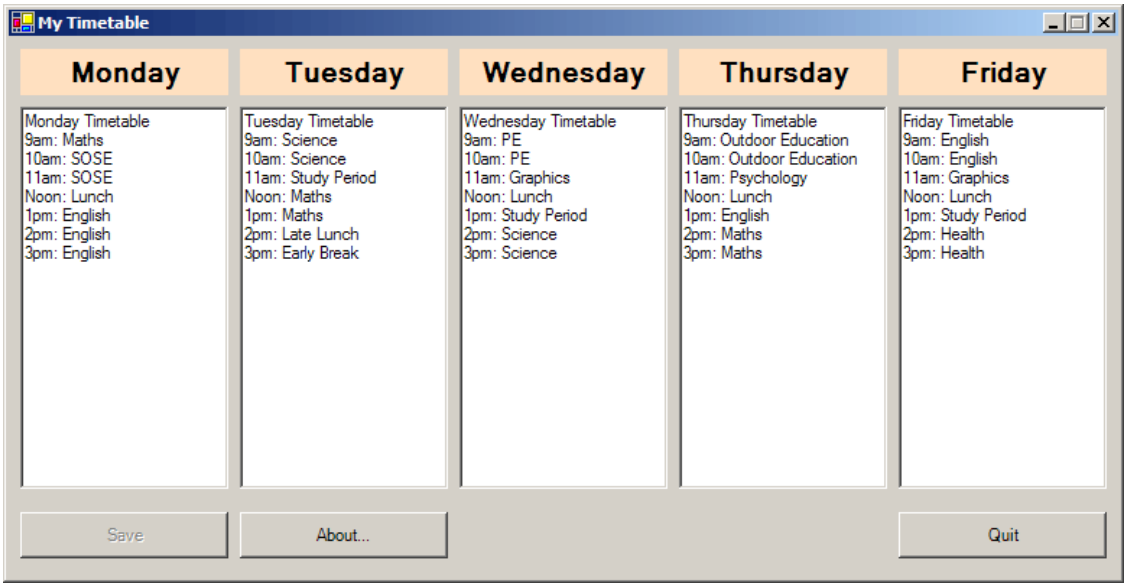
Student Timetable Manager

Problem Description: Design a program that can be used by a student to keep track of their weekly timetable. The program should be easy to use and store each of the days' activities in separate text files for later access (one for each day). It should also incorporate an 'About' button which, when clicked, gives some information about the program in a new dialog.

Skills Covered: Text file manipulation, form design, multiple forms.

To help students keep track of their weekly timetables, the program will display a number of areas in which the details of each day's classes can be entered. This information will be saved and loaded to and from text files which will enable the program to remember the information that has been entered into it.



Setting up the objects for our program

Follow these steps to create the program.

1. Start a new Visual Basic.NET program called ‘StudentTimeTable’.

2. Name the form ‘frmTimeTable’. This form will be used to hold all of the controls for our program.

3. Create five Labels and place them along the top of the form. These labels will be used as headings for each of the days of the school week. Name the labels ‘lblMonday’, ‘lblTuesday’, ‘lblWednesday’, ‘lblThursday’ and ‘lblFriday’.

4. Create five TextBoxes and place these beneath the labels. These text boxes will be used to hold the timetable details for each day. Textboxes have been used so that the user can click inside them and edit or add new information. Name the text boxes ‘txtMonday’, ‘txtTuesday’, ‘txtWednesday’, ‘txtThursday’, ‘txtFriday’. You will also need to change the AutoSize property to ‘False’ or you will not be able to make them the required shape.

5. Draw three Buttons at the bottom of the form. The first Button will be used to save the information that is currently in the TextBoxes. Name this button ‘btnSave’.

6. The second Button will be used to display information about the program. Name this Button ‘btnAbout’.

7. The third Button will be used to quit the program. Name this Button ‘btnQuit’.

8. You will also need to create some text files so that when the program is run for the first time, it is able to read some data in. The text files can be blank, but they must exist. Create five text files and place them in the ‘bin’ folder of this program. Name them ‘Monday.txt’, ‘Tuesday.txt’, ‘Wednesday.txt’, ‘Thursday.txt’ and ‘Friday.txt’.

Changing the properties of the objects

Click on each object in turn and set the following properties:

frmTimeTable

StartPosition: CenterScreen
Text: My Timetable

By setting the form's properties in this way, we can create a borderless form that appears on the screen that cannot be moved, closed or resized.

lblMonday, lblTuesday, lblWednesday, lblThursday, lblFriday

Text: Monday, Tuesday, Wednesday, Thursday, Friday (respectively)
TextAlign: MiddleCenter
Set the Font and BackColor properties yourself, so that each heading is colourful and easy to read.

txtMonday, txtTuesday, txtWednesday, txtThursday, txtFriday

Text: *Blank*
AutoSize: False
MultiLine: True
Set the Font and BackColor properties yourself, so that the text is colourful and easy to read.

btnSave

Text: Save
Enabled: False
Set the Font and BackColor properties yourself, so that the button is colourful and easy to read.

btnAbout

Text: About…
Set the Font and BackColor properties yourself, so that the button is colourful and easy to read.

btnQuit

Text: Quit
Set the Font and BackColor properties yourself, so that the button is colourful and easy to read.

