Summary

The purpose of this application note is to introduce the reader to Microsoft Visual Studio and to utilize that program to create a usable Graphical User Interface (GUI). For this specific report we will go through examples of what was created for our Haptic User Interface project.

Keywords:

GUI, Microsoft Visual Studio, Interface, Haptic, Display
# Table of Contents

- Introduction.................................................................................................................. 3
- Getting Started................................................................................................................. 3
- Example GUI.................................................................................................................... 4
- Customizing a GUI............................................................................................................ 5
  - Adding Additional Buttons.......................................................................................... 5
  - Adding Additional Features......................................................................................... 6
- Coding a GUI.................................................................................................................... 7
- Haptic User Interface GUI............................................................................................... 10
  - Ports............................................................................................................................. 11
  - Uploading Images....................................................................................................... 12
  - Pin Display/Mouse...................................................................................................... 14
- Conclusion....................................................................................................................... 15
- Works Cited..................................................................................................................... 16
Introduction

Creating a Graphical User Interface allows a user to interact with electronic devices through the means of images rather than plain text commands. It provides a portal into the electronic and software world that serves as a tool for human interaction. Every single day on devices from personal cell phones to personal computers people use these portals to communicate to electronic software and hardware. Whether it is allowing a user to play a game, navigate an operating system, or even simply type a document, GUI’s are instrumental to making the use of electronics easier for the average user.

This application note will guide you through creating a GUI with Microsoft Visual Studio and demonstrate how it has been accomplished for the current Haptic User Interface (HUI) project.

Getting Started

The first step to creating a GUI is to decide which program you will use to code in. For this application note we are using Microsoft Visual Studio. This is already installed on every DECS computer in the College of Engineering at Michigan State University. You can access it by clicking Start-> All Programs ->Microsoft Visual Studio 2008. If you do not have access to this program it is available at the following link:

http://www.microsoft.com/visualstudio/eng/visual-studio-update

Visual Studio supports several different programming languages. For this application note we will be using the language C#. You will need to indicate this before starting a new project.

Prior to demonstrating the construction of the Haptic User Interface design we will go through a simple example to become familiar with how a GUI is created.
Example GUI

Our example GUI will feature several buttons that will obtain values from a PIC Microcontroller and some type of circuitry. This note is specifically about the GUI and not the hardware implementation, so we will focus entirely on how the code creates and affects a GUI and how you can customize the GUI.

First download the project from the following link:

http://www.egr.msu.edu/classes/ece480/capstone/ForMiniprojects/LAB4VB.zip

This will open a project called SampleSerial. Open the Visual Basic file. On the left window a project titled “SampleSerial” will be present toward the top and underneath that “Form1.vb” will be present. Double click “Form1.vb” to bring up the current GUI. Figure 1 shows the left window where you can find “Form1.vb” and Figure 2 shows the GUI that will show.

As you can see from the GUI, a sample button and output box has been coded for you already. To see this code, double click the “GetSerial” button. The code in this section is what is executed when you click that button while the program is running. Everything inside the GUI box that is shown above is what will be shown when a program runs. As you can see, you first create the aesthetic GUI by creating buttons and boxes or anything else, and then you program those buttons to provide functionality.

We will begin customizing this GUI to become familiar with the implementation.
Customizing a GUI

Adding Additional Buttons:

To add additional buttons to your current GUI right click the “GetSerial” button and select copy. Left click in grey space and then right-click->paste. A new button will form with the same name. Currently, no code exists for this button; however, double clicking the button will show that a new function for the button has automatically been created and is awaiting code implementation.

What if you don’t want to keep the name “GetSerial”? To edit the name of the button, right click and select Properties. This will open a box on the right with editing options. While plenty of options exist for you to edit ranging from button colors and style to font, we will focus solely on the name for now.

Figure 3 shows where you can edit a name. In the Appearance category beside the “Text” option there is the name “GetSerial”. Left click the “GetSerial” word and begin typing a new word. Figures 4 and 5 show examples of how editing the name and colors can create various customization options.

You can place new buttons anywhere on the interface you wish. The openness of Visual Studio allows you to completely customize your GUI to your needs and wishes. Simply click a button and drag it anywhere inside the interface to customize your GUI.

To add more output boxes follow the same format by copying existing boxes and pasting them in the interface.
Adding Additional Features:

Other features aside from buttons and output boxes exist in Visual Studio. We will briefly go over how to add a graph.

