JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2006**

Vol.5, No. 2, March-April 2006

A Pedagogical Experiment: Introducing Students to a First GUI Project Using C#/.NET

Richard Wiener, Editor-in-Chief, JOT, Associate Professor, Department of Computer Science, University of Colorado at Colorado Springs

1 INTRODUCTION

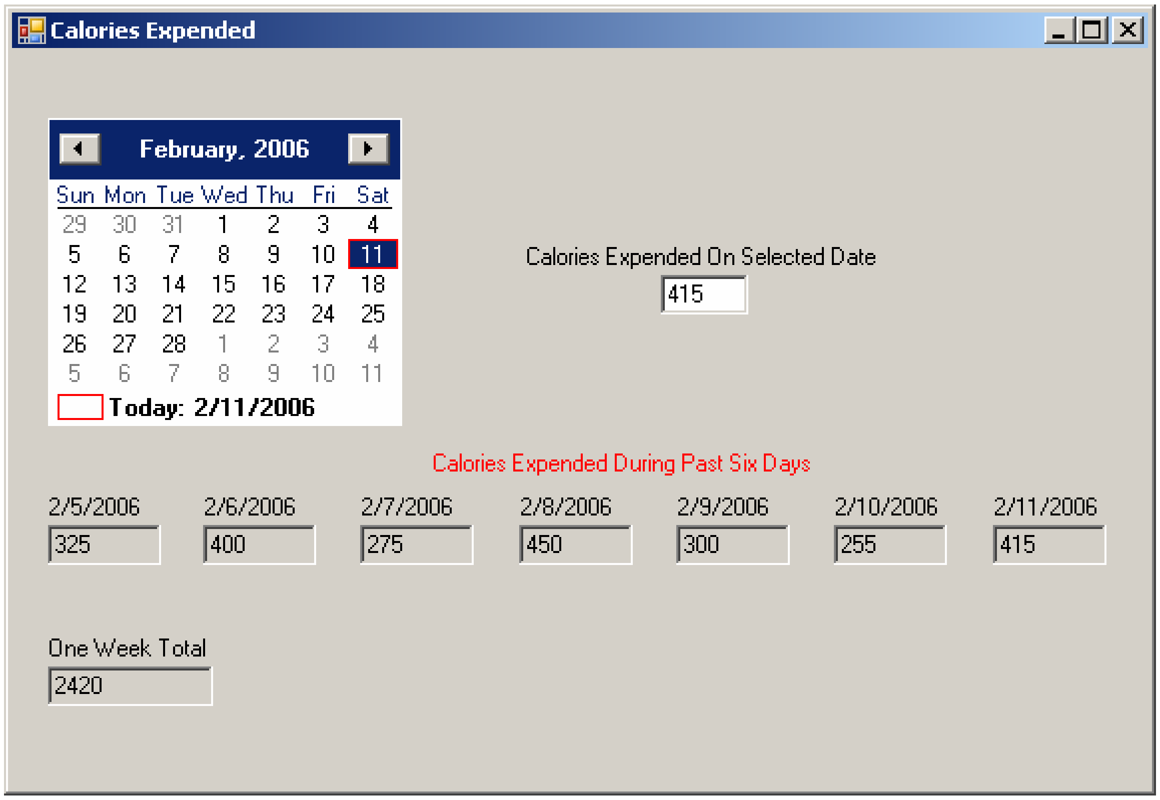
It can be difficult to design a first GUI-based student project that is reasonably challenging, realistic, fun and most importantly instructive. Such a project should deal with a problem that students can relate to in a problem domain that they all can easily understand.

My experience has shown that to be effective such a first project should involve a reasonable amount of "hand-holding" in which students are directed to perform a series of steps that introduces them to the construction tools and steps that help ensure their success and provides motivation and excitement about future projects that they will be expected to work on.

The project to be described in this paper is being presented to a group of 26 students in a course that deals with modern software development using C#/.NET(and using a book soon to be published with the same title "Modern Software Development Using C#/.NET" by Richard Wiener, 711 pages, Thompson Course Technology, April, 2006. The students are new to the C# language and the .NET framework but have prior experience with Java as their first programming language, studied in CS 1 and CS 2. Most of the students have little or no experience constructing a GUI application.