Adding code to our program

Some notes about the code

Generally, the code listed for each particular object includes the 'Private Sub' declaration, which (in most cases) you will not need to type in from scratch. At some points within the code, you will notice an underscore '_' character at the end of a line of code. This can be used if you wish to split a single line of code over several lines. If you would prefer to type the line in as a single line, you can remove the underscore character and combine it with the line underneath. Make sure that when you type the underscore character in, that it is preceded by a space.

1. The first thing that we need to do, is to set up our program so that it will be able to read and write from a text file. To do this, switch to code view and move to the line about 'Public Class frmTimetable'. Type in the following line of code:

   ```
   Imports System.IO
   ```

   This line simply tells the Visual Basic.Net compiler to include the code library containing the commands for reading and writing from text files, into our program.

2. Let's start by putting some code into our 'Quit' button. Double click on 'btnQuit' to bring up its code and type the following:

```
Private Sub btnQuit_Click(ByVal sender As System.Object, ByVal _
e As System.EventArgs) Handles btnQuit.Click

      If btnSave.Enabled = False Then

            'No changes have been made since last opening, _
            close the form

            Me.Dispose()
      Else

      'Some changes have been made - tell the user this

            If MsgBox("Changes to your timetable since the _
            last visit will be lost! Are you sure you want to _
            quit?", MsgBoxStyle.OKCancel) = MsgBoxResult.OK Then

                  'They clicked OK, so quit.

                  Me.Dispose()
            End If
      End If

End Sub
```

What is this code doing? Firstly, we have the `'Private Sub'` declaration of the object's code, which is generated for us automatically. We won't worry about this. Then, we have an 'If-Else' statement, with another 'If' statement nested inside it. The first 'If' statement makes use of the fact that we will use the 'Enabled' property of 'btnSave' to determine whether any changes have been made to the TextBoxes since the program started running (or the last save).

When the program starts, 'btnSave' has 'Enabled = False' (which makes sense, because no changes have been made to the information, making saving it rather redundant). When the user alters the information (types in a TextBox), we set 'Enabled = True' for 'btnSave', allowing the user to save any changes. Hence, we can tell whether any changes have been made based on the Enabled value of 'btnSave', as the first 'If' statement is doing in the code above. If 'btnSave' is not enabled, then no changes have been made since the last save, and we can quit the program (using the code '`Me.Dispose()`'). Otherwise, changes *have* been made, and we must alert the user to this using the 'MsgBox()' function. The first argument 'MsgBox()' takes is a string containing the prompt for the user. The second is a constant defining which buttons will appear in the prompt – our above selection of '`MsgBoxStyle.OKCancel`' gives us (unsurprisingly) an 'OK' button and a 'Cancel' button.
We only wish to perform an action (quitting) if the user clicks 'OK' to disregard changes, so we compare the MsgBox() function with '`MsgBoxResult.OK`' in our second 'If' statement. If the two are equal (i.e. the user clicked 'OK'), we exit the program, once again using the '`Me.Dispose()`' command. If the user clicks 'Cancel', the dispose statement will not be triggered and nothing will happen.

3. We will now add some code that will set 'btnSave.Enabled' to 'True' whenever the user alters the data by typing in a TextBox. We *could* do this by adding the following code to each of the textboxes' 'TextChanged' events:

```
Private Sub txtMonday_TextChanged(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
txtMonday.TextChanged

      btnSave.Enabled = True

End Sub
```

This, however, would lead to repetition of unnecessary code. What would be better is to write our own 'Private Sub' that is activated by the user typing in any of the five textboxes on the form. Move to a blank line below one of the 'End Sub' lines and type in the following 'Private Sub':

```
Private Sub text_TextChanged(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles txtMonday.TextChanged, _
txtTuesday.TextChanged, txtWednesday.TextChanged, _
txtThursday.TextChanged, txtFriday.TextChanged

      btnSave.Enabled = True

End Sub
```

This code creates a new private subroutine called 'text_TextChanged'. The key part of the declaration is the list of events after the '`Handles`' keyword. This is a list of all actions that will trigger this particular piece of code – in this case, changing any of the text in any of the textboxes will trigger the code and enable 'btnSave'.

