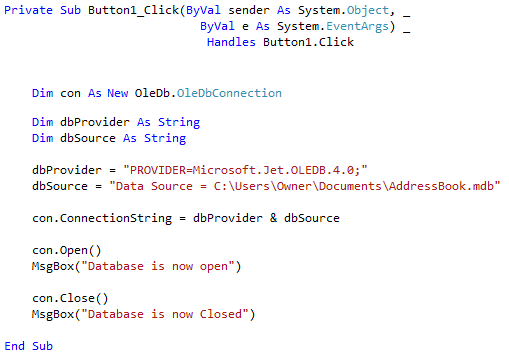
**con.Open()**

**MsgBox("Database is now open")**

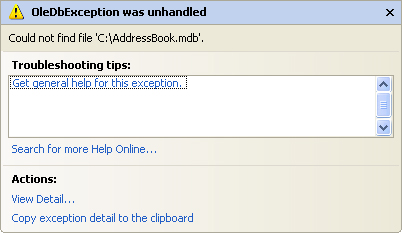
**con.Close()**

**MsgBox("Database is now Closed")**

Your coding window will then look like this (use the file path below, if you have Vista or Windows 7, after moving the database to your Documents folder):



Test out your new code by running your programme. Click your button and the two message boxes should display. If they don't, make sure your Data Source path is correct. If it isn't, you might see this error message:



The error message is a bit on the vague and mysterious side. But what it's saying is that it can't find the path to the database, so it can't Open the connection. The line con.Open in your code will then be highlighted in green. You need to specify the correct path to your database. When you do, you'll see the message boxes from our code, and not the big one above.

Now that we've opened a connection to the database, we need to read the information from it. This is where the DataSet and the DataAdapter come in.

Part of an ongoing tutorial. The first part of Databases and VB .NET can be found here:

[Coding your own VB .NET database projects](http://www.homeandlearn.co.uk/net/nets12p4.html)

In the [previous part](http://www.homeandlearn.co.uk/net/nets12p4.html), you learned how to set up a Connection Object. This was so that you could open a connection to the database itself. But that's not the end of it. The data from the database needs to be stored somewhere, so that we can manipulate it.

ADO.NET uses something called a **DataSet** to hold all of your information from the database (you can also use a DataTable, if all you want to do is read information, and not have people write to your database.). But the **DataSet** (and Data Table) will hold a copy of the information from the database.

The DataSet is not something you can draw on your form, like a Button or a Textbox. The DataSet is something that is hidden from you, and just stored in memory. Imagine a grid with rows and columns. Each imaginary row of the DataSet represents a Row of information in your Access database. And each imaginary column represents a Column of information in your Access database (called a Field in Access).

This, then, is a DataSet. But what's a Data Adapter?

The Connection Object and the DataSet can't see each other. They need a go-between so that they can communicate. This go-between is called a Data Adapter. The Data Adapter contacts your Connection Object, and then executes a query that you set up. The results of that query are then stored in the DataSet.

The Data Adapter and DataSet are objects. You set them up like this:

**Dim ds As New DataSet  
Dim da As OleDb.OleDbDataAdapter**

**da = New OleDb.OleDbDataAdapter( sql, con )**

The code needs a little explaining, though. First, the Data Adapter.

### The Data Adapter

The Data Adapter is a property of the OLEDB object, hence the full stop between the two:

**OleDb.OleDbDataAdapter**

We're passing this object to the variable called **da**. This variable will then hold a reference to the Data Adapter.

While the second line in the code above sets up a reference to the Data Adapter, the third line creates a new Data Adapter object. You need to put two things in the round brackets of the Object declaration: Your SQL string (which we'll get to shortly), and your connection object. Our Connection Object is stored in the variable which we've called **con**. (Like all variable you can call it practically anything you like. We've gone for something short and memorable.) You then pass the New Data Adapter to your variable (**da** for us):

**da = New OleDb.OleDbDataAdapter(sql, con )**

We need something else, though. The **sql** in between the round brackets is the name of a variable. We haven't yet set this up. We'll have a look at SQL in a moment. But bear in mind what the Data Adaptor is doing: ***Acting as a go-between for the Connection Object and the Data Set***

[No more reading these lessons online - get the eBook here!](http://www.homeandlearn.co.uk/bookshop/net.html)

### Structured Query Language

SQL (pronounced SeeKwel), is short for Structured Query Language, and is a way to query and write to databases (not just Access). The basics are quite easy to learn. If you want to grab all of the records from a table in a database, you use the SELECT word. Like this:

**SELECT \* FROM Table\_Name**

SQL is not case sensitive, so the above line could be written:

**Select \* from Table\_Name**

But your SQL statements are easier to read if you type the keywords in uppercase letters. The keywords in the lines above are **SELECT** and **FROM**. The asterisk means "All Records". Table\_Name is the name of a table in your database. So the whole line reads:

**"SELECT all the records FROM the table called Table\_Name"**

You don't need to select all (\*) the records from your database. You can just select the columns that you need. The name of the table in our database is **tblContacts**. If we wanted to select just the first name and surname columns from this table, we can specify that in our SQL String:

**SELECT tblContacts.FirstName, tblContacts.Surname FROM tblContacts**

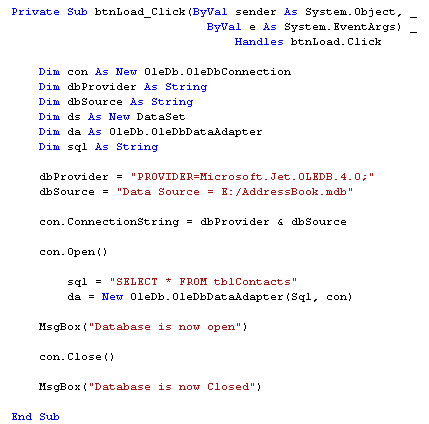
When this SQL statement is executed, only the FirstName and Surname columns from the database will be returned.

There are a lot more SQL commands, but for our purposes this is enough.

Because we want to SELECT all (\*) the records from the table called tblContacts, we pass this string to the string variable we have called sql:

**sql = "SELECT \* FROM tblContacts"**

Your code window should now look like this (though the file path to your database might be different):



Now that the Data Adapter has selected all of the records from the table in our database, we need somewhere to put those records - in the **DataSet**.

### Filling the DataSet

The Data Adapter can Fill a DataSet with records from a Table. You only need a single line of code to do this:

**da.Fill(ds, "AddressBook")**

As soon as you type the name of your Data Adapter (**da** for us), you'll get a pop up box of properties and methods. Select Fill from the list, then type a pair of round brackets. In between the round brackets, you need two things: the **Name** of your DataSet (**ds**, in our case), and an identifying name. This identifying name can be anything you like. But it is just used to identify this particular Data Adapter Fill. We could have called it "Bacon Sandwich", if we wanted:

**da.Fill(ds, "Bacon Sandwich ")**

The code above still works. But it's better to stick to something a little more descriptive than "Bacon Sandwich"!