The technical issues that are dealt with in this first GUI project include: How to construct a GUI that contains textbox and label components that are programmatically controlled as well as the use of a powerful standard calendar component. Event-handling is used and illustrated for the calendar component, an "Enter" key entry in a textbox component, and a form-closing handler. Another major issue illustrated is the use of a generic collection class to hold the application data coupled with object persistence that enables the application to open and retrieve the application data and save this data before closing the application.

The application is specified as follows. We wish to construct a GUI application that allows a user to specify the number of calories expended using an exercise machine at some health club for any given selected date using a standard .NET calendar component. The GUI should show the number of calories expended for the date selected (current date by default) as well as the previous six days from either the current date or the date selected as well as the total number of calories for the seven days shown. The collection object used to store the historical data should be automatically loaded when the application begins and automatically saved when the application ends. A user interface is shown below that meets the above specifications. Each student is free to modify the layout of this interface, if desired, but not change the set of components that are used. When another date is selected, all the data should be automatically updated including the labels on top of the seven non-editable output boxes. This is shown in the second screen shot below where February 5th is selected.



Calories Expended On Selected Date
415

Calories Expended During Past Six Days

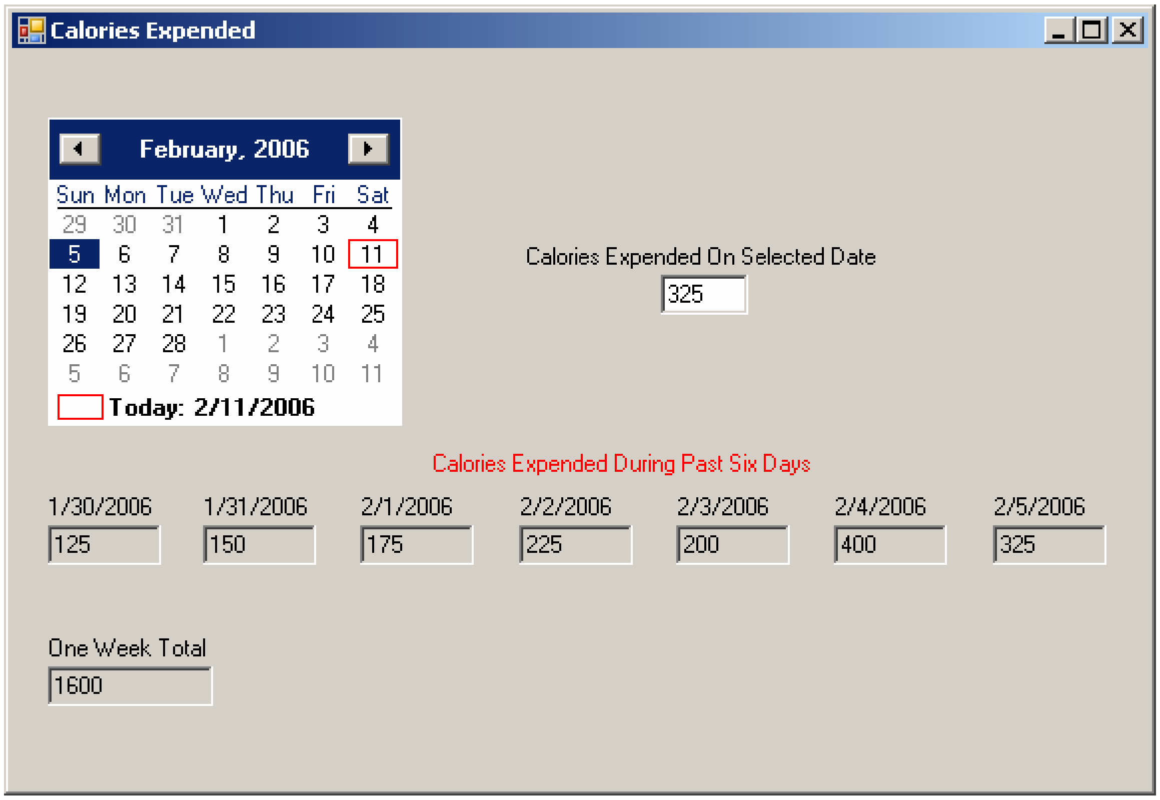
2/5/2006	2/6/2006	2/7/2006	2/8/2006	2/9/2006	2/10/2006	2/11/2006
325	400	275	450	300	255	415

