MEDIS: A Methodology for the Formation of Highly Qualified Engineers at Masters Level in the Design and Development of Advanced Industrial Informatics Systems

WP2.1 Chapter 6.1: Graphic User Interface (GUI) Part I

Co-funded by the Tempus Programme of the European Union

544490-TEMPUS-1-2013-1-ES-TEMPUS-JPCR

Authors: Capella, J.V. Perles, A., Albaladejo, J

MEDIS: A Methodology for the Formation of Highly Qualified Engineers at Masters Level in the Design and Development of Advanced Industrial Informatics Systems

WP2.1 Chapter 6.1: Graphic User Interface (GUI) Part I

Contract Number: 544490-TEMPUS-1-2013-1-ES-TEMPUS-JPCR

Starting date: 01/12/2013 **Ending date:** 30/11/2016

Deliverable Number: 2.1

Title of the Deliverable: AIISM teaching resources - Industrial Computers

Task/WP related to the Deliverable: Development of the AIISM teaching resources - Industrial Computers

Type (Internal or Restricted or Public): Public

Author(s): Capella, J.V. Perles, A., Albaladejo, J

Contractual Date of Delivery to the CEC: 30/09/2014

Actual Date of Delivery to the CEC: 30/09/2014

Project Co-ordinator

Company name :	Universitat Politecnica de Valencia (UPV)
Name of representative :	Houcine Hassan
Address :	Camino de Vera, s/n. 46022-Valencia (Spain)
Phone number :	+34 96 387 7578
Fax number :	+34 963877579
E-mail :	husein@upv.es
Project WEB site address :	https://www.medis-tempus.eu

Context

WP 2	Design of the AIISM-PBL methodology
WPLeader	Universitat Politècnica deValència (UPV)
Task 2.1	Development of the AIISM teaching resources - Industrial Computers
Task Leader	UPV
Dependencies	MDU, TUSofia, USTUTT, UP

Author(s)	Capella, J.V. Perles, A., Albaladejo, J
Reviewer(s)	Hassan, H., Martínez, J.M., Domínguez, C.

History

Version	Date	Author	Comments
0.1	01/03/2014	UPV Team	Initial draft
1.0	19/09/2014	UPV Team	Final version

Table of Contents

1 Executive summary

WP 2.1 details the learning materials of the Advanced Industrial Informatics Specialization Modules (AIISM) related to the Industrial Computers Module.

The contents of this package follows the guidelines presented in the UPV's documentation of the WP 1 (Industrial Computers Module)

- The PBL methodology was presented in WP 1.1
- The list of the module's chapters and the temporal scheduling in WP 1.2
- The required human and material resources in WP 1.3
- The evaluation in WP 1.4

During the development of this WP a separate document has been created for each of the chapters of the Industrial Computers Module (list of chapters in WP1.2).

In each of these documents, section 2 introduces the chapter; sections 3, 4, 5 and 6 details the Lecture, Laboratory, Seminar and Mini-project of the chapter; section 7 lists the bibliography and the references.

2 Introduction

The chapter #6 is dedicated to the graphical user interface (GUI). Effective and usable user interface and interaction design is essential for desktop and mobile devices. And it's critical for a growing number of industrial platforms and applications. In recent years, we have seen the rapid proliferation of products with integrated digital user interfaces. A new product or application without an attractive and usable interface seems behind the times.

3 Lecture

3.1 Objectives

- To understand the concept of graphical user interface (GUI) and basic related concepts.
- To know the GUI development philosophy aspects.
- To know the possibilities of a GUI design environment and how to use it.
- To be able to design an intuitive and accurate graphical user interface.

3.2 Introduction

A great evolution has taken place from legacy dashboards and control panels to modern graphical, intuitive and tactile or motion based user interfaces (see figure 1).

User interface design is a fast-moving discipline, with new types emerging as new technologies become available for consumer and commercial products. Graphical user interface (GUI) refers to the graphical elements on a two-dimensional screen; and physical user interface to the operable parts of a technology-embedded object. There is also natural user interface and gestural user interface, discussed later. The goal of all of the above is to make the interaction between a person and a product as easy and intuitive as possible

Good user interface design is driven by creating an easy to use product. Designing usable products is not easy, particularly in consumer applications where the range of user characteristics (e.g., physical capabilities, usage patterns, technical skill) is so broad and versatile. In these cases the GUI design needs to be intuitive and still provide the right range of features.

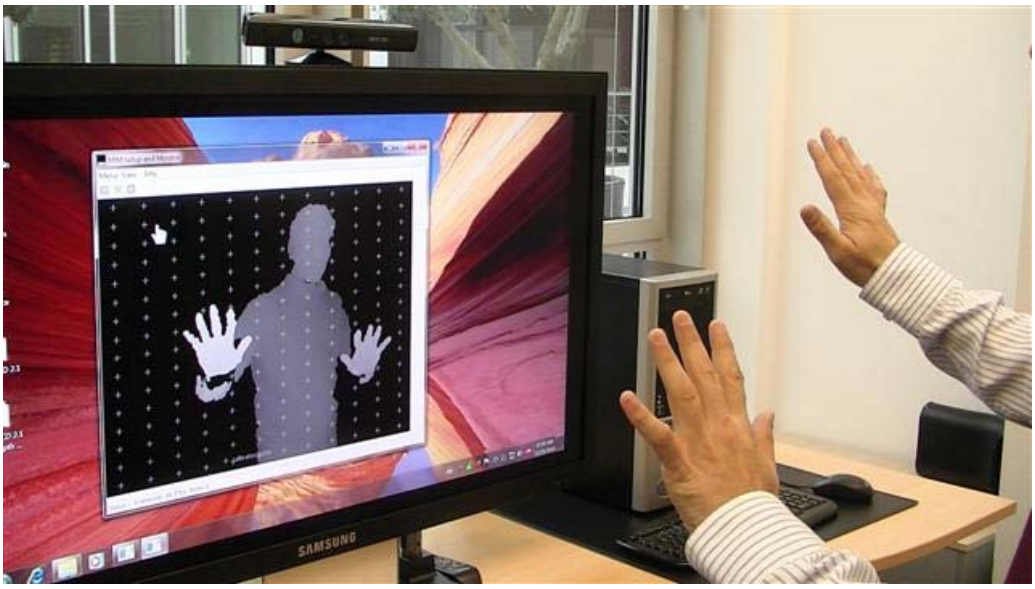


Figure 1: Current trends in user interfaces

3.3 Design concepts

As we saw in previous chapters, any software implementation of an industrial computer system can be divided into different separate modules, each of which implements a particular aspect of the system. Among these, we can highlight the central core of the program, which imbricates all logical system entities and their possible relationships. Another is the module of graphical user interface or GUI, which includes all the software responsible for representing the user information, such as consisting of windows, menus, buttons, tabs, etc. These two modules, as well as others, need to communicate with them, so that the GUI should provide the user the information from the system core, and the core must be updated with the information provided by the user. Then, a good design should identify and specify either all modules as separate units, so:

- Each of these modules must be specified, designed, implemented, tested, and verified independently.
- These modules must be maintained and expanded each independently of the other.
- Can be reused in other systems.
- The GUI module must be able to be exchanged for another one (eg, when new technological advances were available).