4. There are two occasions when we will wish to access the text files storing our data – to load the information when our program starts, and to save the information when the user clicks the 'Save' button. We will deal with the code that is run whenever the form is loaded first:

```
Private Sub frmTimeTable_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load

      'Declare the variables we will be using to access our files

      Dim MyReader As StreamReader
      MyReader = File.OpenText("monday.txt")
      txtMonday.Text = MyReader.ReadToEnd
      MyReader.Close()

      MyReader = File.OpenText("tuesday.txt")
      txtTuesday.Text = MyReader.ReadToEnd
      MyReader.Close()

      MyReader = File.OpenText("wednesday.txt")
      txtWednesday.Text = MyReader.ReadToEnd
      MyReader.Close()

      MyReader = File.OpenText("thursday.txt")
      txtThursday.Text = MyReader.ReadToEnd
      MyReader.Close()

      MyReader = File.OpenText("friday.txt")
      txtFriday.Text = MyReader.ReadToEnd
      MyReader.Close()

End Sub
```

As you can see, file access in Visual Basic.NET is very simple. We firstly declare the variable 'MyReader' as type 'StreamReader'. This allows us to access the files by issuing the '.OpenText' method and passing the name of a text file to the

reader. We then assign the data read from the file to the relevant text box before closing the reader. This last step is important because it frees the file for later access.

5. The code used to write the contents of the TextBoxes back into the text files is equally as simple, as shown below:

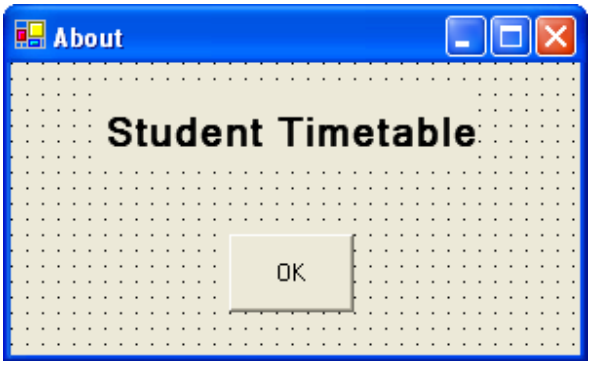
```
Private Sub btnSave_Click(ByVal sender As System.Object, ByVal _
e As System.EventArgs) Handles btnSave.Click

      Dim MyWriter As StreamWriter

      MyWriter = File.CreateText("monday.txt")
      MyWriter.Write(txtMonday.Text)
      MyWriter.Close()

      MyWriter = File.CreateText("tuesday.txt")
      MyWriter.Write(txtTuesday.Text)
      MyWriter.Close()

      MyWriter = File.CreateText("wednesday.txt")
      MyWriter.Write(txtWednesday.Text)
      MyWriter.Close()

      MyWriter = File.CreateText("thursday.txt")
      MyWriter.Write(txtThursday.Text)
      MyWriter.Close()

      MyWriter = File.CreateText("friday.txt")
      MyWriter.Write(txtFriday.Text)
      MyWriter.Close()

      btnSave.Enabled = False

End Sub
```

   There a few differences to take note of here. Firstly, the variable declared is of type 'StreamWriter' rather than 'StreamReader'. You will also notice that the method used to open each file is '.CreateText' rather than '.OpenText'. The '.Write' method of the 'StreamWriter' writes whatever is passed to it as an argument (in this case the text of our textboxes) into the currently open file. Again, it is important to close the files after writing to them.

   The very last line of this code disables 'btnSave' to avoid the redundancy of writing the unchanged data into the files again.

6. We will now create a form to display a message about the program – what it is, who it was created by, and when. Go to the 'Project' menu and click 'Add Windows Form'. Rename the form to 'frmAbout' and add as many labels as needed, along with a button which you should rename to 'btnOK'. When done,

your form should look something like the one below. You should set the 'StartPosition' of the form to 'CenterParent'.



The only code you will need to add for this form is to close the form when the user clicks 'OK'. Double click on the 'OK' button and enter the following code:

```
Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles btnOK.Click

      Me.Dispose()

End Sub
```

7. The last piece of code you will need to add is to 'btnAbout' on the main form, which will pop up the about form when clicked. Here is the code you should add:

```
Private Sub btnAbout_Click(ByVal sender As System.Object, ByVal _
e As System.EventArgs) Handles btnAbout.Click

      Dim AboutForm As New frmAbout
      AboutForm.ShowDialog()

End Sub
```

This highlights an important concept change from Visual Basic 6 to Visual Basic.NET. Your creation of 'frmAbout' has not created an instance of that form, but rather a class which you can subsequently use. Therefore, you must declare a new instance of the form with the '`Dim AboutForm As New frmAbout`' statement. The method '.ShowDialog()' displays the form in a mode that doesn't allow the user to return to the main form without first closing the about form.

Testing the program

You should now be able to test the program by creating the required text files, then running the program and altering and saving the data.

Further ideas to develop

- Use a single text file for all of the data.
- Add error handling and checking into the processing of the files (boring, yes, but an essential part of programming).
- Add a timer to automatically save the data every 10 seconds.