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 1: Introduction

Co-funded by the Tempus Programme of the European Union

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

Authors: Domínguez, C.; Hassan, H.; Martínez JM

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 1: Introduction

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): Domínguez, C., Martínez, J.M., Hassan, H.

[illegible]

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)	Domínguez, C.; Hassan, H.; Martínez JM
Reviewers	Perles, A., Capella, J.V., Albaladejo, J.

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 # we are going to make an introduction on different possibilities that we can find when designing an industrial computing system. Design options will be presented in relation to the hardware and to the software.

Different parts in an industrial computer system, comprising from the user interface, the interface with the process and the digital image are presented.

In addition there will be a lab exercise to use the Qt programming environment with a simple application.

Part of the seminar will delve into the C programming language watching different programming tools

Finally, an analysis of the necessary requirements are presented to address the miniproject

3 Lecture

3.1 Introduction

This chapter pretends to deepen into the general structure of an industrial informatics system and to know and get students to be aware of the applications of this type that can be around him.

Students will learn the basic ideas of possible hardware and software solutions and choose which projects to tackle industrial computing.

3.2 Architecture of an Industrial Informatics System

Figure 1 shows a simple schematic of a generic industrial informatics system industry. On the one hand is often necessary to know the status of the industrial system and the other may need to act on it, for it will make use of appropriate sensors and actuators.

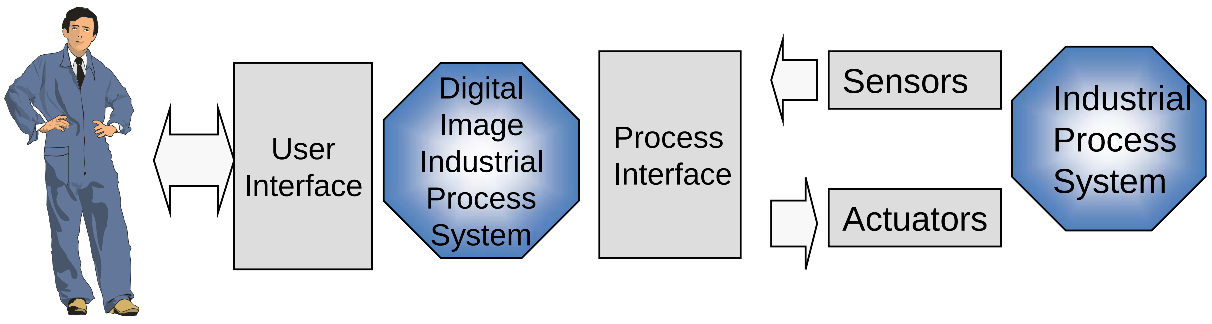

User Interface
Digital Image Industrial Process System
Process Interface
Sensors
Actuators
Industrial Process System

Figure 1: Diagram of an industrial informatics system

Part of an industrial computer system must be able to interact with the process through sensors and actuators. In this part we will call **process interface**. This interface will update the internal digital image of the process inside the computer application.

The computer application usually needs to interact with an operator to display information and receive orders. This part of the system will call **operator interface**.

3.3 Hardware Solutions

In the industrial computing market are a lot of hardware solutions to choose from depending on the needs of the particular project.

This section describes the major groups of solutions described in terms of hardware and what is appropriate for each type of problem.

In most cases, teams will have to operate in harsh environments, which typically include mechanical protection against electromagnetic interference and against corrosive environments.

3.3.1 Embedded Systems

Solution of embedded system design is based on the own choice of the designer of the chips that will form the system, the development of boards and electronic circuits.

It is the choice for Industrial Informatics Systems often provides the best solution to every problem and the most economical in terms of material cost of the final system (Figure 2).

But it is the most expensive in terms of investment in the design stage; it requires specialized personnel time and very expensive tools (software programs design, printed circuit simulators, cross compilers, etc.).

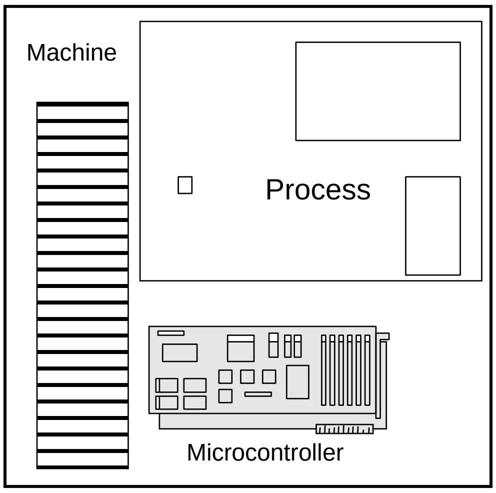


Figure 2: Example of an embedded system structure

When is the right solution?