When small systems are developed is easy to mix code between GUI and core modules. This must be avoided due to the above reasons, in order to make possible to reuse and exchange the modules.

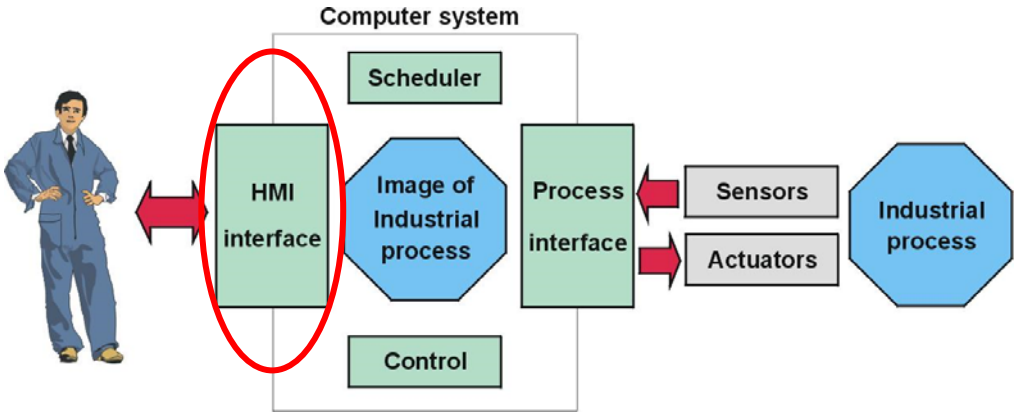


Figure 2: The GUI module in relation to the other modules

3.4 Basic concepts about graphical representation

To understand several concepts related with the GUI, you need some background in computer graphical representation. First concept is the pixel that stands for picture element. The resolution is the number of pixels displayed in one screen and a graphics card (figure 3) is a device equipped with an electronic circuitry capable of display on screen textual and graphical information (see figure 3).

Figure 3: A graphics card

3.5 Pixels and coordinates

Locations in computer graphics are stored as mathematical coordinates, but the display surface of an output device is an actual physical object. Thus, it's important to keep in mind the distinction between physical pixels and logical pixels.

3.5.1 Physical pixels

Physical pixels are the actual dots displayed on an output device. Each one takes up a small amount of space on the surface of the device. Physical pixels are manipulated directly by the display hardware and form the smallest independently programmable physical elements on the display surface. That's the ideal, anyway. In practice, however, the display hardware may juxtapose or overlay several smaller dots to form an individual pixel. This is true in the case of most analog color CRT devices, which use several differently colored dots to display what the eye, at a normal viewing distance, perceives as a single, uniformly colored pixel.

Because physical pixels cover a fixed area on the display surface, there are practical limits to how close together two adjacent pixels can be. Asking a piece of display hardware to provide too high a resolution--too many pixels on a given display surface--will create blurring and other deterioration of image quality if adjacent pixels overlap or collide.

3.5.2 Logical pixels

In contrast to physical pixels, logical pixels are like mathematical points: they specify a location, but are assumed to occupy no area. Thus, the mapping between logical pixel values in the bitmap data and physical pixels on the screen must take into account the actual size and arrangement of the physical pixels. A dense and brightly colored bitmap, for example, may lose its vibrancy when displayed on too large a monitor, because the pixels must be spread out to cover the surface. Figure 4 shows the difference between physical and logical pixels.

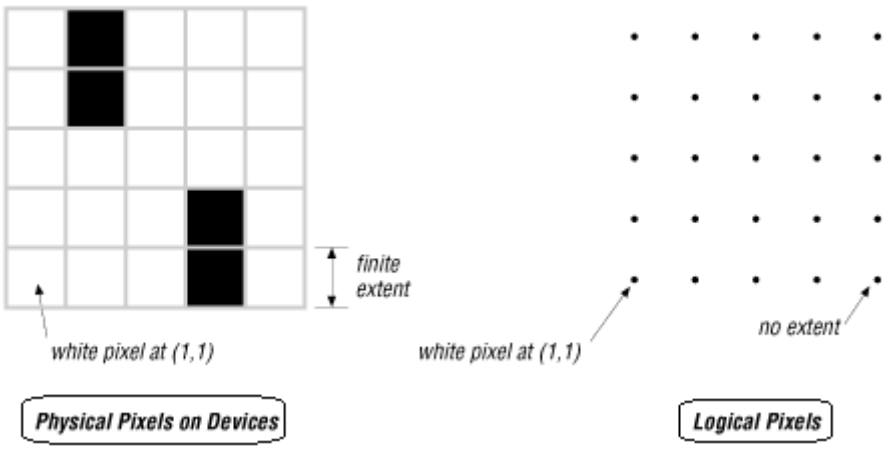


Figure 4: Physical pixels vs logial pixels

3.6 Resolution, pixel depth and displays

The number of bits in a value used to represent a pixel governs the number of colors the pixel can exhibit. The more bits per pixel, the greater the number of possible colors. More bits per pixel also means that more space is needed to store the pixel values representing a bitmap covering a given area on the surface of a display device. As technology has evolved, display devices handling more colors have become available at lower cost, which has fueled an increased demand for storage space.

Most modern output devices can display between two and more than 16 million colors simultaneously, corresponding to one and 24 bits of storage per pixel, respectively. Bi-level, or 1-bit, displays use one bit of pixel-value information to represent each pixel, which then can have two color states. The most common 1-bit displays are monochrome monitors and black-and-white printers, of course. Things that reproduce well in black and white--line drawings, text, and some types of clip art--are usually stored as 1-bit data.

The resolution is the number of pixels displayed in one screen (see figure 5). With higher resolution more quality in graphics representation. In graphical mode, a display can represent pixels in different modes such as 640x480, 1024x768, 1152x864, etc.