Add the new line after the creation of the Data Adaptor:

**da = New OleDb.OleDbDataAdapter(sql, con)  
da.Fill(ds, "AddressBook")**

And that's it. The DataSet (**ds**) will now be filled with the records we selected from the table called**tblContact**. There's only one slight problem - nobody can see the data yet! We'll tackle that in the next part.

Part of an ongoing tutorial.The first part of Databases and VB .NET can be found here:

[Coding your own VB .NET database projects](http://www.homeandlearn.co.uk/net/nets12p4.html)

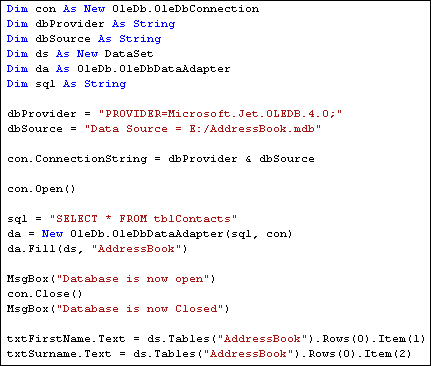
In the [previous section](http://www.homeandlearn.co.uk/net/nets12p5.html), we saw what Data Adaptors and DataSets were. We created a Data Adaptor so that it could fill a DataSet with records from our database. What we want to do now is to display the records on a Form, so that people can see them. So so this:

* Add two textboxes to your form
* Change the **Name** properties of your textboxes to **txtFirstName** and **txtSurname**
* Go back to your code window
* Add the following two lines:

**txtFirstName.Text = ds.Tables("AddressBook").Rows(0).Item(1)  
txtSurname.Text = ds.Tables("AddressBook").Rows(0).Item(2)**

You can add them after the line that closes the connection to the database. Once the DataSet has been filled, a connection to a database can be closed.

Your code should now look like this:



Before the code is explained, run your programme and click the button. You should see "John Smith" displayed in your two textboxes.

So let's examine the code that assigns the data from the DataSet to the textboxes. The first line was this:

**txtFirstName.Text = ds.Tables("AddressBook").Rows(0).Item(1)**

It's rather a long line! But after the equals sign, you type the name of your DataSet (**ds** for us). After a full stop, select **Tables** from the popup list. The **Tables** property needs something in between round brackets. Quite bizarrely, this is NOT the name of your database table! It's that identifier you used with the Data Adapter Fill. We used the identifier "**AddressBook**". If we had used "Bacon Sandwich" then we'd put this:

**ds.Tables("Bacon Sandwich")**

But we didn't, so our code is:

**ds.Tables("AddressBook")**

Type a full stop and you'll see another list popping up at you. Select **Rows** from the list. In between round brackets, you need a number. This is a Row number from the DataSet. We want the first row, which is row zero in the DataSet:

**ds.Tables("AddressBook").Rows( 0 )**

Type full stop after Rows(0) and the popup list appears again. To identify a **Column** from the DataSet, you use **Item**. In between round brackets, you type which column you want:

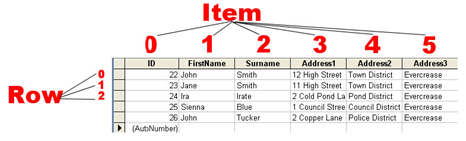
**ds.Tables("AddressBook").Rows(0).Item( 1 )**

In our Access database, column zero is used for an ID field. The **FirstName** column is the second column in our Access database. Because the Item collection is zero based, this is item 1 in the DataSet.

You can also refer to the column name itself for the Item property, rather than a number. So you can do this:

**ds.Tables("AddressBook").Rows(0).Item("FirstName")  
ds.Tables("AddressBook").Rows(0).Item("Surname")**

If you get the name of the column wrong, then VB throws up an error. But an image might clear things up. The image below shows what the items and rows are in the database.



The image shows which are the **Rows** and which are the **Items** in the Access database Table. So the**Items** go down and the **Rows** go across.

However, we want to be able to scroll through the table. We want to be able to click a button and see the next record. Or click another button and see the previous record. You can do this by incrementing the Row number. To see the next record, we'd want this:

**txtFirstName.Text = ds.Tables("AddressBook").Rows(1).Item(1)  
txtSurname.Text = ds.Tables("AddressBook").Rows(1).Item(2)**

The record after that would then be:

**txtFirstName.Text = ds.Tables("AddressBook").Rows(2).Item(1)  
txtSurname.Text = ds.Tables("AddressBook").Rows(2).Item(2)**

So by incrementing and decrementing the Row number, you can navigate through the records. Let's see how that's done.

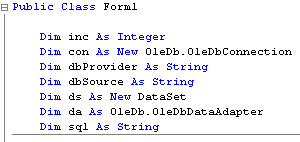
Part of an ongoing tutorial. The first part of Databases and VB .NET can be found here:

[Coding your own VB .NET database projects](http://www.homeandlearn.co.uk/net/nets12p4.html)

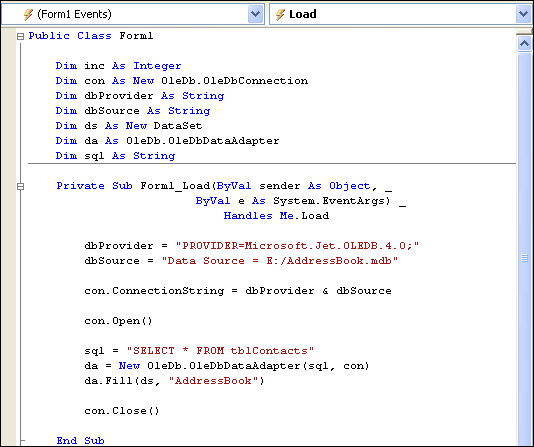
You saw in the [previous section](http://www.homeandlearn.co.uk/net/nets12p6.html) that you can navigate through the records of a database by incrementing or decrementing the Row number of the DataSet. In this section, we're going to see a more practical example of how to do that.

To navigate through the dataset, let's change our form. By adding some navigation buttons, we can duplicate [what the wizard did](http://www.homeandlearn.co.uk/net/nets12p2ed.html). We'll also need to move the code we already have. So let's start with that.