- For large number of production where the development cost is offset by the savings generated by the material cost
- For specific designs where it is difficult to resort to other solutions.

The Industrial Informatics Systems designed as "Embedded System" is the solution that most people use it without realizing it. Thousands of small electronic utensils used this solution, for example:

- In consumer electronics: televisions, stereos, CD players, videos, cameras, cash cards, etc.
- In computer: printers, scanners, hard drives, digital monitors, modems, network cards, etc.
- In instrumentation: digital oscilloscopes, etc.
- In automotive: injection system, ABS, airbag, etc.
- Robots, etc.

Based solutions in embedded systems often make use of more specific processors such as microcontrollers that are complete computers on a single chip, and DSP (Digital Signal Processor) (Figure 3).

Figure 3: Example of an embedded control system board

3.3.2 Systems based on Industrial Bus

Another hardware solution for Industrial Informatics Systems is the Industrial Bus based, that consists on an industrial computer system mounted easily from standard commercial cards and systems designed. The cards are interconnected using a standardized bus, implying that all must follow a standard so that they can communicate with each other.

Thus, the standard bus is the definition of a specification to be met cards to be connected to it and refers to:

- Mechanical: Wherein are defined the dimensions of the card, connector dimensions that must take these to join them to the bus separation between cards, etc.
- Electrical specifications. Where are defined the electrical characteristics of the signals to be injected into the bus.
- Functional specifications. By means of which the function of each power line of the bus (data, address, control or supply) is described.

Figure 4 shows a diagram of this solution. On a physical bus composed of a board with standardized connectors commercial that is connected to commercial boards (can be from different manufacturers) that the designer has chosen.

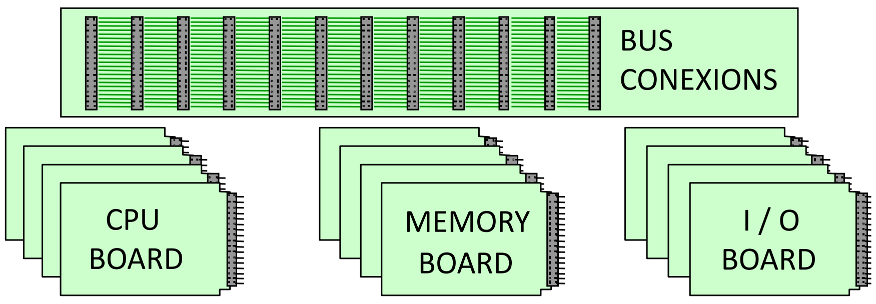


Figure 4: Elements of a Standard Bus

This is a solution that tries to get a specific system without the disadvantages of embedded systems. It normally needs specific programming languages and operating systems for their development.

When is the right solution?

- For systems where the cost of the industrial informatics system no effect on the overall cost of the system.
- Integrated systems with great flexibility in terms of extensions and improvements

In the market there are various standardized industrial buses, for example, the VME bus. The Figure 5 and Figure 6 show the computer system of a European commercial helicopter last generation VME bus and implemented using a processor board based on Motorola PowerPC processor.

Solving industrial bus type is currently applied to the production industry, flight systems (airplanes, helicopters, satellites, etc.), military systems (tanks, ships, missiles, etc.), etc.