One Week Total
2420

Some students might wish to layout the non-editable output text boxes that contain the current and past six days of data vertically rather than horizontally. Students are encouraged to experiment with different layouts using the one editable input text box and the eight non-editable output text boxes, the other labels and the calendar component.

This paper will be used by the students to assist them in constructing their first GUI project application. A significant amount of step-by-step help will be provided in the details to follow in the hopes that this will motivate experimentation on the part of the students and provide the foundation for future GUI project development in which students will be on their own. Feedback from students working on this project will be solicited to determine whether this approach to introducing GUI development in C#/.NET is effective or should be abandoned.

The remaining portion of this paper is aimed at students developing their first non-trivial GUI application and also educators and other readers who wish to eavesdrop on the details being provided for the students as they engage in their first C#/.NET GUI project.


Calories Expended

February, 2006

Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	1	2	3	4
5	6	7	8	9	10	11

Today: 2/11/2006

Calories Expended On Selected Date: 325

Calories Expended During Past Six Days

1/30/2006	1/31/2006	2/1/2006	2/2/2006	2/3/2006	2/4/2006	2/5/2006
125	150	175	225	200	400	325

One Week Total: 1600

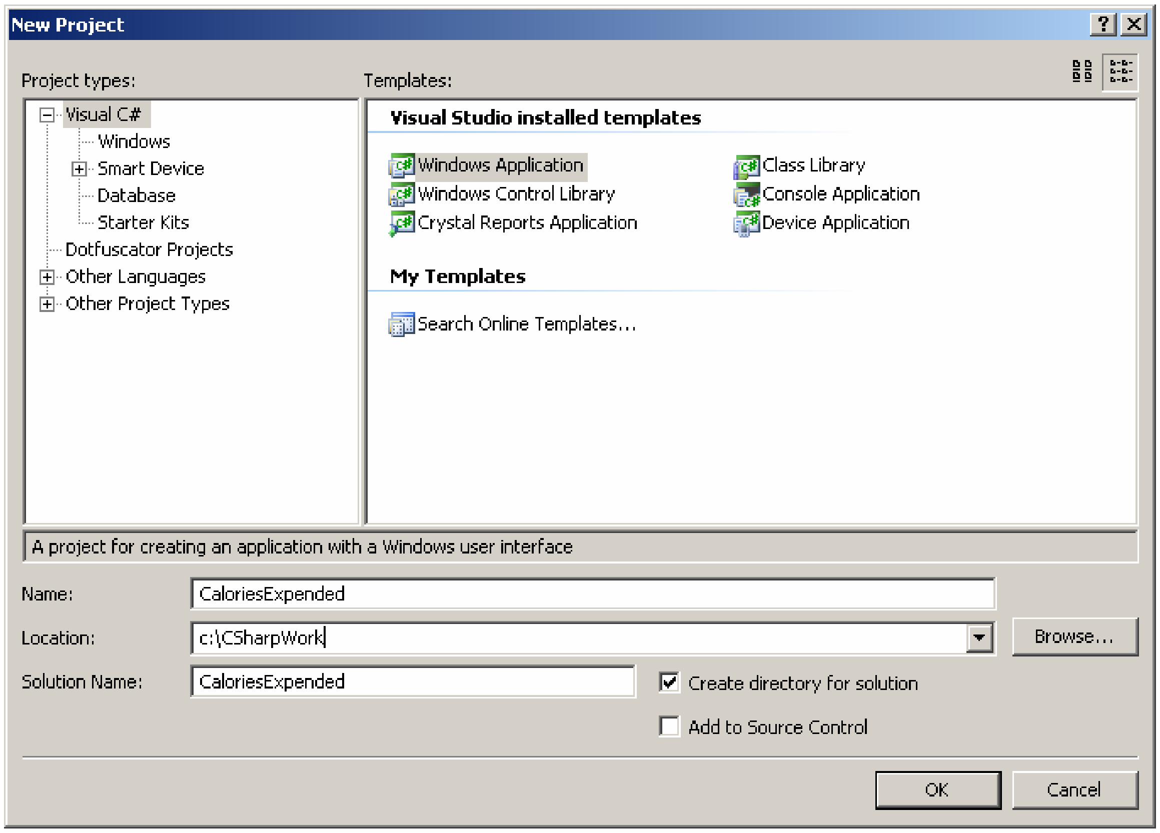
2 LET'S DO IT USING VISUAL STUDIO AND SOME C# PROGRAMMING

The first step is to create a GUI project using Visual Studio (Professional Edition used for this paper but the Express Edition works just as well).

The screen shot below shows how to get started. The name of the project, *CaloriesExpended*, will become the name of the executable file (in the \bin subdirectory). Visual Studio will create a subdirectory, *CaloriesExpended*, under *c:\CSharpWork\* using the entries shown below.

The blank design form shown below will be created by Visual Studio because a Windows Application was chosen. If you select View/Code menu item, the following partial class will be displayed (code generated by Visual Studio):