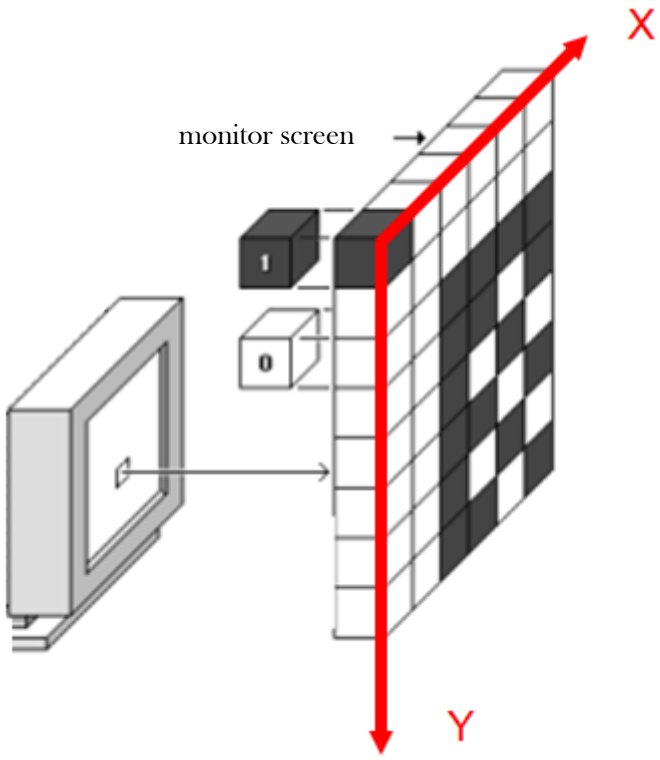


Figure 5: Graphical representation

3.7 Pixel Data and Palettes

Obviously, pixel values stored in a file correspond to colors. But how are the colors actually specified?

One-bit pixel data, capable of having the values 0 and 1, can only fully represent images containing two colors. Thus, there are only two ways of matching up pixel values in the file with colors on a screen. In most situations, you'll find that a convention already exists that establishes which value corresponds to which color, although a separate mechanism may be available in the file to change this. This definition can also be changed on the fly by the rendering application.

Pixel data consisting of more than one bit per pixel usually represents a set of index values into a color palette, although in some cases there is a direct numerical representation of the color in a color definition scheme.

A palette, which is sometimes referred to as a color map, index map, color table, or look-up table (LUT), is a 1-dimensional array of color values. As the synonym look-up table suggests, it is the cornerstone of the method whereby colors can be referred to indirectly by specifying their positions in an array. Using this method, data in a file can be stored as a series of index values, usually small integer values, which can drastically reduce the size of the pixel data when only a small number of colors need to be represented. Bitmaps using this method of color representation are said to use indirect, or pseudo-color storage.

Four-bit pixel data, for instance, can be used to represent images consisting of 16 colors. These 16 colors are usually defined in a palette that is almost always included somewhere in

the file. Each of the pixel values making up the pixel data is an index into this palette and consists of one of the values 0 to 15. The job of a rendering application is to read and examine a pixel value from the file, use it as an index into the palette, and retrieve the value of the color from the palette, which it then uses to specify a colored pixel on an output device. Figure 6 shows how a palette may be used to specify a color.

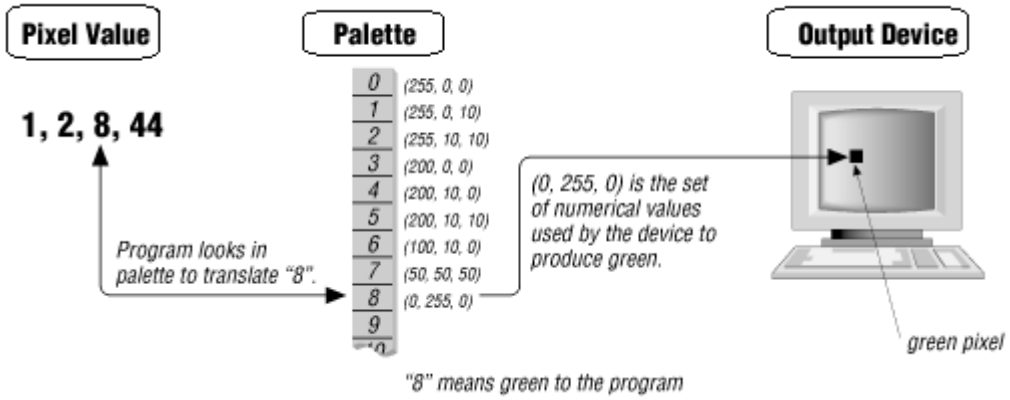


Figure 6: Graphical representation

The palette is an array of colors defined as accurately as possible. In practice, each palette element is usually 24 bits, or three bytes, long, although to accommodate future expansion and machine dependencies, each element is sometimes stored as 32 bits, or four bytes. Curiously, color models, many of which existed prior to the computer era, are often built around the equal partition of the possible colors into three variables, thus neatly fitting into three bytes of data storage.

What this means is that palettes are three or four times as large as the maximum number of colors defined.

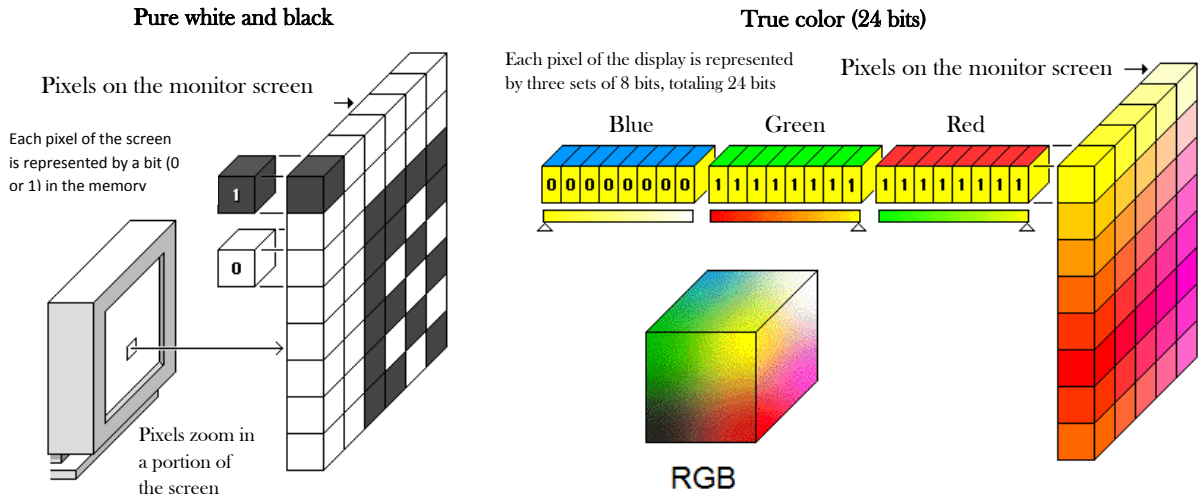


Figure 7: Color palette and the RGB

In this manner, the video memory is the amount of memory required to support a graphics mode. For example, an image of 640 x 480 pixels with 24 bit color will require nearly one megabyte:

640 x 480 x 24 = 7.372.800 bits

7.372.800 / 8 = 921.600 bytes

921.600 / 1.024 = 900 kilobytes = 900 kb

4 Lab

4.1 Objective

The main goal is to develop a GUI programming the basic Qt controls.