The first step to implement a graph is to install Dundas Chart tools. Click **Tools->Choose Toolbox Items**. Select **Browse** and add this file: DundasWinChart.dll. Ensure that a check is next to the name “Chart” and then click **OK**. Following this, click **View->Toolbox**. A “General” tab is present, and under this you should see “Dundas Chart Tools.” If you double click on “Chart” a box will appear and you can select your type of chart you wish to add. After selecting which type of chart you wish to add, press **Finish**. You have now successfully added a chart to your GUI and it is ready for programming.

![Figure 6: Example GUI](image)
Coding a GUI

After successfully creating and editing the visual GUI you now need to add functional code for each feature for it to execute precisely what it is you need it to accomplish. We will briefly go over example codes that buttons could use, what functions look like, and other simple coding practices.

In Figure 6 you will see several buttons that have various names. Each of these buttons needs to be added in the **Public Class** domain. Earlier we edited button names to appear as something different in the GUI; however, the button name for the coding portion was automatically assigned based on the functionality (ex. A new button will be named “Button2” even if you edited it to be called “Sample.”) While changing the name is not necessary (In your code “Button2” will always be assigned to the GUI “Sample” button), if you wish to change the code name follow the following example:

1. Right-Click on the button or feature you wish to edit and select **Properties**
2. Under the “Design” category you will see a (Name) option. On the right it will have a name like “Button2”. Left-click once on “Button2” (or whatever the name may be) and create your own name.

The benefit for editing the code name lies in organization of your code. For example, while we may always remember “Button2” = “Sample”, it could just be easier to rename the code name of the button to match the GUI name.

Now we are ready to edit the code of the GUI. Double-clicking a button will bring up the code of your GUI. Every button needs to be assigned a pre-existing function from the Visual Studio Library. For the following code examples we will be using the GUI from Figure 6. If it has not been done so already, edit your **Public Class Form1** to resemble Figure 7.

```vbnet
Public Class Form1
    Inherits System.Windows.Forms.Form
    Dim serialIn As New Rs232
    Dim start As DateTime
    Dim time As TimeSpan
    Friend WithEvents Textbox1 As System.Windows.Forms.TextBox
    Friend WithEvents Buttons As System.Windows.Forms.Button

    Friend WithEvents button1 As System.Windows.Forms.Button
    Dim sleeptime As Integer
    Friend WithEvents buttons As System.Windows.Forms.Button
    Dim sampleRate As Integer
    Declare Sub Sleep Lib "kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)
End Class
```

Figure 7
Following the Public Class domain is a portion of code generated by Visual Studio. Do not edit this code. Scroll down until you see “#End Region” and the start of the functions of your buttons. Each button has a function called a “Private Sub”. Following that the code name of the button and what the user does to the button is defined (Button2_Click) along with other code provided by Visual Studio. What you edit is between that first line defining the function and the last line – “End Sub”. Figure 8 shows an example.

![Code Snippet]

Figure 8: Edit the code where the red arrows appear above

We will provide an example of how to make functionality for a button and for a graph. During the following instruction keep in mind that a GUI is designed to interact with some type of device – for example, a microcontroller. We have not discussed how to bridge the connection between the GUI and the microcontroller as this application note focuses simply on creating the GUI. Because of this, there may exist functions or terms we have not discussed as they may be a part of, or being sent to, a microcontroller code.

Figures 9 and 10 have codes that are examples of implementing buttons:

```vbnet
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    serial.Text = "The button was clicked!"
End Sub
```

Figure 9: This button writes the letter “b” to a microcontroller and reads what is sent back by that microcontroller. It sends the result to a text box.

```vbnet
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    serial.Write("red")
End Sub
```

Figure 10: This button simply writes the word “red” to a microcontroller.

As you can see in the codes above, the buttons are designed to send something to another device. This device receives that information and performs certain operations depending on what it is programmed to do. For the first button, it receives some type of information back from that device and sends that data to the textbox function to be displayed in the GUI.
The following code demonstrates how a graph could be coded:

```vbnet
Private Sub GetGraph_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Chart1.Series("Graph").Click
    Chart1.Series("Graph").Points.Clear()
    Chart1.ChartAreas("Default").AxisX.Title = "Time (s)"
    Chart1.ChartAreas("Default").AxisY.Title = "Volts (V)"
    Dim x As Integer
    x = 0
    start = DateTime.Now
    For i = 1 To samplesrate
        gate31.Write("graph")
        serial_in = gate31.Read(2)
        m = MCB(CDSerial(serial_in))
        serial_in = gate31.Read(2)
        m = m + MCB(CDSerial(serial_in))
        Sleep(2000)
        time = DateTime.Now.Subtract(start)
        Chart1.Series("Graph").Points.AddXY(time.TotalSeconds, x * 5 / 1023)
        Sleep(sleptime)
    Next
    Chart1.ChartAreas("Default").AxisY.Minimum = 0
    Chart1.ChartAreas("Default").AxisY.Maximum = 0
    Chart1.ChartAreas("Default").AxisX.Minimum = 0
    Chart1.ChartAreas("Default").AxisX.Maximum = time.TotalSeconds
    End Sub
```