At the moment, all our code is in the Button we added to the form. We're going to delete this button, so we need to move it out of there. The variable declarations can be moved right to the top of the coding window. That way, any button can see the variables. So move your variables declarations to the top, as in the image below (don't forget to add the **Dim inc As Integer** line):



We can move a few lines to the Form Load event. So, create a Form Load event, as you did in a previous section. Now move all but the textbox lines to there. Your coding window should then look like this (you can delete the message box lines, or just comment them out):



For your button, all you should have left are these two lines:

**txtFirstName.Text = ds.Tables("AddressBook").Rows(inc).Item(1)  
txtSurname.Text = ds.Tables("AddressBook").Rows(inc).Item(2)**

Since we're going to be deleting this button, this code can be moved. Because all the buttons need to put something into the textboxes, the two lines we have left are an ideal candidate for a Subroutine. So add the following Sub to your code:

**Private Sub NavigateRecords()**

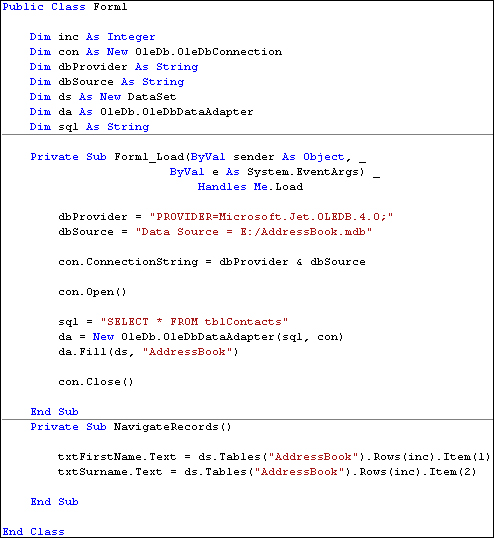
**txtFirstName.Text = ds.Tables("AddressBook").Rows(inc).Item(1)  
txtSurname.Text = ds.Tables("AddressBook").Rows(inc).Item(2)**

**End Sub**

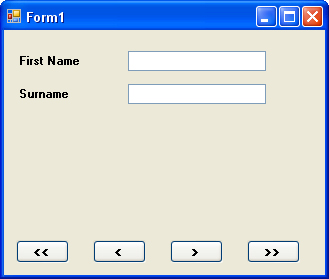
When we navigate through the DataSet, we'll call this subroutine.

Now that all of your code has gone from your button, you can delete the button code altogether. Return to you form, click on the button to select it, then press the delete key on your keyboard. This will remove the button itself from your form. (You can also right click on the button, and then select Delete from the menu.)

Here's what your coding window should like:



Now you can re-design the form. Add four new buttons, and change the Name properties to: btnNext, btnPrevious, btnFirst, and btnLast. Change the Text properties to >, <, <<, and >>. Your form will then look like this:



Just a couple of more things to set up before we get started. Add a new variable declaration to the top of your code, just under the Dim inc As Integer line. Add this:

**Dim MaxRows As Integer**

We can store how many rows are in the DataSet with this variable. You can get how many rows are in the DataSet with this:

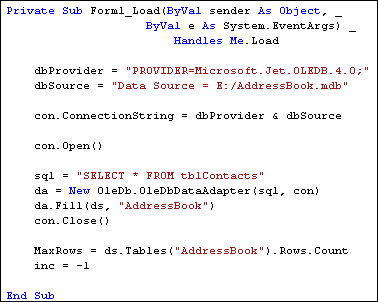
**MaxRows = ds.Tables("AddressBook").Rows.Count**

So the Rows property has a Count Method. This simply counts how many rows are in the DataSet. We're passing that number to a variable called MaxRows. You can then test what is in the variable, and see if the inc counter doesn't go past it. You need to do this because VB throws up an error message if try to go past the last row in the DataSet. (Previous versions of VB had some called an EOF and BOF properties. These checked the End of File and Before End of File. These properties have now gone.)

Add the following two lines of code to the Form Load Event of Form1:

**MaxRows = ds.Tables("AddressBook").Rows.Count  
inc = - 1**

Your code should then look like this:



Notice the other line of code for the Form Load event:

**inc = - 1**

This line sets the inc variable to minus one when the form loads. When the Buttons are clicked, this will ensure that we're moving the counter on by the correct amount.

In the next Part, we'll see how the Buttons on the form work.

Part of an ongoing tutorial. This lessons is part of an ongoing tutorial. The first part is here:

[Coding your own VB .NET database projects](http://www.homeandlearn.co.uk/net/nets12p4.html)

In [the last lesson](http://www.homeandlearn.co.uk/net/nets12p7.html), you set up a Form with four buttons and two textboxes. In this lesson, you'll add the code for the buttons.

**How to Move Forward One Record at a Time**

Double click your **Next Record** button to access the code. Add the following If … Else Statement:

**If inc <> MaxRows - 1 Then**

**inc = inc + 1**

**NavigateRecords()**

**Else**

**MsgBox("No More Rows")**

**End If**

We're checking to see if the value in **inc** does not equal the value in **MaxRows** - 1. If they are both equal then we know we've reached the last record in the DataSet. In which case, we just display a message box. If they are not equal, these two lines get executed:

**inc = inc + 1  
NavigateRecords()**

First, we move the **inc** counter on by one. Then we call the Sub we set up:

**NavigateRecords()**

Our Subroutine is where the action takes place, and the values from the DataSet are placed in the textboxes. Here it is again:

**Private Sub NavigateRecords()**

**txtFirstName.Text = ds.Tables("AddressBook").Rows(inc).Item(1)  
txtSurname.Text = ds.Tables("AddressBook").Rows(inc).Item(2)**

**End Sub**

The part that moves the record forward (and backwards soon) is this part:

**Rows( inc )**

Previously, we hard-coded this with:

**Rows( 0 )**

Now the value is coming from the variable called **inc**. Because we're incrementing this variable with code, the value will change each time the button is clicked. And so a different record will be displayed.

You can test out your Next button. Run your programme and click the button. You should now be able to move forward through the DataSet. When you get to the end, you should see the message box display "No More Rows".

None of the other button will work yet, of course. So let's move backwards.

**Move Back One Record at a Time**

To move backwards through the DataSet, we need to decrement the **inc** counter. All this means is deducting 1 from whatever is currently in inc.

But we also need to check that inc doesn't go past zero, which is the first record in the DataSet. Here's the code to add to your **btnPrevious**:

**If inc > 0 Then**

**inc = inc - 1**

**NavigateRecords()**

**Else**

**MsgBox("First Record")**

**End If**

So the If statement first checks that **inc** is greater than zero. If it is, inc gets 1 deducted from. Then the NavigateRecords() subroutine gets called. If **inc** is zero or less, then we display a message.

When you've finished adding the code, test your programme out. Click the Previous button first. The message box should display, even though no records have been loaded into the textboxes. This is because the variable **inc** has a value of -1 when the form first loads. It only gets moved on to zero when the Next button is clicked. You could amend your IF Statement to this:

**If inc > 0 Then**

**inc = inc - 1**

**NavigateRecords()**

**ElseIf inc = -1 Then**

**MsgBox("No Records Yet")**

**ElseIf inc = 0 Then**

**MsgBox("First Record")**

**End If**

This new If Statement now checks to see if inc is equal to minus 1, and displays a message if it does. It also checks if inc is equal to zero, and displays the "First Record" message box.

