# Using Microsoft Visual Studio to Create a Graphical User Interface

ECE 480: Design Team 11

**Application Note** 

Joshua Folks

April 3, 2015



#### Abstract:

Software Application programming involves the concept of human-computer interaction and in this area of the program, a graphical user interface is very important. Visual widgets such as checkboxes and buttons are used to manipulate information to simulate interactions with the program. A well-designed GUI gives a flexible structure where the interface is independent from, but directly connected to the application functionality. This quality is directly proportional to the user friendliness of the application. This note will briefly explain how to properly create a Graphical User Interface (GUI) while ensuring that the user friendliness and the functionality of the application are maintained at a high standard.

# **Table of Contents**

Abstract	1
Introduction	3
Operation	3
Operation	3
Visual Studio Methods	4
Interface Types	6
Understanding Variables	7
Final Forms	_
Final Forms	/
Conclusion	8

Key Words: Interface, GUI, IDE

#### Introduction:

Establishing a connection between the user interface and the application functionality has been simply implemented since the late 1970's. Xerox PARC has been a major contributor to user interfaces including the first ever, Alto, where most general purpose GUI's are derived from. The 1980's were revolutionary for GUI's across all platforms. The Xerox Star and the Apple Lisa both introduced major functionality from the menu bar, window controls, double-clickable icons, to dialog boxes, and monochrome displays. The evolution and popularity of the user interface continued to grow beyond initial thoughts. This increase of its use made it impossible to create a standard look for GUI's. The wide range of operating systems with very distinct GUI's are among the most popular and most recognizable. Microsoft Windows and Mac OS X are huge for laptop/desktop computers and Android, Apple iOS, and Windows Phone are the notable GUI's for handheld devices. The creation of a user-friendly interface does not involve a complex series of operations in today's age. Different programs such as Microsoft Visual Studio make it very simple to get a GUI with high application functionality and are very attractive to users across the board.

# **Objective:**

This note will discuss the methods and different types of interfaces that Microsoft Visual Studio provides in the IDE. Visual Studio is able to pre generate code to better understand how the user interface is connected to the functionality of the program. The same functionality can be achieved by different complex methods and many simple methods that are available. Linking menu bars or tools that are thoroughly placed on the form to the code is very important to a successful GUI.

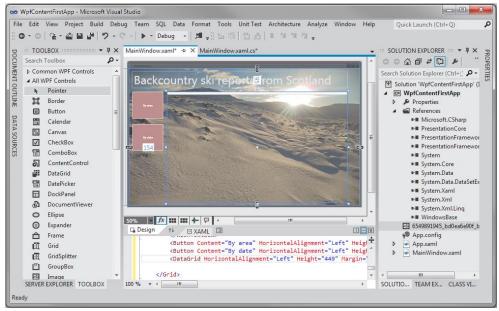


Figure 1: General Pictures showing Visual Studio's IDE

A large part of working with the user interface involves understanding the code that joins everything together. Since Visual Studio pre generates this code, they give an upper edge on the amount of practices to put in the application by adding simplicity. After all of the design and functionality is completed, the .exe file will be automatically generated to run the program inside and outside of Visual Studio. This note will give examples of final forms and give a more clear view of what is possible with Visual Studio.

#### **Operation:**

Microsoft Visual Studio has an advanced variety of tools to create a high level performance GUI's. This integrated development environment (IDE) is compatible with many different languages, but remains simple to maintain the functionality of the application. Visual Studio has a large assortment of options to create a GUI. As shown in Figure 1, depending on the language that was chosen, there are lists of different projects that can be selected such as a MFC Application, Windows Console Application, and Windows Forms Application. Each one of these different types of applications can be used for a common interface. Still they each have their own special attributes, whether they differ from the code that they generate or the type of methods they used to create the actual form.

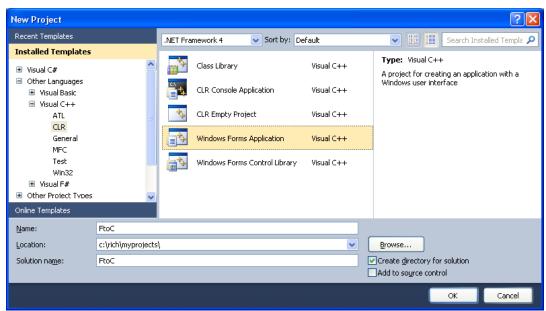


Figure 2: New Project showing various Windows Applications

The Windows Forms Application is selected as the base for this note. The empty form for the Windows Application will appear and this will serve as a foundation for the GUI as seen in Figure 2. All types of GUI's consist of basic tools including: buttons, text boxes, labels, check boxes, etc. The more intuitive controls that the GUI handles care able to correlate with the functionality of the application. However, more controls can decrease the level of user friendliness of the actual GUI.