From this code you can see that we create everything on the graph from axis titles to data to be displayed. You can also see that the word “graph” is sent to a foreign device and then that device sends back information that this code interprets and displays in the GUI. An example of what could result from this code being implemented is shown in Figure 12.

![Sample Serial GUI](image)

Figure 12: Notice the axis intervals, data, and titles

Now that we have gone over the basics of creating a GUI through Visual Studio we will briefly summarize how these tools have been applied to create a GUI for the Haptic User Interface project.
Haptic User Interface GUI

Building upon the example from above we are able to create a specific GUI to fit our needs for the project. We will explore the aesthetics of the GUI and a few examples of code to demonstrate how it operates.

Since we have already demonstrated how to add buttons and other features and customize those features through various options, we will simply show our final GUI for the project in its completed form.

![Completed GUI for the HUI Project](image)

Aside from standard buttons that are positioned along the bottom-left and top-middle of the GUI, there are some other features included here. The large grey box in the middle is the portion of the GUI that will show the image that has been uploaded. The white box shows every image’s file name that has been uploaded and you can scroll through them. The top features a drop-down menu for Port options. This exists for the GUI to interact with a device.

Each of these features has been fully programmed to function properly. We will now summarize each function and its code.
Ports:

The top-left portion of the GUI features a drop-down menu called “Ports”. What this does is establish a connection between the GUI and a device. The GUI automatically determines if a port is available and selects the top option by default. By selecting the down arrow you can select more ports if they are available. Clicking the “Open Port” button opens the port for communication to the device through the selected option. Figure 14 demonstrates through some code how this is accomplished and Figure 15 demonstrates the use of the drop-down menu.

```csharp
// list all the ports available in the system
foreach (string port in SerialPort.GetPortNames())
{
    // add each port name to the ports list
    cboPorts.Items.Add(port);
}

private void btnOpen_Click(object sender, EventArgs e)
{
    // use the port specified in drop list
    _port = new SerialPort(cboPorts.Text);
    // set the default values of the port
    _port.BaudRate = 9600;
    _port.Databits = 8;
    _port.StopBits = StopBits.One;
    _port.Parity = Parity.None;
    _port.ReadTimeout = 5000;
    // "turn off" all pins
    _port.DTR = false;
    _port.RTS = false;
    _port.CTS = false;
    _port.GX = false;
    // change the enable property for buttons in order to prevent the user from "reopening" the button
    cboPorts.Enabled = false;
    btnOpen.Enabled = false;
    btnClose.Enabled = false;
}
```

Figure 14: (a) Shows the user every port that is available; (b) Selects the first port available as default if multiple ports exist, and outputs an error message if no ports are detected; (c) Upon clicking the “Open Port” button this code is executed; (d) Upon clicking the “Close Ports” button this code is executed.

```csharp
private void btnClose_Click(object sender, EventArgs e)
{
    // "turn off" pins
    ResetPins();
    // close port
    _port.Close();
    // allow user to reopen port
    cboPorts.Enabled = true;
    btnOpen.Enabled = true;
    btnClose.Enabled = false;
}
```

Figure 15
Uploading Images:

In the bottom-left of the GUI a button exists called “Add Picture”. This button is programmed to add images and, through a different portion of the code, convert that image to greyscale. Upon clicking the button a display window pops up prompting the user to select an image of predetermined formats. Once the image has been selected, converted to greyscale, and loaded, it will display in the box on the right. Figure 16 shows through the code how this is accomplished and Figure 17 shows the pop up window that prompts the user to select an image.

Figure 16: Upon clicking the button the user is prompted to select a file that only ends in the appropriate file format. Following that, the program attempts to add the file to the list. If it is successful, the image is loaded in the GUI. If it is not successful, an error message displays.