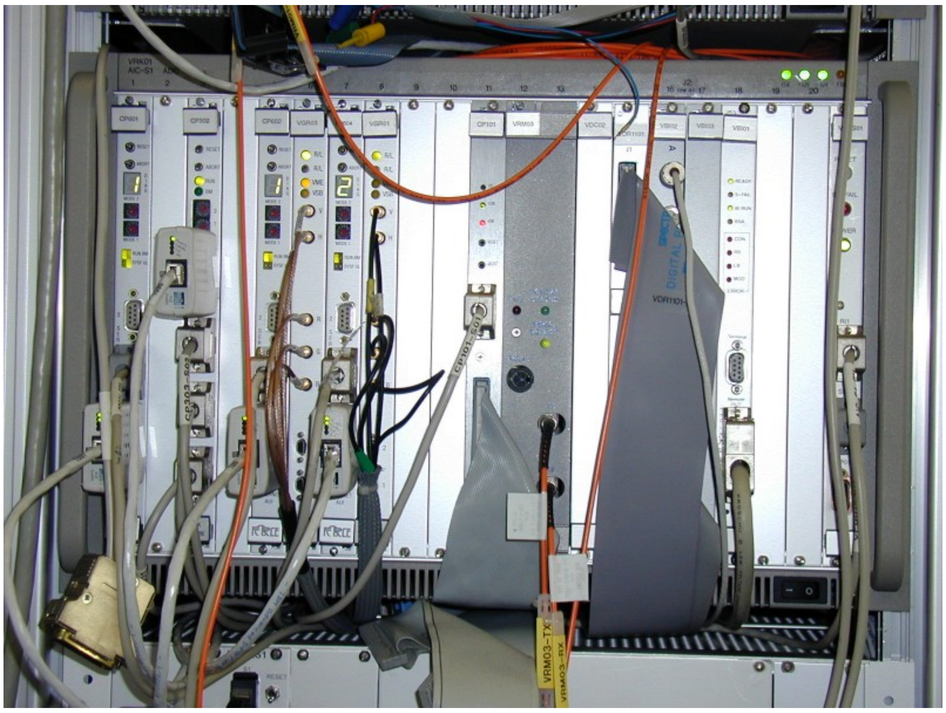


Figure 5: Computer system of a helicopter (German Aerospace Center)

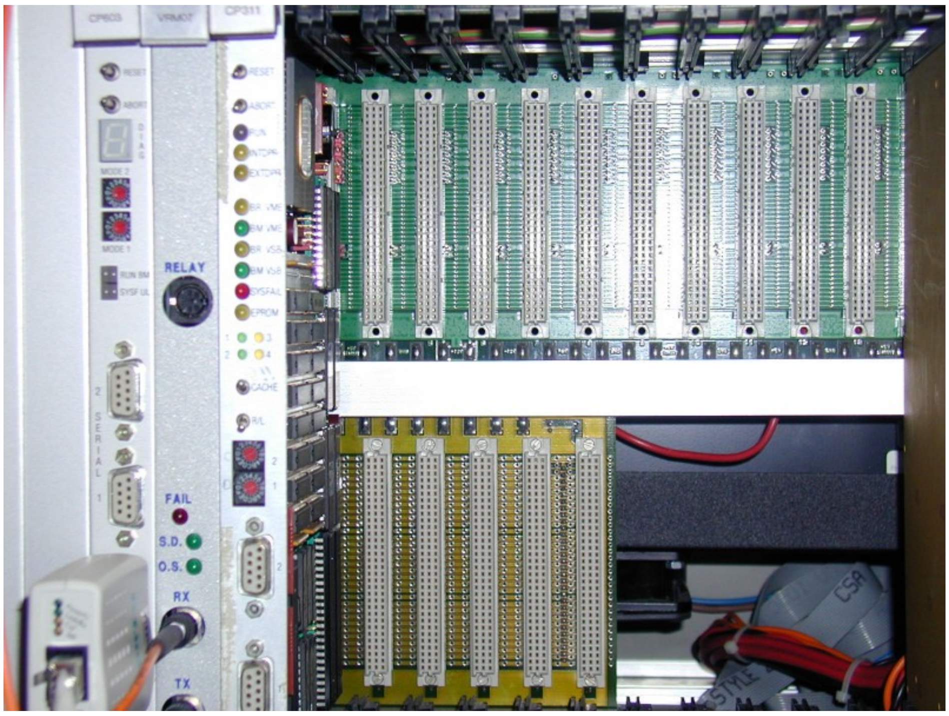


Figure 6: Detail of the bus connectors

3.3.3 Complete Computer Systems

Another solution for Industrial Informatics Systems is based on the use of full commercial computer systems that are presented in a board or in a box already assembled and ready for operation.

Such systems usually have integrated all you need to design an Industrial Informatics System, including some components that are not required in some applications, it is generic systems whose purpose is to be easily adapted to many problems. Although are not the best solution because it is not designed to solve a specific problem, note that it is a quick and cheap solution as it is usually made in large series.

When is the right solution?

- Flexible industrial systems and low/moderate costs for material and industrial informatics systems design.

In this field it is easy to find complete microcontroller systems on a board (see Figure 7) ready to use simply by turning on the power and even single board computers (see Figure 8), a very attractive solution for many developers because there are many software and hardware available with low cost.

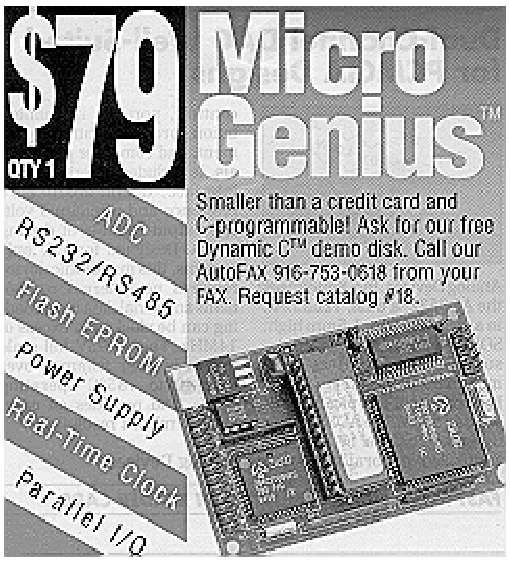

$79
QTY 1
Micro Genius™
Smaller than a credit card and C-programmable! Ask for our free Dynamic C™ demo disk. Call our AutoFAX 916-753-0618 from your FAX. Request catalog #18.
ADC
RS232/RS485
Flash EPROM
Power Supply
Real-Time Clock
Parallel I/O