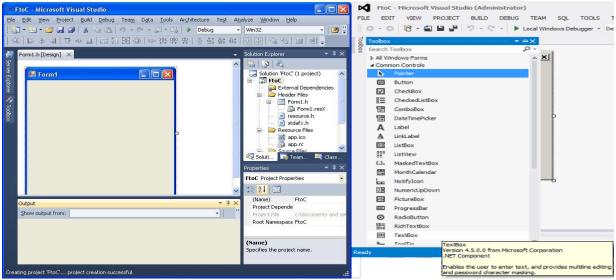


Figure 3: Initial Form that is generated as well as the common tool for GUI's

#### **Visual Studio Methods:**

A key feature that Visual Studio implements in their IDE is the drag/drop method. This method allows the user to pick and choose any tool desired to be a part of the final GUI and drag and drop the tool on to the windows form. With this method the code that runs in parallel with the tools is pre-generated. This takes away the complex and time-consuming task of matching code with each tool. Since there can be a vast number of different and similar controls on the GUI, there is a simple naming convention that is used when using Visual Studio. Each tool placed on the form has a preference window that deciphers what the tool is named, what happens when that tool is clicked, and many other layout settings. This allows the user to organize the GUI more effectively than having tools on the GUI that one cannot decode the uses of in the code. The code behind the GUI is very important since it tells the application the functionality of the program. As previously stated, the code is pre generated in a format that is understandable to even the novice programmer. Double clicking on any tool on the GUI will result in generated code that will trigger a function. Figure 3 shows the function that is produced with the parameters.

Figure 4: Generated Function that correlates to tool on GUI

This section of the code handles an important piece of the interface; it directly connects the user interface to the functionality of the program. This function handles what happens when something on the form is clicked. In this specific case, Figure 3 shows the function that will be referenced once the first button is clicked. All further actions the user wants to implements after the button is clicked will be placed within? this function. There are no limits placed on the process the form should enter once a tool is clicked. Visual Studio supports the ability to create

different dialog boxes from a form or a different dialog box, the ability to make calculations, and creating different forms all while maintaining the true performance of the GUI.

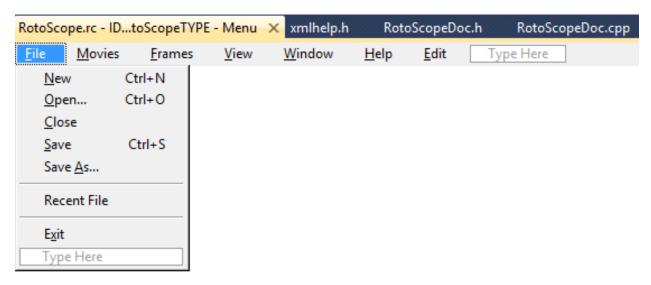


Figure 5: Type of Interface Supported by Visual Studio

# **Interface Types:**

Creativity is highly supported within Visual Studio. When making an intricate GUI, simplicity is a key concept when considering the design. Rather then having multiple buttons and methods directly on the form within the GUI, it is better to design a menu bar to have multiple options and less clutter. Figure 4 shows how this can be implemented. With each menu item there is a menu option that goes into the dropdown tab. These menu options are clickable in the interface and they each have an ID tied to them. In order to gain the interactive part with the options an event handler can be paired with a certain class. This handler will then be referenced to whenever it is clicked in the running of the program. This type of interface is widely used in the world of GUI's.

### **Understanding Variables:**

Visual Studio is a remarkable IDE that not only pre generates the functions that coordinate with clicking in the interface, but also generates the variables that are needed for the program. The structure of the program is formatted in a way where referencing variables on the form done by pointing to the form and gaining access to variables. In order to do this, the user would have to reference the pointer to the form in the form. This process seems complicated, but it is simply done by using *this* followed by a -> sign. This takes the pointer of *this* that is currently pointing to the form and then is able to access its items such as objects and variables. Depending on which variables or objects are accessed there are many ways to actually reference the text in the text box or if a check box has been checked for example. Figure 5