4.2 Equipment

- PC compatible computer with Microsoft Windows 7 operating system.
- Open source version of Qt framework for MWindows.

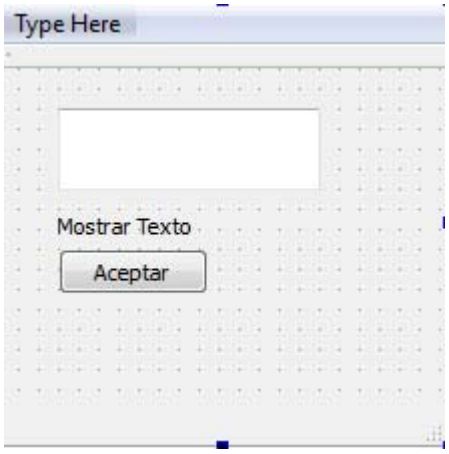
4.3 Departing point

The student should have completed the previous practices.

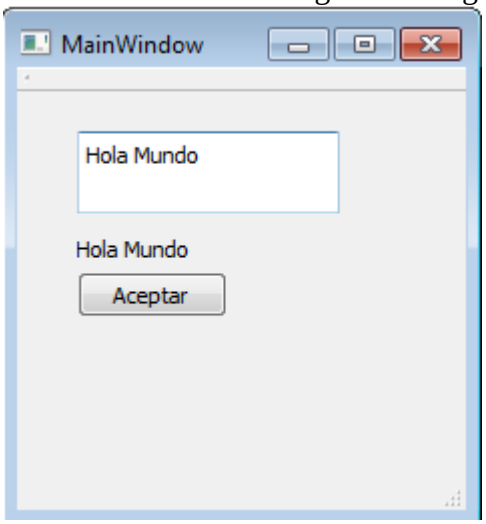
4.4 Activities

Enter text using the IGU:

Introduce in the main window of our application an element of type "TextEdit". Then to find out if the text entered is correct we will use a Label type element. As to check you entered data we create for ourselves a button to somehow already explained in previous



After his execution will get a message similar to the following:



To get the result, code similar to the following will be implemented:

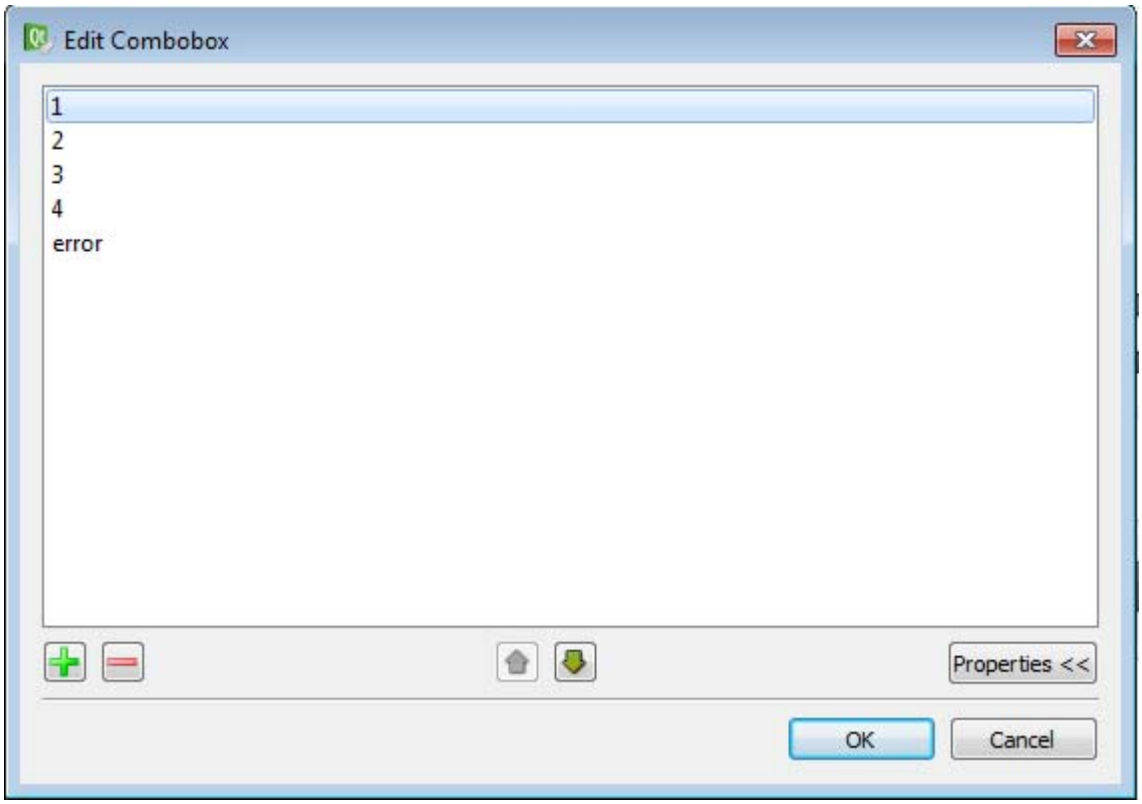
```
void MainWindow::on_BtAccept_clicked() {
   datoentrada = ui->inputText->toPlainText();
   ui->outputText->setText(datoentrada);
}
```

Where ui->inputText-toPlainText() will return the text entered in the white box, in a string format, therefore, for that we have a variable of type QString where to store the value

`ui->outputText->setText(QString )` used to print messages via the Graphic interface, for it will pass the value that is stored in the variable `datoentrada` which is of type QString

Similarly we can perform the same operation with various elements that we can serve to enter data into our application, another way is to use the Combo box element to drag it our main window, and we'll add another Label element.