Figure 7: Business board based on a microcontroller (#CANVI)

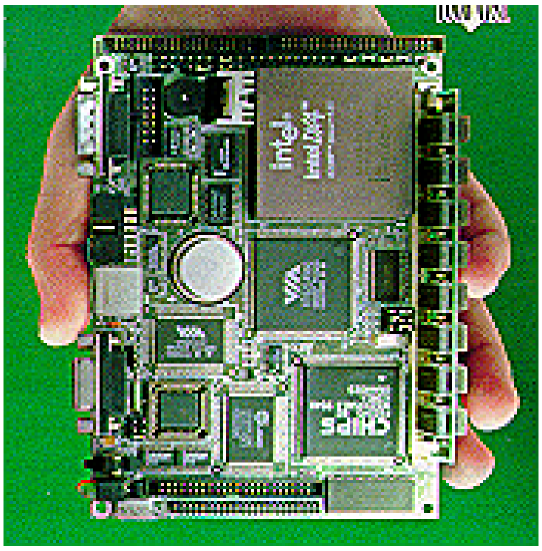


Figure 8: Complete single board computer system (#CANVI)

The microcontroller board systems are used in systems such as: vending, weather stations, slots, etc.

The complete PC systems and similar types are used: industrial control systems and monitoring, automated banking machines, POS, robots, etc.

3.3.4 Specific Computer Systems

There are many solutions based on industrial data that make up the computer system so that it is adapted to a specific type of problems and does not require computer skills.

Some examples of this are the **programmable controllers** or **PLC** (Programmable Logic Controller) and **industrial controllers**.

3.3.4.1 Programmable Controllers

PLCs (see Figure 9) are used in automation for direct replacement of electromechanical systems based on relays and contactors in order to get a more flexible and cheaper system (wired logic is replaced by programmable).

As they are designed with the aim that can be used for the people that normally employ automatic systems based on relays, all are programmed using the called contacts diagrams or logic functions.

Successfully applied to automatic systems (packing, sealing, automatic conveyors, etc.), to lights, to elevators, etc.

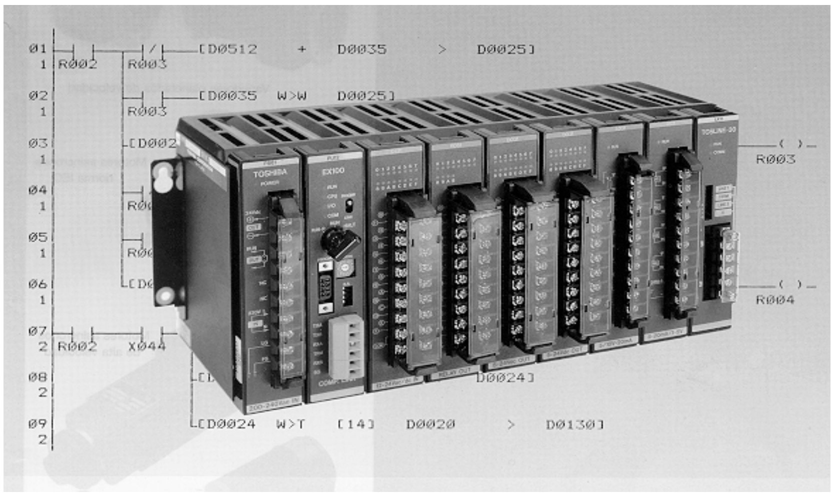


Figure 9: PLC (#CANVI)

3.3.4.2 Industrial Controllers

An industrial controller is a device designed to control a specific physical quantity, for example, the temperature of a liquid, the speed of a motor, a piston pressure, etc.

They employ discrete theories on closed-loop control is typically implemented by a microcontroller.

They are a simple and inexpensive solution to problems of closed-loop control. An example can is shown in Figure 10:

Figure 10: Digital PID Controller of Fuji

3.4 Software Solutions

A computer system without software is useless. This section provides an overview of possible solutions for the software will be provided.

3.4.1 Programming Languages

A programming language is more than a simple notation to operate computers. Although the hardware is the same, it may work differently only changing programming language, as each new language brings a new concept to the computer.

The only language that truly understands a computer system is the **machine code**, a sequence of digital values (0 and 1) as the processor interprets simple commands. They are machine dependent language, meaning that each processor uses its particular machine code and imply a deep knowledge of the internal structure of the computer to operate on it.