**Moving to the Last Record in the DataSet**

To jump to the last record in the DataSet, you only need to know how many records have been loaded into the DataSet - the **MaxRows** variable in our code. You can then set the **inc** counter to that value, but minus 1. Here's the code to add to your **btnLast**:

**If inc <> MaxRows - 1 Then**

**inc = MaxRows - 1**

**NavigateRecords()**

**End If**

The reason we're saying **MaxRows - 1** is that the row count might be 5, say, but the first record in the DataSet starts at zero. So the total number of records would be zero to 4. Inside of the If Statement, we're setting the **inc** counter to MaxRows - 1, then calling the NavigateRecords() subroutine.

That's all we need to do. So run your programme. Click the Last button, and you should see the last record displayed in your textboxes.

**Moving to the First Record in the DataSet**

Moving to the first record is fairly straightforward. We only need to set the **inc** counter to zero, if it's not already at that value. Then call the Sub:

**If inc <> 0 Then**

**inc = 0**

**NavigateRecords()**

**End If**

Add the code to your **btnFirst**. Run your programme and test out all of your buttons. You should be able to move through the names in the database, and jump to the first and last records.

As yet, though, we don't have a way to add new records, to update records, or to delete them. Let's do that next.

Part of an ongoing tutorial. This lessons is part of an ongoing tutorial. The first part is here:

[Coding your own VB .NET database projects](http://www.homeandlearn.co.uk/net/nets12p4.html)

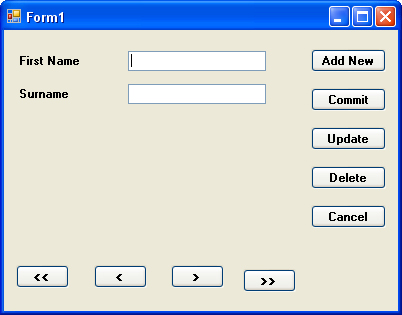
In the [last section](http://www.homeandlearn.co.uk/net/nets12p8.html), you learned how to move through the records in your DataSet, and how to display the records in Textboxes on your form. In this lesson, we'll see how to add new records, how to delete them and how to Update a records.

Before we start the coding for these new buttons, it's important to understand that the DataSet is***disconnected*** from the database. What this means is that if you're adding a new record, you're not adding it to the database: you're adding it to the **DataSet**! Similarly, if you're updating or Deleting, you doing it to the DataSet, and **NOT** to the database. After you have made all of your changes, you THEN commit these changes to the database. You do this by issuing a separate command. But we'll see how it all works.

You'll need to add a few more buttons to your form - five of them. Change the **Name** properties of the new Buttons to the following:

**btnAddNew  
btnCommit  
btnUpdate  
btnDelete  
btnClear**

Change the **Text** properties of the buttons to **Add New** **Record**, **Commit Changes**, **Update Record**, **Delete** **Record**, and **Clear/Cancel**. Your form might look something like this:



We'll start with the Update Record button

### Updating a Record

To reference a particular column (item) in a row of the DataSet, the code is this:

**ds.Tables("AddressBook").Rows(2).Item(1)**

That will return whatever is at Item 1 on Row 2.

As well as returning a value, you can also set a value. You do it like this:

**ds.Tables("AddressBook").Rows(2).Item(1) = "Jane"**

Now Item 1 Row 2 will contain the text "Jane". This won't, however, effect the database! The changes will just get made to the **DataSet**. To illustrate this, add the following code to your **btnUpdate**:

**ds.Tables("AddressBook").Rows(inc).Item(1) = txtFirstName.Text  
ds.Tables("AddressBook").Rows(inc).Item(2) = txtSurname.Text**

**MsgBox("Data updated")**

Run your programme, and click the **Next Record** button to move to the first record. "John" should be displayed in your first textbox, and "Smith" in the second textbox. Click inside the textboxes and change "John" to "Joan" and "Smith" to "Smithy". (Without the quotes). Now click your **Update Record** button. Move to the next record by clicking your **Next Record** button, and then move back to the first record. You should see that the first record is now "Joan Smithy".

Close down your programme, then run it again. Click the **Next Record** button to move to the first record. It will still be "John Smith". The data you updated has been lost! So here, again, is why:

**"Changes are made to the DataSet, and NOT to the Database"**

To update the database, you need some extra code. Amend your code to this (the new lines are in bold, red text):

**Dim cb As New OleDb.OleDbCommandBuilder(da)**

**ds.Tables("AddressBook").Rows(inc).Item(1) = txtFirstName.Text  
ds.Tables("AddressBook").Rows(inc).Item(2) = txtSurname.Text**

**da.Update(ds, "AddressBook")**

**MsgBox("Data updated")**

The first new line is this:

**Dim cb As New OleDb.OleDbCommandBuilder(da)**

To update the database itself, you need something called a **Command Builder**. The Command Builder will build a SQL string for you. In between round brackets, you type the name of your Data Adapter, **da** in our case. The command builder is then stored in a variable, which we have called **cb**.

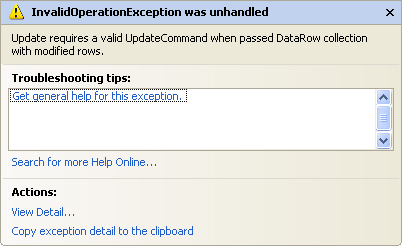
The second new line is where the action is:

**da.Update(ds, "AddressBook")**

The **da** variable is holding our Data Adapter. One of the methods of the Data Adapter is **Update**. In between the round brackets, you need the name of your DataSet (**ds**, for us). The "**AddressBook**" part is optional. It's what we've called our DataSet, and is here to avoid any confusion.

But the Data Adapter will then contact the database. Because we have a Command Builder, the Data Adapter can then update your database with the values from the DataSet.

Without the Command Builder, though, the Data Adapter can't do it's job. Try this. Comment out the Command Builder line (put a single quote before the "D" of Dim). Run your programme again, and then try and update a record. You'll get this error message:



The error is because you haven't got a command builder - a Valid Update Command.   
Delete the comment from your Command Builder line and the error message goes away.

You should now be able to make changes to the database itself (as long as the Access database isn't Read Only).

Try it out. Run your programme, and change one of the records. Click the **Update** button. Then close the programme down, and load it up again. You should see your new changes displayed in the textboxes.

### Exercise

There's one slight problem with the code above, though. Try clicking the **Update** button before clicking the **Next Record** button. What happens? Do you know why you get the error message? Write code to stop this happening

In the next part, we'll see how to add a new record.

Part of an ongoing tutorial. This lessons is part of an ongoing tutorial. The first part is here:

[Coding your own VB .NET database projects](http://www.homeandlearn.co.uk/net/nets12p4.html)

In the [previous part](http://www.homeandlearn.co.uk/net/nets12p9.html), you learned how to Update records in the database. In the part, we'll see how to add a new record to the database using VB .NET code.

**Add a New Record**

Adding a new record is slightly more complex. First, you have to add a new Row to the DataSet, then commit the new Row to the Database.