shows how the code references the variables in the form. The layout, naming, location, and index are all set by this method that Visual Studio implements.

```
Form1.h* X Form1.h [Design]*
test::Form1
                       CCACDONT (Bellett Systems 111211001131
                 this->label1 = (gcnew System::Windows::Forms::Label());
                 this->button1 = (gcnew System::Windows::Forms::Button());
                 this->SuspendLayout();
                 //
                 // checkBox1
                 //
                 this->checkBox1->AutoSize = true;
                 this->checkBox1->Location = System::Drawing::Point(30, 64);
                 this->checkBox1->Name = L"checkBox1";
                 this->checkBox1->Size = System::Drawing::Size(80, 17);
                 this->checkBox1->TabIndex = 0:
                 this->checkBox1->Text = L"checkBox1";
                 this->checkBox1->UseVisualStyleBackColor = true;
                 //
                 // textBox1
                 this->textBox1->Location = System::Drawing::Point(68, 87);
                 this->textBox1->Name = L"textBox1";
                 this->textBox1->Size = System::Drawing::Size(100, 20);
                 this->textBox1->TabIndex = 1;
                 //
                 // label1
                 //
                 this->label1->AutoSize = true;
                 this->label1->Location = System::Drawing::Point(27, 87);
                 this->label1->Name = L"label1";
                 this->label1->Size = System::Drawing::Size(35, 13);
                 this->label1->TabIndex = 2;
                 this->label1->Text = L"label1";
                 //
                 // button1
                 //
                 this->button1->Location = System::Drawing::Point(188, 214);
100 %
```

Figure 6: Displaying variables reference in code from the form

# **Final Forms:**

A user interface is required to have a purpose beyond the attractive graphics. Being able to add text boxes that react upon different booleans of checked boxes will ensure the right steps that the user needs to know for interactivity. Figure 6 shows a few different final forms of GUI's that have been created by Visual Studio.

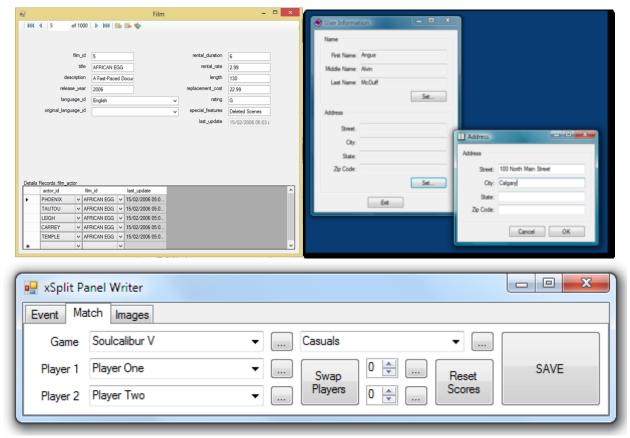


Figure 7: Different GUI's that are created used Visual Studio

The logic behind the entire interface goes hand and hand with what appears on the interface. However, there will be different if and while statements that can be used to improve the overall performance. For example there could be multiple if statements that would disable text boxes depending on radio buttons or while loops that could continue provided the amount of time per loop from the user. If your application involves drop down menus, they can be directly correlated to the code and improve the user friendliness by minimizing how many actions the user has to make in order to increase the ease of use. The application functionality dictates the user interface and once that is decided then the interactivity should be simplified.

#### **Conclusion:**

Microsoft Visual Studio has simple, yet complex methods that are used to create a graphical user interface that has its ease of use range from easy to hard. With the pre-generated code, variables from the form, and different methods to have a less cluttered interfaces, the user can achieve a great amount of application functionality. The final form of the application must include user friendliness and the purpose of the program proportionally. Design Team 11 has decided to use Microsoft Visual Studio to program the GUI that interacts with ReplicatorG. This program will serve as an editor that will change the Gcode produced by ReplicatorG, to update

it with the current settings. User interfaces are a key attribute when producing applications that serves a purpose to the user with the ease to understand the function and application.

# **References:**

- <a href="http://apievangelist.com/2014/08/19/bing-developer-assistant-for-visual-studio-delivers-relevant-api-code/">http://apievangelist.com/2014/08/19/bing-developer-assistant-for-visual-studio-delivers-relevant-api-code/</a>
- http://toastytech.com/guis/guitimeline.html
- <a href="http://holowczak.com/getting-started-with-windows-forms-using-visual-c-tutorial/4/">http://holowczak.com/getting-started-with-windows-forms-using-visual-c-tutorial/4/</a>
- <a href="http://stackoverflow.com/questions/9213839/visual-studio-c-sharp-windows-forms-changing-button-color">http://stackoverflow.com/questions/9213839/visual-studio-c-sharp-windows-forms-changing-button-color</a>
- <a href="http://www.codeproject.com/Articles/17371/Passing-Data-between-Windows-Forms">http://www.codeproject.com/Articles/17371/Passing-Data-between-Windows-Forms</a>
- <a href="http://dev.mysql.com/doc/connector-net/en/connector-net-visual-studio-forms-project-wizard-usage.html">http://dev.mysql.com/doc/connector-net/en/connector-net-visual-studio-forms-project-wizard-usage.html</a>