Writing programs in this language presents a great difficulty, so a first step in the development of programming are the **assembly languages**, which consist basically the instructions written in machine language symbolically using mnemonics, readable codes that makes more easier to remember. A program written in this language is not directly executable by the machine, and must be translated into an equivalent program in machine code programs. The software that provides automatic translation is called assembler.

The use of assembly language is an ideal solution for systems based on small microcontrollers and DSPs because they generate compact and fast code. Figure 11 shows an example of assembly program

```
CrtCopyWin:
        JC      @@4
@@1:    LODSW
        MOV     BX,AX
@@2:    IN      AL,DX
        TEST    AL,1
        JNE     @@2
        CLI
@@3:    IN      AL,DX
        TEST    AL,1
        JE      @@3
        MOV     AX,BX
        STOSW
        STI
        LOOP    @@1
        RET
@@4:    REP     MOVSW
        RET
```

Figure 11: Fragment of an assembly program

The need for more general languages that allow a similar symbolism to the logical-mathematical writing gives rise to the so-called high-level languages. Not interested in the concepts of the machine, which programs are shorter and clearer, and because of this independence of the machine, a program can be executed on different computers.

Examples of such languages are: Pascal, Cobol, Modula, Prolog, C, Basic, Lisp, Fortran, Ada, HTML, Java, etc.

The use of high-level languages implies the existence of programs that can translate the instructions pertaining to the high-level language to machine instructions directly executable. These translation programs can be compilers or interpreters. Table 1 shows a comparison of an instruction in different languages.

Lenguaje	Código
Lenguaje máquina Intel 8x86	1011000000010010
Lenguaje ensamblador Intel 8x86	MOV AL, 12h
Lenguaje C	A = 0x12;

Table 1: Example of different programs

A compiler translates the original program (source) to the equivalent program in machine code (object) directly executable by the machine. The resulting programs are running fast but not like their equivalents in assembler code usually because its code size is larger than the equivalent machine code. A typical example of a compiled language is C.

An interpreter directly executes the original program instructions performing a rendition machine code of each instruction. The program implementation is much slower than its equivalent in assembler and compiler language, usually smaller than its compiled equivalent but always need interpreter, which usually occupies a lot of space. A typical example of an interpreted language is HTML used in web pages.

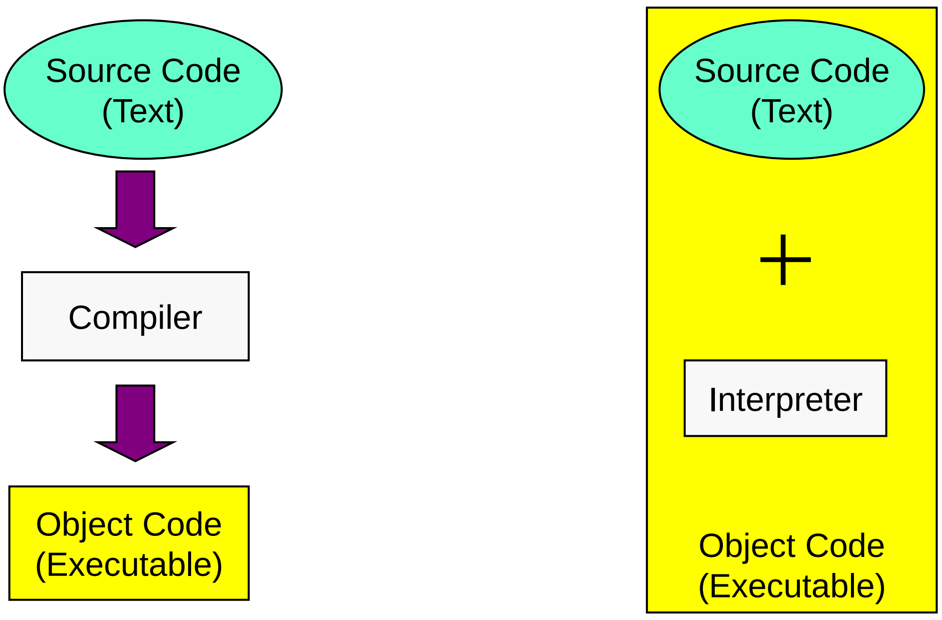


Figure 12: Scheme for the Obtaining compiled and interpreted executable

Many high-level languages are suitable for the development of industrial applications, but currently one of the most prevalent is C++. Except for small embedded systems, most of the industrial applications use high-level languages, either interpreted or compiled.

3.4.2 Operating Systems

When the computer system is becoming moderately complex may be useful to incorporate in the project an **operating system**.