But the **Add New Record** button on our form is quite simple. The only thing it does is to switch off other buttons, and clear the textboxes, ready for a new entry. Here's the code for your **Add New Record** button:

**btnCommit.Enabled = True btnAddNew.Enabled = False  
btnUpdate.Enabled = False  
btnDelete.Enabled = False**

**txtFirstName.Clear()  
txtSurname.Clear()**

So three buttons are switched off when the **Add New Record** button is clicked, and one is switched on. The button that gets switched on is the Commit Changes button. The Enabled property of**btnCommit** gets set to **True**. But, for this to work, you need to set it to **False** when the form loads. So return to your Form. Click **btnCommit** to select it. Then locate the **Enabled** Property in the Properties box. Set it to **False**. When the Form starts up, the button will be switched off.

The Clear/Cancel button can be used to switch it back on again. So add this code to your btnClear:

**btnCommit.Enabled = False  
btnAddNew.Enabled = True  
btnUpdate.Enabled = True  
btnDelete.Enabled = True**

**inc = 0  
NavigateRecords()**

We're switching the **Commit Changes** button off, and the other three back on. The other two lines just make sure that we display the first record again, after the Cancel button is clicked. Otherwise the textboxes will all be blank.

To add a new record to the database, we'll use the **Commit Change**s button. So double click your**btnCommit** to access its code. Add the following:

**If inc <> -1 Then**

**Dim cb As New OleDb.OleDbCommandBuilder(da)  
Dim dsNewRow As DataRow**

**dsNewRow = ds.Tables("AddressBook").NewRow()**

**dsNewRow.Item("FirstName") = txtFirstName.Text  
dsNewRow.Item("Surname") = txtSurname.Text**

**ds.Tables("AddressBook").Rows.Add(dsNewRow)/p>**

**da.Update(ds, "AddressBook")**

**MsgBox("New Record added to the Database")**

**btnCommit.Enabled = False  
btnAddNew.Enabled = True  
btnUpdate.Enabled = True>  
btnDelete.Enabled = True**

**End If**

The code is somewhat longer than usual, but we'll go through it.

The first line is an If Statement. We're just checking that there is a valid record to add. If there's not, the**inc** variable will be on minus 1. Inside of the If Statement, we first set up a **Command Builder**, [as before](http://www.homeandlearn.co.uk/net/nets12p9.html). The next line is this:

**Dim dsNewRow As DataRow**

If you want to add a new row to your DataSet, you need a **DataRow** object. This line just sets up a variable called **dsNewRow**. The type of variable is a DataRow.

To create the new DataRow object, this line comes next:

**dsNewRow = ds.Tables("AddressBook").NewRow()**

We're just saying, "Create a New Row object in the AddressBook DataSet, and store this in the variable called dsNewRow." As you can see, **NewRow()** is a method of **ds.Tables**. Use this method to add rows to your DataSet.

The actual values we want to store in the rows are coming from the textboxes. So we have these two lines:

**dsNewRow.Item("FirstName") = txtFirstName.Text  
dsNewRow.Item("Surname") = txtSurname.Text**

The **dsNewRow** object we created has a Property called **Item**. This is like the Item property you used earlier. It represents a column in your DataSet. We could have said this instead:

**dsNewRow.Item(1) = txtFirstName.Text  
dsNewRow.Item(2) = txtSurname.Text**

The **Item** property is now using the index number of the DataSet columns, rather than the names. The results is the same, though: to store new values in these properties. We're storing the text from the textboxes to our new Row.

We now only need to call the Method that actually adds the Row to the DataSet:

**ds.Tables("AddressBook").Rows.Add(dsNewRow)**

To add the Row, you use the **Add** method of the Rows property of the DataSet. In between the round brackets, you need the name of your DataRow (the variable **dsNewRow**, in our case).

You should know what the rest of the code does. Here's the next line:

**da.Update(ds, "AddressBook")**

Again, we're just using the **Update** method of the Data Adapter, just like last time. The rest of the code just displays a message box, and resets the button.

But to add a new Row to a DataSet, here's a recap on what to do:

* Create a **DataRow** variable
* Cretae an Object from this variable by using the **NewRow()** method of the DataSet **Tables**property
* Assign values to the **Items** in the new Row
* Use the **Add** method of the DataSet to add the new row

A little more complicated, but it does work! Try your programme out. Click your **Add New Record**button. The textboxes should go blank, and three of the buttons will be switched off. Enter a new First Name and Surname, and then click the **Commit Changes** button. You should see the message box telling you that a new record has been added to the database. To see the new record, close down your programme, and run it again. The new record will be there.

In the next part, you'll learn how to delete a record from the database.

Part of an ongoing tutorial. This lessons is part of an ongoing tutorial. The first part is here:

[Coding your own VB .NET database projects](http://www.homeandlearn.co.uk/net/nets12p4.html)

In the [last part](http://www.homeandlearn.co.uk/net/nets12p10.html), you saw how to Add a new record to the database using VB .NET code. In this final part, you'll learn how to delete records.

### Deleting Records from a Database

The code to delete a record is a little easier than last time. Double click your **btnDelete** and add the following:

**Dim cb As New OleDb.OleDbCommandBuilder(da)**

**ds.Tables("AddressBook").Rows(inc).Delete()  
MaxRows = MaxRows - 1**

**inc = 0  
da.Update(ds, "AddressBook")  
NavigateRecords()**

You've met most of it before. First we set up a Command Builder. Then we have this line:

**ds.Tables("AddressBook").Rows(inc).Delete()**

Just as there is an **Add** method of the DataSet Rows property, so there is a **Delete** method. You don't need anything between the round brackets, this time. We've specified the Row to delete with:

**Rows( inc )**

The **inc** variable is setting which particular Row we're on. When the **Delete** method is called, it is this row that will be deleted.

However, it will only be deleted from the DataSet. To delete the row from the underlying database, we have this again:

**da.Update(ds, "AddressBook")**

The Command Builder, in conjunction with the Data Adapter, will take care of the deleting. All you need to is call the **Update** method of the Data Adapter.

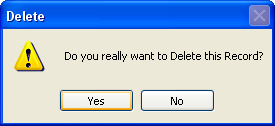
The **MaxRows** line in the code just deducts 1 from the variable. This just ensures that the number of rows in the DataSet matches the number we have in the MaxRows variable.

We also reset the **inc** variable to zero, and call the **NavigateRecords()** subroutine. This will mean that the first record is displayed, after a record has been deleted.

Try out your programme. Click the **Next Record** button a few times to move to a valid record. Then click the **Delete Record** button. The record will be deleted from the DataSet AND the database. The record that is then displayed will be the first one.

There's another problem, though: if you click the **Delete Record** button before the**Next Record**button, you'll get an error message. You can add an If Statement to check that the inc variable does not equal minus 1.

Another thing you can do is to display a message box asking users if they really want to delete this record. Here's one in action:



To get this in your own programme, add the following code to the very top of your Delete button code:

**If MessageBox.Show("Do you really want to Delete this Record?", "Delete", MessageBoxButtons.YesNo, MessageBoxIcon.Warning) = DialogResult.No Then**

**MsgBox("Operation Cancelled")  
Exit Sub**

**End If**

The first two lines of the code are really one line, spread out so as to fit on this page.