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace CaloriesExpended {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }
    }
}
```



A project for creating an application with a Windows user interface

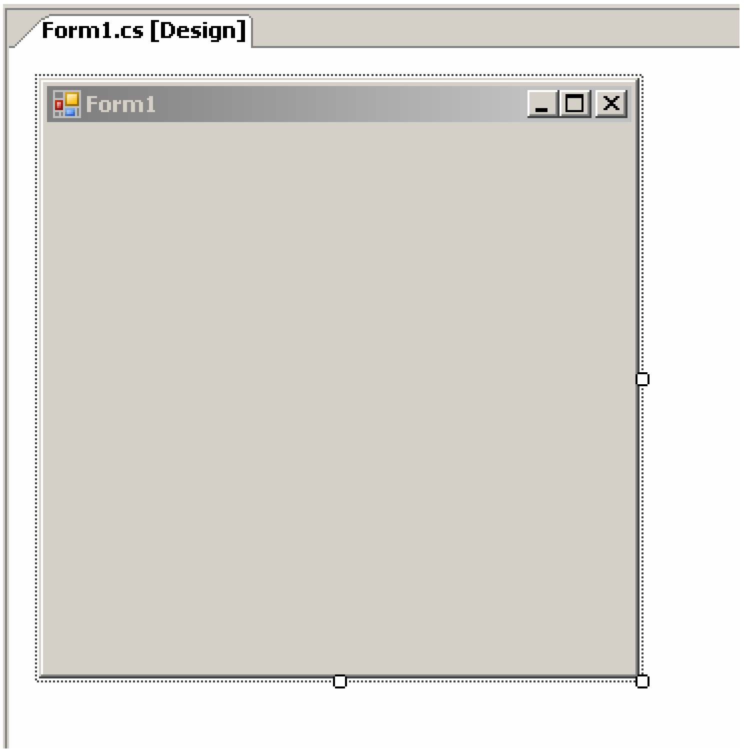
Name: CaloriesExpended

Location: c:\CSharpWork

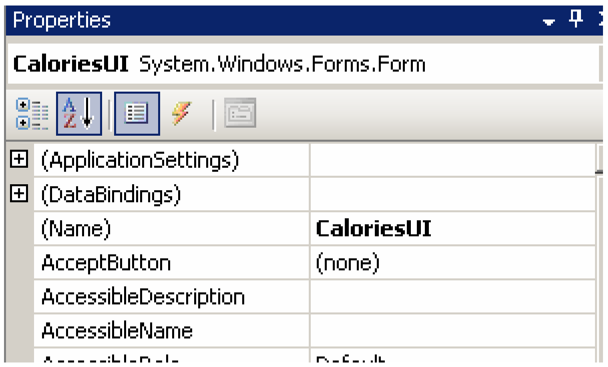
Solution Name: CaloriesExpended

[✓] Create directory for solution

[ ] Add to Source Control



The name of the *Form* component (*Form1*) should be changed to *CaloriesUI* using the Properties window as follows:

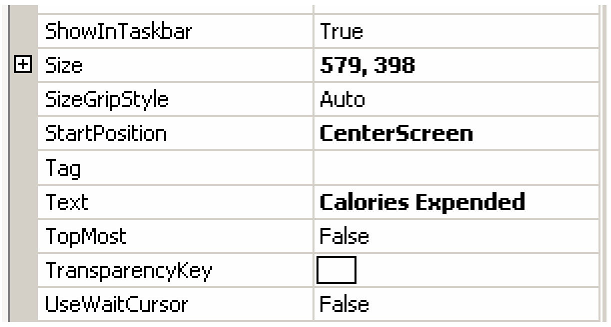


CaloriesUI System.Windows.Forms.Form

⊞ (ApplicationSettings)	
⊞ (DataBindings)	
(Name)	**CaloriesUI**
AcceptButton	(none)
AccessibleDescription	
AccessibleName	

The form is expanded using the mouse. Each student is free to size the form as desired. The size of the form created here is shown as 579 (width) x 398 (height) pixels.

Next the *StartPosition* property and *Text* property are modified as shown below.



ShowInTaskbar	True
⊞ Size	**579, 398**
SizeGripStyle	Auto
StartPosition	**CenterScreen**
Tag	
Text	**Calories Expended**
TopMost	False
TransparencyKey	☐
UseWaitCursor	False

Next using the Toolbox and dropping and dragging standard visual components onto the *CaloriesUI* form, the following user interface is produced:

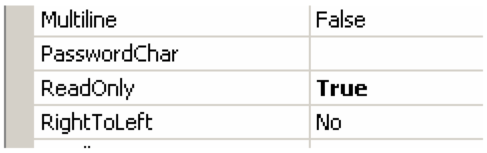
Each visual component should be given a meaningful name. The following table indicates the names used for the components shown above.

Visual Component	**Name Assigned**	**Description**
MonthCalendar	**monthCalendar**	Calendar component
TextBox	**caloriesTextBox**	Input for calories on selected date
TextBox	**sixDays**	Output textbox with label “6 days ago”
TextBox	**fiveDays**	Output textbox with label “5 days ago”
TextBox	**fourDays**	Output textbox with label “4 days ago”
TextBox	**threeDays**	Output textbox with label “3 days ago”
TextBox	**twoDays**	Output textbox with label “2 days ago”
TextBox	**oneDay**	Output textbox with label “1 day ago”