An operating system is just a set of programs that make easier the use of the system by creating like a "virtual machine". In Figure 13 this situation is shown, the operating system creates a layer that hides the complexities of the machine showing it in a more simple way to use.

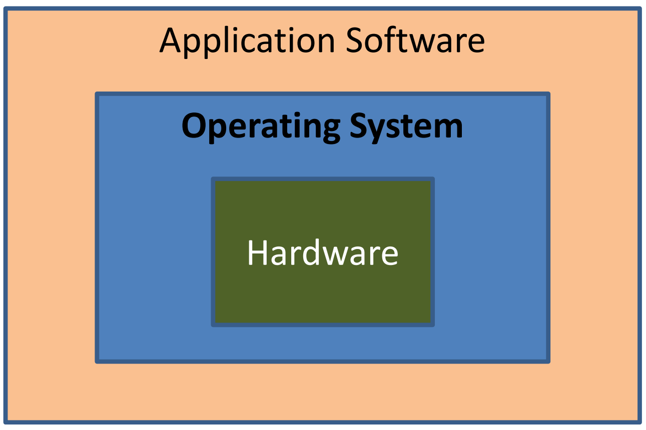


Figure 13: Hardware abstraction created by the operating system

Modern industrial informatics systems are quite complex and difficult and mostly unprofitable write programs that control all elements and used correctly. Ideally, the programmer is isolated from the hardware complexity, and to achieve this it has added a layer of software over hardware, which is responsible for managing all elements of the system, and presents the user interface of a virtual machine powerful, easy to understand and program.

Most current operating systems allow the apparently simultaneous execution of several programs (called tasks) while delivering the processor between each. These operating systems are called multitasking.

In the market there are a lot of available operating systems, from the so-called general purpose as Windows, MacOS, OS/2, Unix (Linux, HP-UX, Solaris, ...) to specific operating systems for industrial applications such as Lynx RT-DOS, etc. through TinyOS called microkernels small processors and microcontrollers.

3.4.2.1 Real-Time Operating Systems

In the market for operating systems for industrial computer systems include the so-called real-time operating systems that are multitasking operating systems where it is important that tasks meet deadlines.

The real-time tasks and, therefore, must meet deadlines can be divided into two groups:

- **Hard Tasks,** in which breaches of the terms can cause serious damage. Such as in the control system of an aircraft.
- **Soft Tasks,** in which breaches of the terms degrade performance. For example, a mobile telephone system.

Figure 14shows the times involved in the real-time tasks.

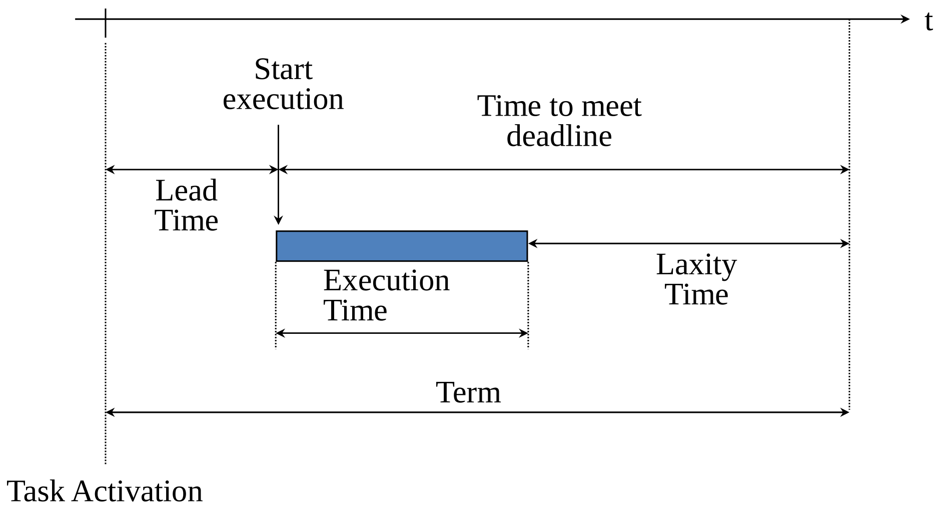


Figure 14: Times involved in a real time task

Most operating systems designed for industrial computer systems typically include real-time properties.

3.4.2.2 Rapid Development Environments of Industrial Computer Systems

An option to use a standard programming language is to use applications developed specifically for industrial computer systems.

Such environments typically have components "prefabricated" prepared to solve common problems.

An example of such systems is the Advantech Genie house, which, through a simple graphical interface, creating industrial control applications using drag and drop components. Another environment is currently booming popular is LabView by National Instruments. Both environments run on the Windows platform.

These solutions may be suitable for very rapid application development at a moderate cost, although some such as Genie suffer from lack of flexibility and difficult interaction with other languages, making it impossible to be able to do more things than the setting allows.