But we're using the new message box function:

**MessageBox.Show()**

In between the round brackets, we specifying the message to display, followed by a caption for the message box. We then have this:

**MessageBoxButtons.YesNo**

You won't have to type all that out; you'll be able to select it from a popup list. But what it does is give you Yes and No buttons on your message box.

After typing a comma, we selected the **MessageBoxIcon**.Warning icon from the popup list.

But you need to check which button the user clicked. This is done with this:

**= DialogResult.No**

Again, you select from a popup list. We want to check if the user clicked the No button. This will mean a change of mind from the user. A value of No will then be returned, which is what we're checking for in the If Statement.

The code for the If Statement itself is this:

**MsgBox("Operation Cancelled")  
Exit Sub**

This will display another message for the user. But most importantly, the subroutine will be exited: we don't want the rest of the Delete code to be executed, if the user clicked the No button.

And that's it for our introduction to database programming. You not only saw how to construct a database programme using the Wizard, but how to write code to do this yourself. There is an awful lot more to database programming, and we've just scratched the surface. But in a beginner's course, that's all we have time for.

The section that follows is all about Forms.

n this section of the course, we'll take a look at some of the extra things you can do with VB.NET forms. First, we'll take a look at the Anchor and Dock properties of a form.

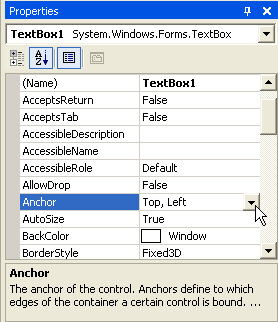
**Anchoring and Docking**

The Anchor and Dock properties of a form are two separate properties. Anchor refers to the position a control has relative to the edges of the form. A textbox, for example, that is anchored to the left edge of a form will stay in the same position as the form is resized. Docking refers to how much space you want the control to take up on the form. If you dock a control to the left of the form, it will stretch itself to the height of the form, but its width will stay the same. Let's take a look at some examples, to clear things up.

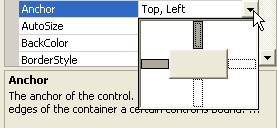
**Anchoring**

Start a new windows projects. Add two textboxes to your form, and set the MultiLine properties of both to True. Change the height of the boxes.

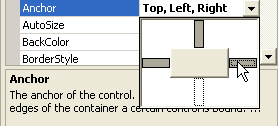
Click on Textbox1 and locate the Anchor property in the Properties box:



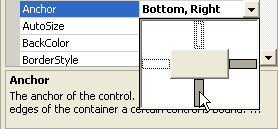
The default is to anchor the control to the Top, Left edge of the form. Click the arrow to reveal a curious drop down box:



The button in the middle represents your control. The big white areas are rather confusing - they don't actually do anything! To change the property, you click the smaller grey or white rectangles between the big white rectangle. Click again to deselect it. In the image below, the property has been changed so that the textbox is anchored to the Top, Left and Right sides of the form:



The next image has the textbox anchored to the Right and Bottom edges of the Form:



Notice where the cursor is in the images, and what has been changed. Click the arrow on the drop down box to confirm your choices.

To see what effect this all has, do the following:

* Set the Anchor property of Textbox1 on the default of **Top**, **Left**
* Change the Anchor property of Textbox2 to **None** (all the small rectangles should be white.)
* Run your programme and drag the edges of the Form outward. This will resize your form

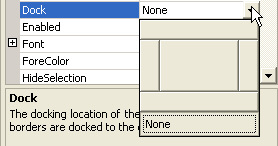
What you should notice is that Textbox1 stays where it is, and that the left edge of Textbox2 moves.

Stop your programme from running. Change the Anchor properties of the two textboxes to anything you like. Run your form again and watch what happens. Try anchoring one textbox to the left and right of the form. Watch what happens.

But anchoring a control to an edge of the form is a useful property to get used to, if you have a form that can be resized and want your controls to stay where they are.

**Docking**

Docking is similar to Anchoring, but this time the control fills a certain area of the form. To see how it works, click on one of your textboxes and locate the Dock property. Click the arrow to reveal a drop down box:



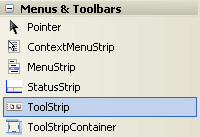
This time, all the rectangles are like buttons. You can only dock to one side at a time, and the default is None. Click a button to see what it does to your textbox. Click the middle one, and the textbox will Fill the whole form.

Docking is quite useful when used with the splitter control and panels, allowing you to create a Windows-style interface.

In the next part, we'll take a look at how to add a Toolbar to your Form.

The toolbar is a very popular and much-used addition to a programme. It's difficult to think of a piece of software that doesn't make use of them. VB.NET lets you add toolbars to your forms, and the process is quite straightforward. Let's see how it's done:

Either start a new Windows project, or keep the one you currently have. To add a toolbar to the top of your form, expand the Toolbox and locate the ToolStrip control:



Double click the ToolStrip control, and it will be added to the top of your form:

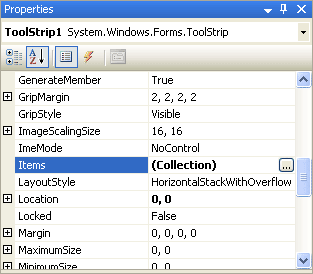


You should also notice the ToolStrip object that appears at the bottom of the window:

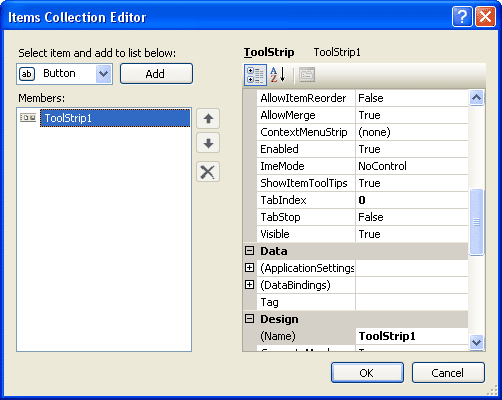
The Toolstrip object

ToolStrips work by adding buttons and images to them. The button is then clicked, and an action performed.