TextBox	**today**	Output textbox with label “Today”
TextBox	**pastWeek**	Output textbox with label “One Week Total”
Label	**sixDaysAgoLbl**	Name of label with text “6 days ago”
Label	**fiveDaysAgoLbl**	Name of label with text “5 days ago”
Label	**fourDaysAgoLbl**	Name of label with text “4 days ago”
Label	**threeDaysAgoLbl**	Name of label with text “3 days ago”
Label	**twoDaysAgoLbl**	Name of label with text “2 days ago”
Label	**oneDayAgoLbl**	Name of label with text “1 day ago”
Label	**todayDate**	Name of label with text “Today ago”

It is highly recommended that students working on this project use the same names for the visual components as those given in the table above in order that the code segments to be discussed below make sense.

It is important to provide meaningful names for the seven *Label* components because their *Text* property will be set each time a date is selected by the user or the default current date is used. Each of the output text boxes shows the calories expended for the date given by the label above the output text box.

The *ReadOnly* property of each of the output text boxes is set to true (from their default of false). Using the visual designer, all the output text boxes can be selected at once and then their *ReadOnly* property set to true.



Multiline	False
PasswordChar	
ReadOnly	**True**
RightToLeft	No

One of the most important features of a class is its fields. They represent the information model for the objects that are constructed from the class. There are three fields that are defined for this class. They are shown below:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

namespace CaloriesExpended {

    public partial class CaloriesUI : Form {

        // Fields
        private DateTime date =
            new DateTime(DateTime.Now.Year,
                         DateTime.Now.Month, DateTime.Now.Day);
        private DateTime selectedDate;
        private Dictionary<DateTime, int> data =
            new Dictionary<DateTime, int>();
```