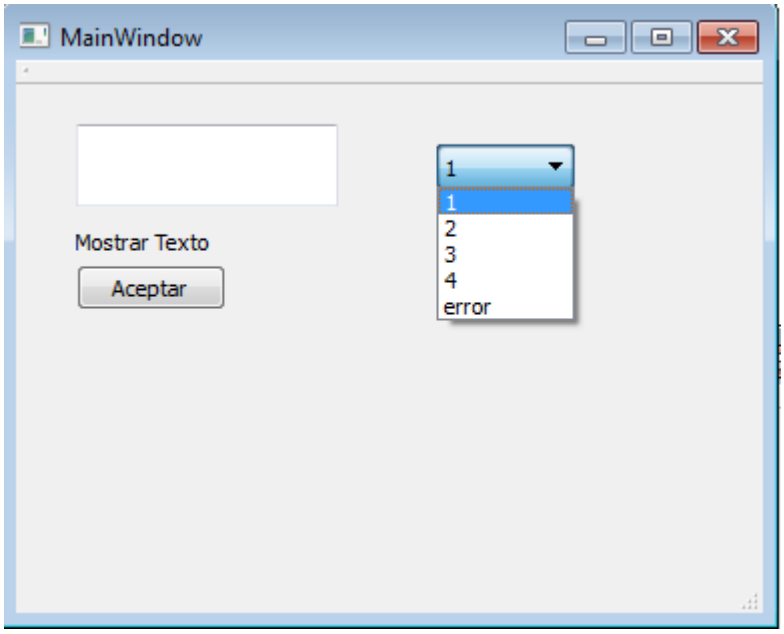
Once we have located in our application we will double clicking or clicking the right mouse button on the object, we'll go edit combo box and a window will appear as follows



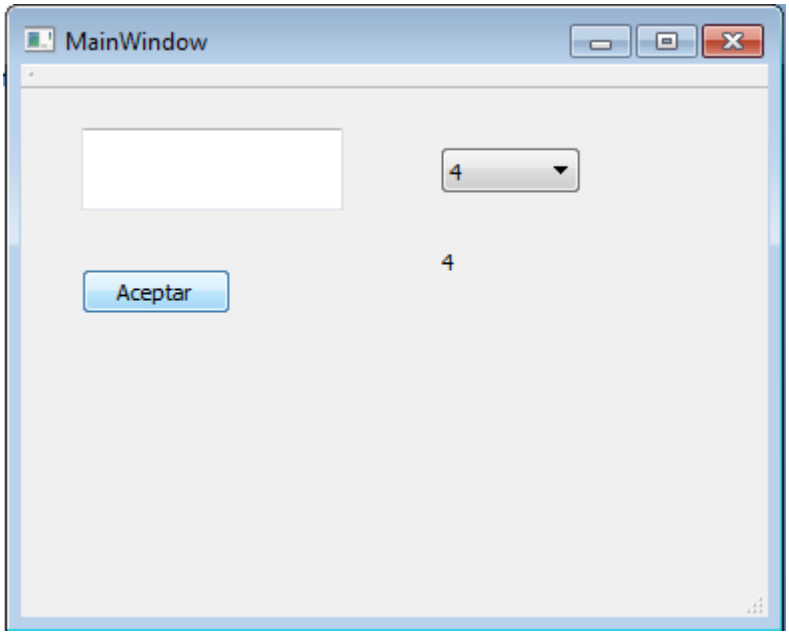
In this window you can add options to the dropdown giving the + button and edit their properties where we have introduced our choices then we give the OK button.

To read the value you have selected in our code we have to write a line similar to the following: combo = = ui->desplegable->currentText(); where currentText() returns the value selected in a QString variable.

Run our application and if we click on the drop down options we will have written.



Obtain a result similar to this after returning to give the button.



The next element to show its operation is the CheckBox element. To start using the drag to the main window of our application. Once dragged we changed the name to make it easier for us to recognize, in this case we have called *enableCheck,* to show its operation was performed as enabling and disabling a button each time you click.

To perform control actions in our first application of all we write the following line of code:

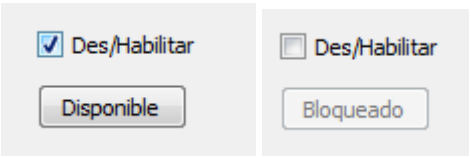
connect(ui->enableCheck, SIGNAL(stateChanged(int)),this,SLOT(habilitar(int)));

thus be established that the signal changes according to the value it has at that time element, so the stateChanged(int) signal were employed also in the Slot we have the call to the function that contains the code necessary to run our code.

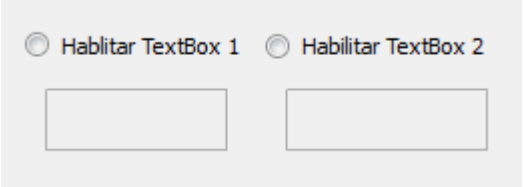
To do this, first add the mainwindow.h file section that puts private slots, the function: void habilitar(int); after entering this line of code we go to mainwindow.cpp file and we can write the following code:

```
void MainWindow::habilitar(int estado){
    if(estado){
        ui->BtTest->setText("Disponible");
        ui->BtTest->setEnabled(ENABLE);
    }
    if(!estado){
        ui->BtTest->setText("Bloqueado");
        ui->BtTest->setEnabled(DISABLE);
    }
}
```

This function receives as argument an integer, which would tell us the current state of the checkbox, if the value is 0 the elements that are associated with its operation will be disabled or as we wish, and vice versa. In this example, we have done that if state = ENABLE, button is available for possible executions, and if it appears DISABLE, button locked. In the header file defined the ENABLE = 1 and DISABLE = 0



The Radiobutton item you can use to select an option, keep in mind that the choice is exclusive, there can only be a selection in a moment of execution. In our example we will use a couple of these elements to demonstrate its operation.