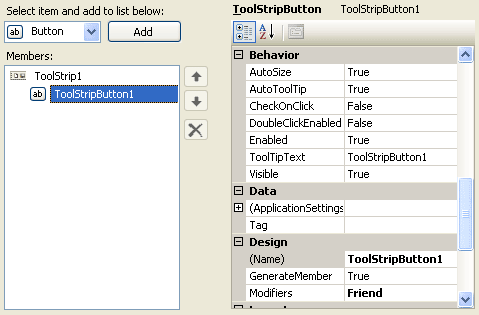
Click on your ToolStrip to select it. In the property box for the ToolStrip, you'll notice that it has the default Name of **ToolStrip1**. We'll keep this Name. But locate the Items (Collection) property:



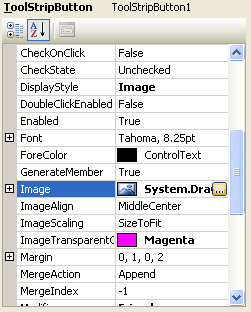
Click the button with the three dots in it. This brings up the Items Collection Editor:



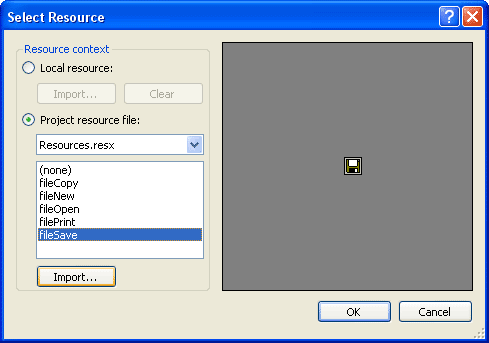
To add a new button to your ToolStrip, click the **Add** button at the top. The button appears in the Members box (ToolStripButton1):



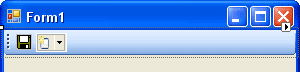
Notice that the new button has its own list of properties, just to the right. To add an image to this new button, locate the Image property:



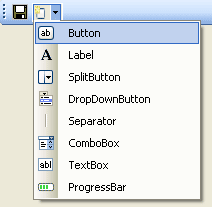
Click the small button with the 3 dots in it to bring up the Select Resource box:



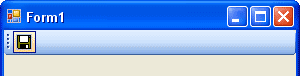
In the image above, we've selected "Project resource file", and then clicked the Import button. We then navigated to some Bitmap images and imported the five that you can see in the screenshot above. (The Bitmap folder is amongst the files you download at the start of this book.) Click OK when you have imported some images. You will be returned to the Item Collection Editor. Click OK on this, as well. The ToolStrip on your form will then look like this:



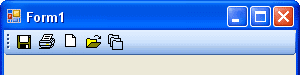
The second of those images is a dropdown list of available ToolStrip options:



So if you want, say, a separator instead of a button, select it from the list. This dropdown list will disappear when you run the form:



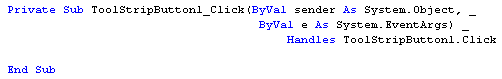
Repeat the steps outlined above, and add some more buttons to your ToolStrip. It should then look something like ours:



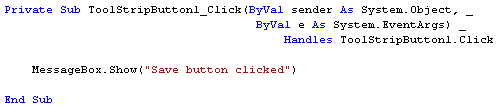
Of course, if you click on the buttons nothing will happen. We need to write the code that gets them to work.

### Coding For your Toolbar buttons

Double click your first ToolStripbutton to bring up the coding window. It should look like this:



You can place any code you like, here. Try a message box, as in the image below:



Run your programme and click your ToolStrip button. You should see the message box display. In a real programme, however, the code would be the same code for a menu item - it's just shortcut, after all!

In the next part, you'll learn how to create multiple forms in a VB NET project.

It's a rare programme that only has one form in it. Most programmes will have other forms. These other forms can be used for things like Find and Replace searches, extra formatting capabilities, to set Options for the programme, and a whole lot more besides. VB.NET let's you add as many forms as you want to your project. But the process is not quite so simple. We'll see how to do it, though.

You can use the form you already have for this, the one with [the ToolStrip](http://www.homeandlearn.co.uk/net/nets13p2.html) on it (or start a new project, if you prefer). But from the VB.NET design environment, click the **Project** menu. From the drop down menu, click**Add Windows Form**. The Add New Item dialogue box appears.

Select **Windows Form** under Templates. Then click inside the Name textbox at the bottom. Change the Name of the form to **frmSecond.vb**. Then click Add.

When you are returned to the design environment, your new form will be displayed:

Two Forms created

To switch between forms, you can click the tabs. In the image, two tabs are displayed: Form1 (the original and first form), and our new form **frmSecond**.

We'll write code to get this new form to display. But it will only appear when a button is clicked on Form1.

So click the tab for Form1, and add a button to this form. Change the **Name** property of the button to**btnShowSecond**. Then double click the button to access the code for it.

In order to display the second form, you have to bear in mind that Forms are Classes. So **frmSecond**is a Class (as is Form1). You first have to create a new object from the class called frmSecond Class. Then call its Show method.

So add this code to your button

**Dim SecondForm As New frmSecond**

**SecondForm.Show()**

The first line sets up a variable called **SecondForm**. When you type "**As New**", you're asking VB.NET to create a New object. If you type a space, you'll see a pop up list. Type the **frm** of frmSecond and you should see it displayed on the list. You can double click the item in the list to add it to your code. But what the line does is create a new Object from the Class called **frmSecond**.

Once we have the Form Object stored in the variable, we can just use the Show method to display the form.

Run your programme and test it out. When you click your button, you should see the second form appear.

However, there's a problem with this code. Click the button again and another copy of **frmSecond**appears. Keep clicking the button and your screen will be filled with the second form!

To prevent this from happening, you can move the code that creates the form object. Move it right to the top of the coding window, just below **Public Class Form1**.

The only code left in the button is the line that Shows the form. A new form object will now not be created every time the button is clicked. If you try it out, you should see only one form appear when the button is clicked, and not multiple forms.

In the next part, we'll take a look at Modal and Non Modal forms.

*This lesson follows on from the previous short lesson:*[*How to create a second form*](http://www.homeandlearn.co.uk/net/nets13p3.html)

A modal from is one that has to be dealt with before a user can continue. An example is the Change Case dialogue box in Microsoft Word. If you try to click away from the dialogue box, you'll here a beep to indicate an error. Until you click either the Cancel or OK buttons, the programme won't let you click anywhere else.

The second form you've [just created](http://www.homeandlearn.co.uk/net/nets13p3.html) is called a Modeless form. These are forms than can be hidden or sent to the taskbar. You can then return to the main form or programme and do things with it.

A Modal form is sometimes called a dialogue box. And we'll see how to create one of these now.

Add a second button to your Form1. Change the **Name** property of the new button to**btnDialogueBox**. Double click the new button and add the following code:

**Dim frmDialogue As New frmSecond**

**frmDialogue.ShowDialog()**

To display a form as a Modal dialogue box, you use the **ShowDialog** method. If you use the Show method, the form is displayed as a Modeless form.

Run your programme. Click your new button, and the second form should display. Move it out the way and try to click a button on Form1. You won't be able to. The second form has to be dealt with before you can access Form1.

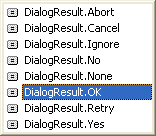
When the form is a Modal dialogue box, you can create OK and Cancel buttons for it. VB.NET then has a trick up its sleeve for these types of buttons. We'll see that trick now.