The field *date*, of .NET type *DateTime*, captures the current year, month and day. It is initialized at its point of declaration to the current date using the *Now* property. The field *selectedDate*, also of type *DateTime*, is returned by the *MonthCalendar* component when the user selects a new date. Finally the field *data*, of type *Dictionary<DateTime, int>*, uses the generic *Dictionary* class with keys declared to be of type *DateTime* and values of type *int* (*System.Int32*). Each entry in *data* contains a key/value pair with the key given by a *DateTime* object and the value by the number of calories expended (represented by an *int*). This collection of key/value pairs stored in *data* will be saved to disk when the application exits and the form is closed and loaded from disk when the application starts.

Please note the two "using" namespaces that are added (*System.IO* and *System.Runtime.Serialization.Formatters.Binary*).

Portions of the constructor, *CaloriesUI* show how to use the *DateTime* class in the context of this application. Part of its implementation is:

```
public CaloriesUI() {
    InitializeComponent();
    // Initialize data labels with last seven days
    todayDate.Text = date.Month + "/" + date.Day + "/" +
                        date.Year;
    DateTime oneDayAgo = date.Subtract(new
                 TimeSpan(24 * 1, 0, 0));
    oneDayAgoLbl.Text = oneDayAgo.Month + "/" +
                 oneDayAgo.Day + "/" + oneDayAgo.Year;
    DateTime twoDaysAgo = date.Subtract(new
                 TimeSpan(24 * 2, 0, 0));
    twoDaysAgoLbl.Text = twoDaysAgo.Month + "/" +
                 twoDaysAgo.Day + "/" + twoDaysAgo.Year;
    // …
    sixDaysAgoLbl.Text = sixDaysAgo.Month + "/" +
                 sixDaysAgo.Day + "/" + sixDaysAgo.Year;

     selectedDate = date;

    // Open persistent data object
    FileStream calorieDataStream =
       new FileStream("Calorie.data",
                      FileMode.OpenOrCreate);
    try {
        BinaryFormatter bf = new BinaryFormatter();
        // Must restore data
        data = (Dictionary<DateTime, int>)
                bf.Deserialize(calorieDataStream);
        calorieDataStream.Close();
    } catch (Exception) {
        calorieDataStream.Close();
    }
    int oneWeekTotal = 0;
    today.Text = data.ContainsKey(selectedDate) ? "" +
                        data[selectedDate] : "";
    if (today.Text != "") {
        oneWeekTotal += Convert.ToInt32(today.Text);
    }
    for (int i = 1; i <= 6; i++) {
        DateTime newDate = selectedDate.Subtract(
                        new TimeSpan(24 * i, 0, 0));
        if (i == 1) {
            oneDay.Text = data.ContainsKey(newDate) ? "" +
                        data[newDate] : "";
            if (oneDay.Text != "") {
                oneWeekTotal += Convert.ToInt32(oneDay.Text);
            }
        } else if (i == 2) {
            twoDays.Text = data.ContainsKey(newDate) ? "" +
                data[newDate] : "";
            if (twoDays.Text != "") {
                oneWeekTotal += Convert.ToInt32(twoDays.Text);
            }
        } else if (i == 3) {
           // …
        }
        // …
        pastWeek.Text = "" + oneWeekTotal;
    }
}
```