Drag our implementation, two Radiobuttons and two TextEdits. First of all, when the application starts, we should disable these TextEdits, to do this we should write:

```
ui->textBox1->setEnabled(false);
ui->textBox2->setEnabled(false);
```

below the line:

```
ui->setupUi(this);
```

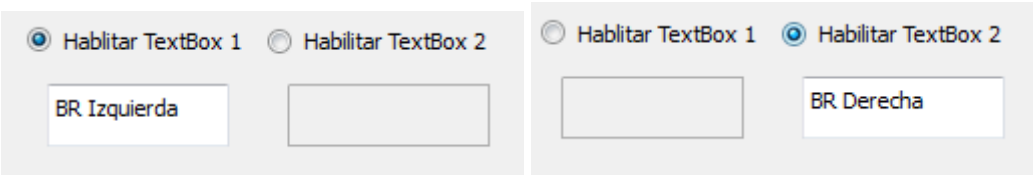
Then we write:

```
connect(ui->RDIzquierda,SIGNAL(clicked(bool)),this,SLOT(seleccionar(bool)));
connect(ui->RDDerecha,SIGNAL(clicked(bool)),this,SLOT(seleccionar(bool)));
```

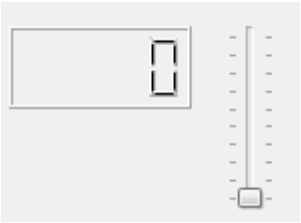
We should detect which has been selected with the clicked signal, and select slot will receive a Boolean value that its implementation will serve us to detect which of the two Radiobutton has been selected. To do this we should write the function:

```
void MainWindow::seleccionar(bool seleccion){
    if(ui->RDIzquierda->isChecked()==seleccion) {
        ui->textBox1->setEnabled(true);
        ui->textBox2->setEnabled(false);
        ui->textBox1->setText("BR Izquierda");
        ui->textBox2->setText(" ");
    }
    if(ui->RDDerecha->isChecked()==seleccion) {
        ui->textBox2->setEnabled(true);
        ui->textBox1->setEnabled(false);
        ui->textBox2->setText("BR Derecha");
        ui->textBox1->setText(" ");
    }
  }
```

`isChecked()`, returns true or false depending on whether it has been selected or no, and with the help of the variable selection we can enable and display a text in the corresponding TextEdit



The next element will display its basic operation is the Slider, two Sliders, Vertical and Horizontal, both are identical. This element allows us to establish a range of values and go as moving changes its value. To demonstrate how it works we will use the LCD Number element, this is a 7-segment display. To do this, add both elements in our application.



Once added two elements select Slider and the secondary button to go to Go to Slot, once there select the slot ValueChange(int)

And we created in the file mainwindow.cpp function:

```
void MainWindow::on_SliderContador_valueChanged(int value)
```

Then we write lines of code:

```
ui->SliderContador->setTickPosition(QSlider::TicksBothSides);
ui->SliderContador->setTickInterval(1);
ui->SliderContador->setMinimum(0);
ui->SliderContador->setMaximum(9);
```

below the line:

```
ui->setupUi(this);
```

these lines we introduced allow us to enable subdivisions along the slider on both sides. Show subdivisions with a range of values, and finally set a minimum and maximum value of the slider.

To check its operation, within the function that created us after creating the slot, introduce the following line:

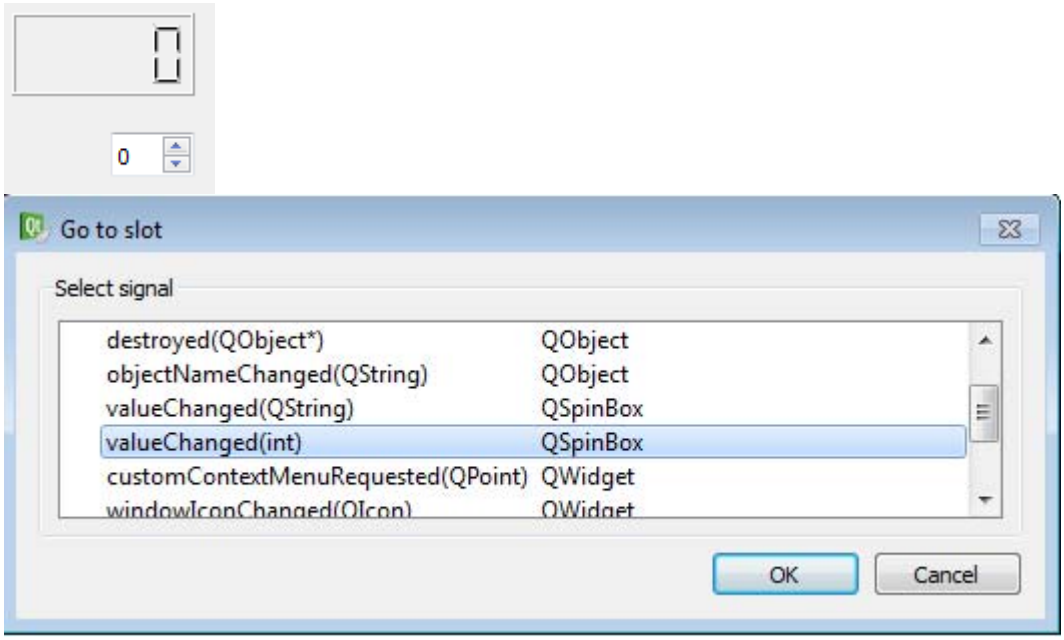
```
ui->SieteSeg->display(value);
```

Remain:

```
void MainWindow::on_SliderContador_valueChanged(int value){
    ui->SieteSeg->display(value);
}
```

This is allowing us to see the values of the slider in the display as we get moving your rod.

The next element will introduce the SpinBox, similarly to what we have done in the previous section, once we dragged the item to our application, we will click and select Go to slot and select valueChanged(int), with this we create a new feature in our file .cpp, we will reuse the LCD Display to check its operation.



The resulting function with its implementation remains similar to the following:

```
void MainWindow::on_spinBox_valueChanged(int value){
   ui->SieteSeg->display(value);
}
```

Our next item to present will be the Dial object, this can serve to simulate controls industrial instruments, and other items. To demonstrate how it works we will make use of a graph, for example to be a graphic Voltage/Time.

drag the Dial object to your application

Then we wrote the lines:

```
ui->dial->setNotchesVisible(true);
ui->dial->setSliderPosition(0);
ui->dial->setMinimum(-15);
ui->dial->setMinimum(-15);
```

below the line:

```
ui->setupUi(this);
```

these lines you do is enable the dial subdivisions, establish an initial position of the Dial, in this case the values can range between -15 and 15 has been chosen to put the average value 0. Once we have done this, go to object and click on Go to slot, and select for example sliderMoved(int) slot, the slot which will allow us is to get the current value of the position of the Dial