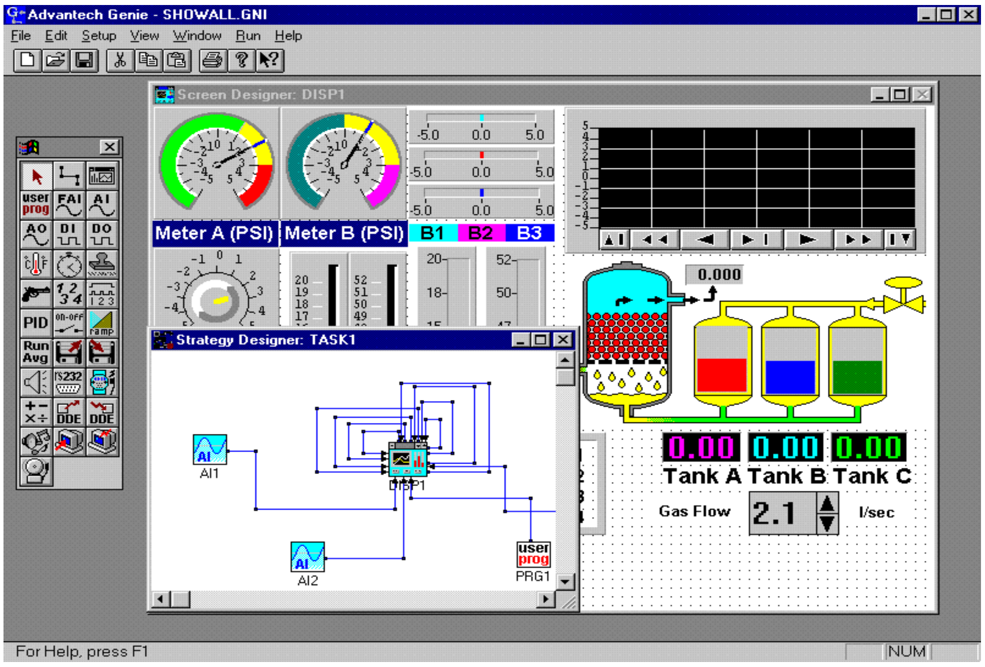


Figure 15: Genie Environment by Advantech

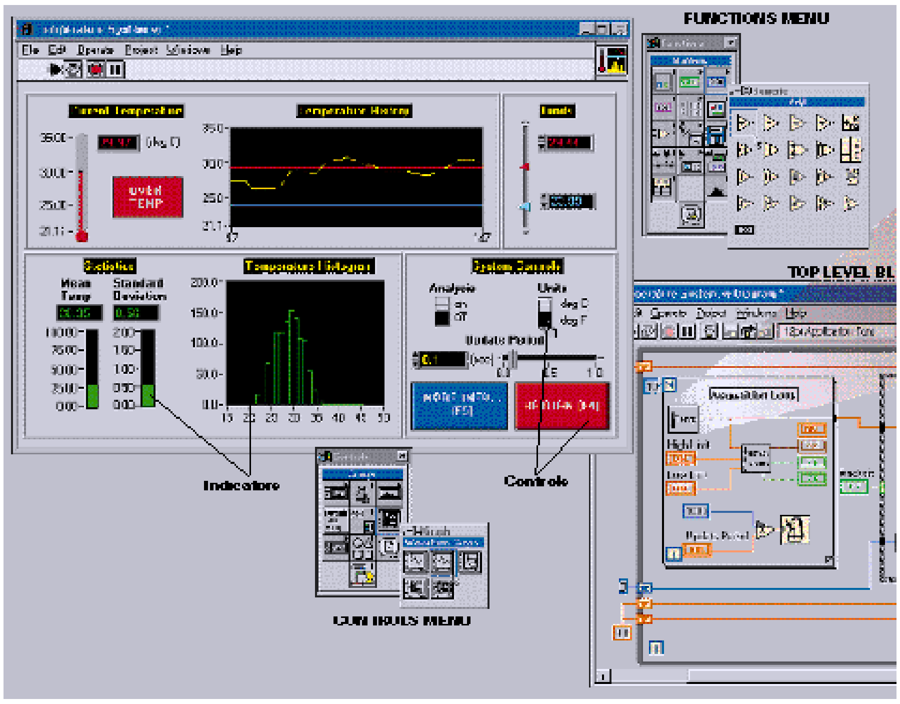


Figure 16: National Instruments LabView Environment

3.5 Additional Tools

In the development of industrial computer systems can be very useful to use tools to facilitate system testing.

Simulators, emulators and prototypes can be very suitable options.

A **simulator** (Figure 18) is used when the real system is not available or can be dangerous and can mimic the operation of a system.