Figure 17: This menu pops up after the “Add Picture” button is pressed
Upon adding the image to the list and converting the image to greyscale, you can add more images if you wish. Doing so creates a list in the white box on the left that you can navigate through by clicking the “Prev” and “Next” buttons. This is demonstrated in Figures 18 and 19.

```csharp
private void btnNext_Click(object sender, EventArgs e)
{
    // if we reached the end of the list, then loop around
    if (lstImages.SelectedIndex == lstImages.Items.Count - 1)
    {
        lstImages.SelectedIndex = 0;
    // otherwise, select the next item
    }
    else
    {
        lstImages.SelectedIndex++;  
    }
}
private void btnPrev_Click(object sender, EventArgs e)
{
    // if we reached the beginning of the list, then loop around
    if (lstImages.SelectedIndex == 0)
    {
        lstImages.SelectedIndex = lstImages.Items.Count - 1;
    // otherwise, select previous item
    }
    else
    {
        lstImages.SelectedIndex--;  
    }
}
```

Figure 18

Figure 19: After loading multiple images a list appears on the left. Clicking “Next” or “Prev” cycles through these images.
Pin Display/Mouse:

Our project uses the mouse to “feel” an image. When the pointer of a mouse hovers of a dark portion of an image, the GUI sends a signal to a device to raise pins to signify that this is part of the image that should be felt by the user. The “Training Mode” button in the top-middle of the GUI provides a great example of what this should look like on a physical device. Upon clicking the button, a new window pops up with rows of blocks that simulate pins. When the mouse cursor hovers over the image and moves around the image, the pins change color representing which pins would be raised on a physical device. Figures 20 and 21 demonstrate this through code and visuals, respectively.

```java
private void guiDisplay_MouseMove(object sender, MouseEventArgs e)
{
    // make sure there is an image loaded in the picture box
    if (this.Image.Image == null)
    {
        return; // if no image then no GUI
    }
    // make sure there is a gray value stored
    grayMap = new byte[256]; // copy the gray scale picture to memory
    int x = 0; // store the x location of the pixel
    int y = 0; // store the y location of the pixel
    // loop through each module row
    for (int moduleX = 0; moduleX < NUM_MODULES; moduleX++)
    {
        //for each module row, loop through each module column
        for (int moduleY = 0; moduleY < NUM_MODULESgetColumn(); moduleY++)
        {
            // reset the gray index to which will send out “row” data
            grayIndex = 0;
            // for each module row, loop through each of its pin rows
            for (int pinRow = 0; pinRow < NUM_PINS; pinRow++)
            {
                // for each pin row, loop through each pin column
                for (int pinColumn = 0; pinColumn < NUM_PINS; pinColumn++)
                {
                    // attempt to read pixel at mouse location
                    // figure out pixel location depending on which module we are reading
                    // and which pin in that module we are reading
                    if (pinRow = 0 && pinColumn = 0)
                    {
                        // store the gray value of that pixel
                        grayValues[pinIndex] = 255 - grayValue;
                        // reset pins when mouse leaves picture
                        ResetPins();
                    }
                }
            }
        }
    }
}
```

(a)

```java
private void guiDisplay_MouseLeave(object sender, MouseEventArgs e)
{
    // reset pins when mouse leaves picture
    ResetPins();
}
```

(b)

```java
private void guiDisplay_MouseLeave(object sender, MouseEventArgs e)
{
    // reset pins when mouse leaves picture
    ResetPins();
}
```

(c)

Figure 20: (a) Ensures an image is loaded, and then assigns pixel values to the image. It loops through each row and column and attempts to read the pixel location. (b) Sends a “white” value to the pins, which turns them on. (c) If the mouse leaves the display window it resets the pins, turning them off.
Conclusion

This application note summarized how to create a Graphical User Interface with Microsoft Visual Studio. It started by going over a few basic generic examples and then concluded with discussing Design Team 6’s project for Spring Semester 2013.

Without Graphical User Interfaces interacting with an electronic device would not be nearly as easy or enjoyable. Of course, GUI’s have not existed forever, and using a computer or other device is possible without one, but they make the use of electronic devices very simple and add on functionality to these machines. When designing a GUI it is important to keep in mind the following:

- Simple is often better – do not try to overdo a design. The simpler the “feel” of the GUI, the easier it is for a user to utilize.
- Code as early and often as possible. Errors are bound to pop up and it is imperative you as a designer to start early for projects with deadlines.
- You want a GUI that is not only functional and does what is intended to do, but is aesthetically pleasing. Design is as much a science as it is an art.
- When coding, ensure to practice standard coding procedures – comment code, name functions and variables appropriately, and utilize libraries.
Works Cited

http://en.wikipedia.org/wiki/Microsoft_Visual_Studio

http://www.egr.msu.edu/classes/ece480/capstone/ForMiniprojects/Lab4.pdf

http://en.wikipedia.org/wiki/Graphical_user_interface