### OK and Cancel Buttons

In the design environment, Click the Tab for your **frmSecond**. When the form is displayed in the design window, add two buttons to it (Make sure you're adding the buttons to the second form and NOT Form1). Change the **Name** property of the first button to **btnOK**, and the **Name** property of the second to **btnCancel**. Double click your OK button and add the following code to it:

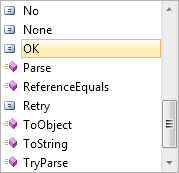
**Me.DialogResult = DialogResult.OK**

The **Me** keyword refers to the current form. When you type a full stop, select **DialogResult** from the pop up list that appears. DialogResult is a property of the Form. It can accept a range of values. As soon as you type a space after the equals sign, you'll see a list with these values on it (VB NET 2008 only. In VB 2010, you have to type the DialogResult):



As you can see, in VB NET 2008, one of these values is **DialogResult.OK**. This indicates that you want to use this button as an OK button. When the button is clicked, VB.NET will return a result of OK for this button.

In VB NET 2010, type **DialogResult** after the equals sign. Type a dot and you'll have this instead of the above image:



Access the code for your Cancel button and add the following line:

**Me.DialogResult = DialogResult.Cancel**

For the Cancel button, we're just selecting **DialogResult.Cancel** from the list. When the button is clicked, VB.NET will return a result of Cancel for this button.

You can test to see what value is stored in **Me.DialogResult**. But you do that from the button that displays the form, **Form1** for us.

So access your Form1 code, and locate the lines that display the second form. The two lines should be these:

**Dim frmDialogue As New frmSecond**

**frmDialogue.ShowDialog()**

Change the second line to this:

**If frmDialogue.ShowDialog() = DialogResult.OK Then**

**MsgBox("OK Button Clicked")**

**End If**

To get at the value of the button clicked, you test to see what result the **ShowDialog** property is. If the**ShowDialog** property of **frmDialogue** is **OK** then you can execute the code that needs executing. If the Cancel button was clicked, however, you don't have to do anything: VB.NET will take of closing your Modal dialogue box for you!

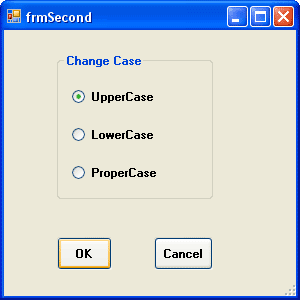
Run your programme and test it out. Click your button to bring up your Modal dialogue box. Click the OK button, and you should see the message box display. Bring the Modal dialogue box up a second time and then click the Cancel button. The form will just close down.

In the next part, we'll see how to return values from a second form.

This lesson follows on from the previous two lessons: [How to create a second form](http://www.homeandlearn.co.uk/net/nets13p3.html)

The form with OK and Cancel buttons on it is not doing much good. We need it do some work for us. Let's turn the form into a Change Case dialogue box.

Design a Form like the one in the following image (this is **frmSecond**):



When you've designed your form, click back on Form1 and add a Textbox to it. When the button on Form1 is clicked, the dialogue box above will display. You can then select an option button to change the case to Upper, Lower or Proper case. This will happen when the OK button is clicked. Whatever text is in Texbox1 on Form1 will be changed accordingly.

Double click the OK button on **frmSecond** to access the code. You should have the following:

**Me.DialogResult = DialogResult.OK**

If you want to refer to Texbox1 on Form1, you can't just do this:

**Form1.Textbox1.Text**

In previous version of VB, that code would be all right. You're saying "Access the Text property of Textbox1 on Form1." The problem in VB.NET is that forms are Classes. They don't become objects until one is created from a Class. So the frmSecond Class knows nothing about Form1. It has no idea what it is.

The solution is to create a textbox object variable on Form1, and assign Textbox1 to this variable. But this variable has to be something that all Classes in the project can see.

So add this near the top of your code window for Form1 (add it just below the Inherits System.Windows.Forms.Form line, or Public Class Form1):

**Public Shared tb As TextBox**

We're setting up a variable which we've called **tb**. A Textbox object is going to be stored in this variable. But notice that the variable is **Public Shared**. This way, **frmSecond** will be able to see the variable.

In the Form Load event for Form1, add the following line:

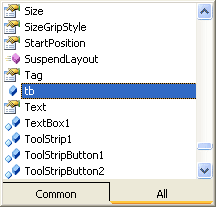
**tb = Textbox1**

When Form1 loads, the textbox called Textbox1 will be assigned to the tb variable. Now Textbox1 can be seen by frmSecond.

Go back to your code for the OK button on frmSecond. Add the following two lines at the top:

**Dim ChangeCase As String  
ChangeCase = Form1.tb.Text**

We're setting up a String variable called ChangeCase. Whatever text is in Textbox1 of Form1 will then be assigned to the ChangeCase variable. But notice that as soon as you type a full stop after Form1, the **tb** variable will be available in the pop up list:



The Public variable called tb holds a reference to Textbox1 on Form1. When you type a full stop after the tb, you get a list popping up. The list is all the Properties and Methods that are available to Textbox1. One of these is the Text property.

We now only need to add the code that does the actual converting. So add this below the two lines you already have:

**Dim ChangeCase As String  
ChangeCase = Form1.tb.Text**

**If optUpper.Checked Then**

**ChangeCase = ChangeCase.ToUpper**

**ElseIf optLower.Checked Then**

**ChangeCase = ChangeCase.ToLower**

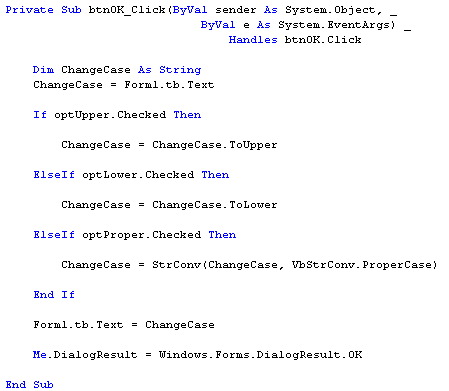
**ElseIf optProper.Checked Then**

**ChangeCase = StrConv(ChangeCase, VbStrConv.ProperCase)**

**End If**

**Form1.tb.Text = ChangeCase**

The three options buttons on our form were called **optUpper**, **optLower** and **optProper**. In the code, we're using an If Statement to see which of these was selected. The one that was chosen will have its**Checked** property set to **True**. We then store into the variable **ChangeCase** the converted text from the textbox. The final line puts the converted text back into Textbox1 on Form1. But you're coding window should look like this:



Note that the DialogResult.OK line is the final line of the code. When you're writing your code, make sure that optUpper, optLower and optProper are changed to whatever you called your Radio Buttons.

When you're finished adding the code, run your programme. Enter some text into Textbox1. Then click the button that brings up the Change Case Dialogue box. Select an option from the three available, and the click OK. The text in Textbox1 should be converted.

Setting and Getting value from one form to another can be quite a tricky process at first. But once you get the hang of it you'll find it's not too difficult.

And that ends this section of this course. There's an awful lot more to learn about Windows Forms, and a bit of experimentation is needed before you become skilled in their use. But in a beginners course, you've learned enough to be going on with.