ANNEX

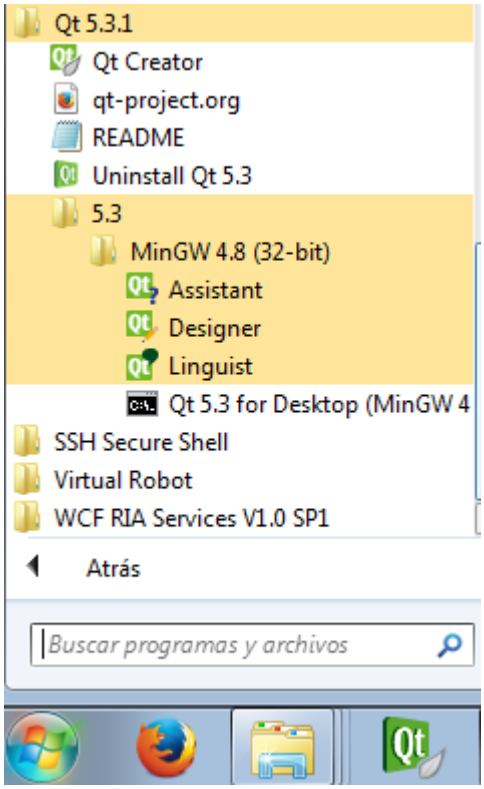
The QWT library can be found in the site http://qwt.sourceforge.net/. This library contains GUI components that can be integrated into the IDE Qt4.4 or later.

To proceed with the installation of the library graphic GWT, we will have to unload the library from the site http://sourceforge.net/projects/qwt/files/qwt/6.1.1/qwt-6.1.1.zip.

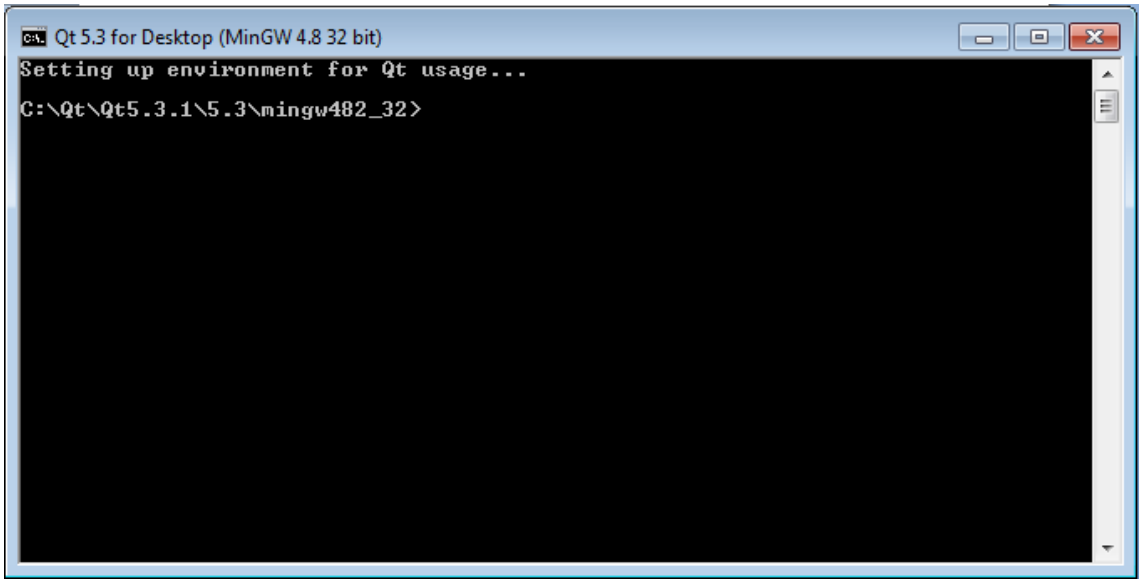
Once we downloaded unzip the file and place it in the C:\infi.

Then we have to go to the Windows Start button, and all programs move to Qt 5.3.1 directory.

Once found the directory, we will enter, and clicked about 5.3 directory, and then the MinGW 4.8 directory and then clicked on Qt 5.3 for Desktop item



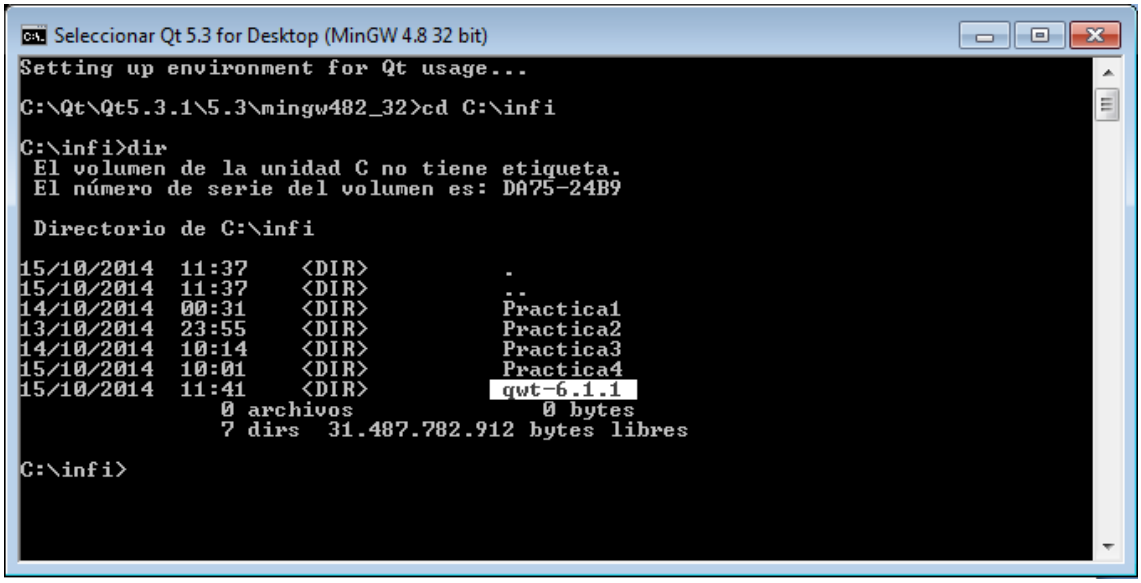
Once we have clicked will launch the system console, similar to the following image



Below us we locate the directory where you unzipped the QWT-6.1.1 library to write this

> "cd C:\infi\" and once we change the route, we can write

> "dir" and show us the contents of this directory



We can see that indeed this qwt-6.1.1 directory

Then we rewrite> "cd qwt-6.1.1" and will access inside the directory. And we have to change a couple of lines before compiling the package. Write> "notepad qwtbuild.pri"