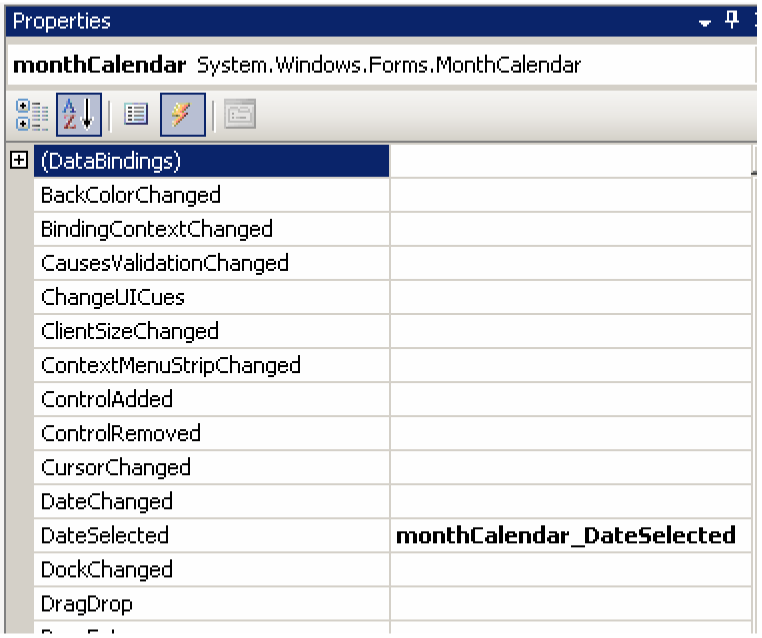
In this constructor many useful technical details are revealed. The properties *Month*, *Day* and *Year* are shown for a *DateTime* object. The *Subtract* query is shown in action along with an instance of the .NET class *TimeSpan*. The use of a *FileStream* and *BinaryFormatter* and the *Deserialize* method is used to retrieve a previously stored persistent object from a file "Calorie.data" that is stored in the application's *bin* subdirectory (by default) is shown.

The segment of code,

```
oneDay.Text = data.ContainsKey(newDate) ? "" +
                          data[newDate] : "";
if (oneDay.Text != "") {
   oneWeekTotal += Convert.ToInt32(oneDay.Text);
}
```

shows how the *Dictionary* object *data* is tested to see whether it contains the key *newDate*. If so the *Text* property of the output *TextBox oneDay* is assigned to "" + *data[newDate]*, if not the *Text* property is assigned to an empty string.

An event handler for the *MonthCalendar* component is created by selecting the monthCalendar component, selecting the event-handling properties using the lightning bolt tab and double clicking in the *DateSelected* row to produce the handler shown below:


Properties

monthCalendar System.Windows.Forms.MonthCalendar

(DataBindings)	
BackColorChanged	
BindingContextChanged	
CausesValidationChanged	
ChangeUICues	
ClientSizeChanged	
ContextMenuStripChanged	
ControlAdded	
ControlRemoved	
CursorChanged	
DateChanged	
DateSelected	monthCalendar_DateSelected
DockChanged	
DragDrop	

The skeletal structure of the *monthCalendar_DateSelected* handler is automatically generated by Visual Studio as follows:

```
private void monthCalendar_DateSelected(object sender,
                                        DateRangeEventArgs e) {
}
```

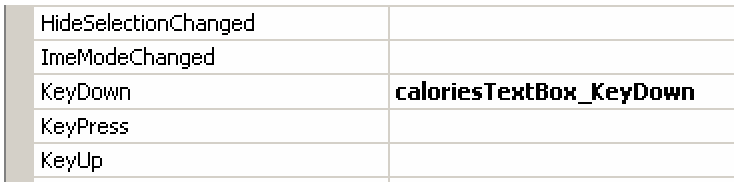
Whenever the user selects a new date using the *MonthCalendar* component, this event handler is triggered. Portions of this event handler are presented below:

```
private void monthCalendar_DateSelected(object sender,
                                  DateRangeEventArgs e) {
   selectedDate =
        new DateTime(e.Start.Year, e.Start.Month, e.Start.Day);
   if (data.ContainsKey(selectedDate)) {
       caloriesTextBox.Text = "" + data[selectedDate];
   } else {
        caloriesTextBox.Text = "";
   }
   todayDate.Text = selectedDate.Month + "/" +
                 selectedDate.Day + "/" + selectedDate.Year;
   DateTime oneDayAgo = selectedDate.Subtract(
                   new TimeSpan(24 * 1, 0, 0));
   oneDayAgoLbl.Text = oneDayAgo.Month + "/" + oneDayAgo.Day +
                   "/" + oneDayAgo.Year;
   // …
   this.ComputeAndFillAllOutputBoxes();
}
```

The private method *ComputeAndFillAllOutputBoxes()* uses the field *data* as well as the *selectedDate* field computed from the parameter, *e*, of type *DateRangeEventArgs*, as shown above, to populate all the output text boxes with information. Each student working on this project must write the complete implementation of this important method.

An event handler that is triggered when the user types the "Enter" key after providing input into the *TextBox* beneath the label "Calories Expended on Selected Date" must be provided.

After selecting this input *Textbox* component and clicking the lightning bolt tab (for event handlers), the *KeyDown* handler is double-clicked producing the handler shown below:



HideSelectionChanged	
ImeModeChanged	
KeyDown	**caloriesTextBox_KeyDown**
KeyPress	
KeyUp	

The skeletal structure of the *KeyDown* handler is automatically created by Visual Studio.

The completed event-handler is shown in its entirety below:

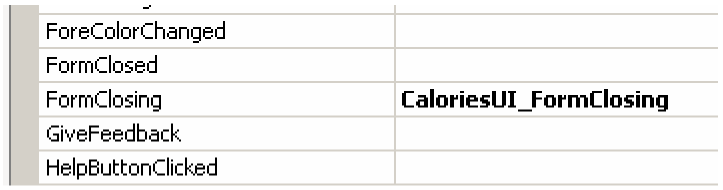
```
private void caloriesTextBox_KeyDown(object sender, KeyEventArgs
                                              e) {
   if (e.KeyCode == Keys.Enter) {
      try {
         String inputString = caloriesTextBox.Text.Trim();
         int inputValue = Convert.ToInt32(inputString);
         data[selectedDate] = inputValue;
         ComputeAndFillAllOutputBoxes();
      } catch (Exception) {
         MessageBox.Show( // Outputs a modal dialog box
             "Must enter an integer in the calories expended
                                              box.");
      }
   }
}
```

The *KeyEventArgs* parameter, *e*, and its *KeyCode* property is used to determine whether the value typed by the user equals *Keys.Enter*. If so, the input entered by the user is converted to an *int* and used to assign to the dictionary data using,

```
data[selectedDate] = inputValue;
```

The final event handler is associated with closing the form when the user clicks the "x" in the upper-right portion of the window containing the user interface.

If the *Form* object is selected and the tab moved again to the lightning bolt icon (event-handling properties), the *FormClosing* property must be double-clicked producing the handler shown below.



ForeColorChanged	
FormClosed	
FormClosing	**CaloriesUI_FormClosing**
GiveFeedback	
HelpButtonClicked	

The complete event-handler that shows how to write the *data* object (of type *Dictionary<DateTime, int>* to persistent storage is given.

```
private void CaloriesUI_FormClosing(object sender,
                                     FormClosingEventArgs e) {
   // Must save persistent data object
   FileStream calorieDataStream = new
            FileStream("Calorie.data", FileMode.OpenOrCreate);
   BinaryFormatter bf = new BinaryFormatter();
   bf.Serialize(calorieDataStream, data);
   calorieDataStream.Close();
}
```

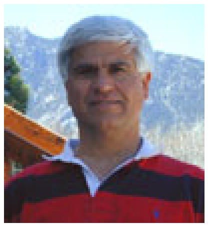
3 NOW DO IT

It is now the responsibility of students who have been assigned the task of implementing this application to do it.

In the next issue of JOT I will report how my students did on this first GUI project and whether the pedagogical approach suggested here has merit.

I welcome comments from other educators and students working on this project (rsw@runbox.com).

About the author



Richard Wiener is Associate Professor of Computer Science at the University of Colorado at Colorado Springs. He is also the Editor-in-Chief of JOT and former Editor-in-Chief of the Journal of Object Oriented Programming. In addition to University work, Dr. Wiener has authored or co-authored 21 books and works actively as a consultant and software contractor whenever the possibility arises. His latest book, to be published by Course Technology in late 2005, is entitled *Modern Software Development Using C#/.NET.*