And the file that we were there we seek the "CONFIG += debug_and_release" line, if you have at the beginning of the line "#" symbol have to remove it, and if not this will have to write.

```
win32 {
    # On Windows you can't mix release and debug l
    # The designer is built in release mode. If yo
    # you need a release version. For your own app
    # might need a debug version.
    # Enable debug_and_release + build_all if you

    CONFIG          += debug_and_release
    CONFIG          += build_all
}
else {
```

Once we have done this save the file, and now we rewrite the following line in the console > “notepad qwtconfig.pri” and again we will open the notebook, now we must look for the following line "QWT_CONFIG += QwtExamples " and we have to activate this option, it will remove the start of his line the symbol " # " being as shown in the following image:

```
#############################################################
# If you want to auto build the examples, enable the line below
# Otherwise you have to build them from the examples directory.
#############################################################

QWT_CONFIG     += QwtExamples

#############################################################
# The playground is primarily intended for the Qwt development
```

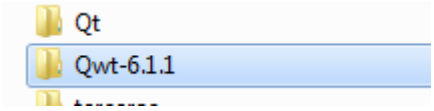
We return to save the modified file and close Notepad. Continuing the open console, we introduce the following commands:

> "qmake qwt.pro"

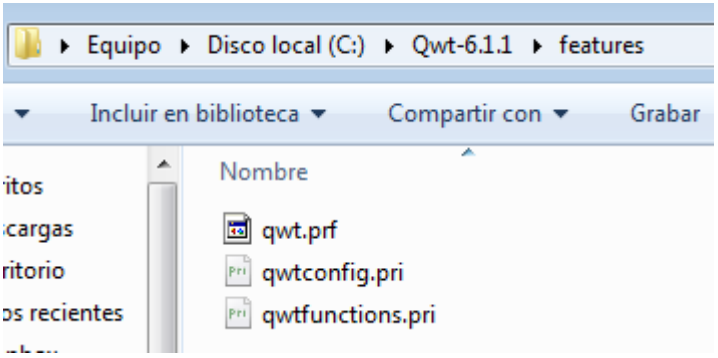
> "mingw32-make"

> "mingw32-make install"

At the end of the compilation process will go to C: \ with scanner and confirm that the directory exists.

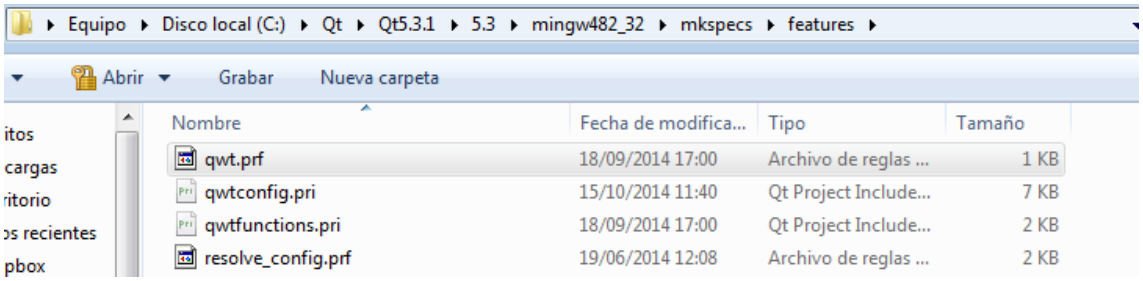


Walked into that directory and we are in the "features" folder and give copy to the elements that are inside the folder.



And we pasting in the following directory

C:\Qt\Qt5.3.1\5.3\mingw482_32\mkspecs\features



Link to added QWT al Qt: http://meleta.es/instalar-qwt-para-qt-widgets-y-graficos-2d/

5 Seminar

5.1 Introduction

This seminar will be to work about the GUIs development and several alternatives in its implementation. On the second part the seminar focuses on the importance of a good GUI.

5.2 Objectives

The main objectives are:

- To relate the GUI concepts with other technical concepts that usually are studied in different subjects, in a contextualized way.
- Acquire team-working skills, discussion the GUI development topic.
- Acquire documentation and presentation skills.
- Acquire critical searching of information skills, necessary to find new resources to develop modern, intuitive and impressive GUIs.

5.3 GUI basic aspects

This seminar will be to work on the fundamentals aspects of graphical user interfaces.

With internet we will collect the following information:

- List of GUI development libraries of components

- Main features of each library and the purpose of its components. The students should discuss about the possibilities of these libraries and components and the possibilitie to integrate it in the GUI of industrial applications.
- Comparative review of each of the components.
- Library would be chosen as the most suitable for the realization of a GUI for and industrial process.

After that students should work in groups of 4 people, in order to think about the resources and possibilities in GUI for industrial applications they search in internet and learned in the lecture and labs. At the end of the seminar they must explain and share their experiences with the whole group.

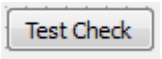
6 Mini-project

6.1 Implementation of the GUI

The activity related with the mini-project will consist in the development of the GUI basics that allows showing the information in a friendly manner to the user. And on other hand must allow obtaining the necessary information from the user in an intuitive manner.

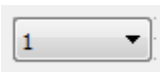
In this activity students should use for the graphical interface design at least the following elements, thinking in the mini-project integration and intuitive use by the user, so that it will facilitate the way to enter the information into the application and observe the behavior of the system:

- Buttons



- TextBox

- ComboBox



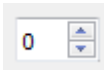
- CheckBox

- RadioButton

- LCD number

- Sliders

- SpinBox



- Dials

After that the student should validate the developed GUI through the appropriate tests.

7 Bibliography

Jeff Johnson , *Designing with the mind in mind : simple guide to understanding user interface design rules*. Ed. Morgan Kaufmann, 2010, ISBN 9780123750303.

Foley, James. D., Andries van Dam, S.K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice*, second edition, Addison-Wesley, Reading, MA, 1990.

Soren Lauesen, *User interface design: a software engineering perspective*. Pearson Addison Wesley, 2007. ISBN 9780321181435.

Ballard, Barbara. *Designing the mobile user experience*. Ed. Wiley, 2007. ISBN 9780470033616.

Rogers, David F. and J. Alan Adams, *Mathematical Elements for Computer Graphics*, second edition, McGraw Hill, New York, NY, 1990.

Wilbert O. Galitz, *The essential guide to user interface design : an introduction to GUI design principles and techniques*, 2nd ed. John Wiley & Sons, 2002. ISBN 0471084646.

http://www.labview.com

Digia, Libreria Qtqml.Models, 2013, http://qt-project.org/doc/qt-5/qtqml-models-qmlmodule.html