An **emulator** (Figure 17**)** replaces part of a more complex system by a factor allowing imitate while incorporating features that facilitate testing.

Both, simulators and emulators can be much more expensive than the actual system itself.

Prototypes (Figure 19**)** are smaller versions of the final system to be used for testing as soon as possible to detect errors in the early stages of the project, allowing you to check if you have chosen the right path.

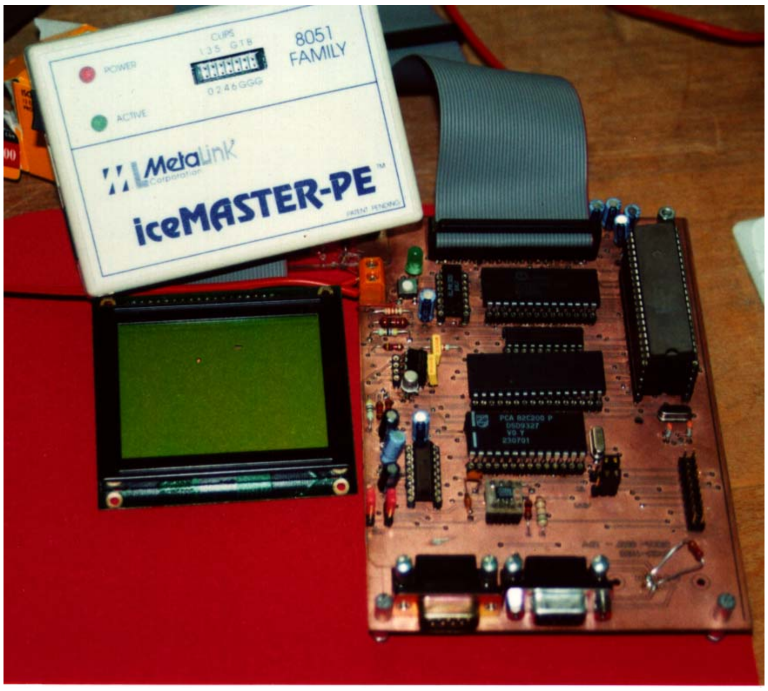


Figure 17: An emulator in the upper left corner

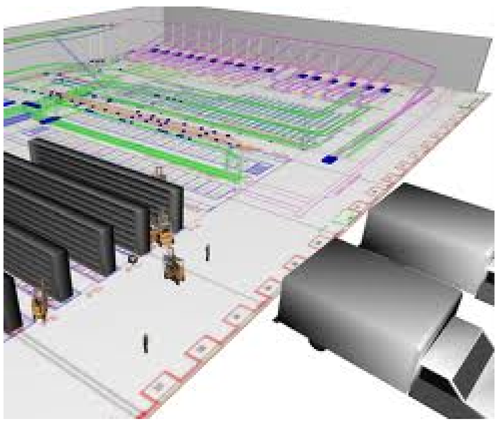


Figure 18: Example of Simulator software

Figure 19: Example of prototype (www.upc.edu)

4 Lab

4.1 Hello World (I)

4.1.1 Goals

Making contact with the visual programming environment for C ++ Qt.

- Create a project.
- Save it correctly.
- Introduce an object.
- Enter the code.
- Getting help.
- Compile and run the project.
- Know the job files.

4.1.2 Materials Required

- PC Compatible
- Development environment QT Creator

4.1.3 Introduction

Qt Creator is an IDE (or IDE: Integrated Development Environment). This development tool for creating applications using cross-compiling between different platforms and operating systems: Windows, Linux, Mac, Android, etc. It was initially created by Nokia. You can create two types of applications: under open source license (GPL) or by paying the license for commercial use ("Commercial"). In our case we will use the GPL open source license, due to our academic environment that is non-profit. For the development of applications development environment QT Creator (version 3.1.2) based on the QT 5.3.1 will be used.

To link to free software (open source Qt 5.3.1) we use the setting "offline". For this, web-Qt: http://qt-project.org/. In downloads menu (downloads), check "*view all downloads*" (Show Downloads) and choose "*opensource offline*" for platform you want. For example, for Windows, it would be this:

http://download.qt-project.org/official_releases/qt/5.3/5.3.1/qt-opensource-windows-x86-mingw482_opengl-5.3.1.exe

4.1.4 Development Application

Develop a graphical application is to make a program where the designer sets its focus on the visual aspect of the application, by distributing the visual elements (called widgets), such as tokens or forms, buttons, menus, pictures text, etc. In these applications the designer sees what is what is observed as a result of the application (WYSIWYG, what You See Is What You Get). The work is an application designer, apart from organizing graphic objects in programming the relationship between the graphical objects of a window, or several windows as required by the application. The QT visual development tool used applications window (or form), and drawing elements you have in the component palette. The designer thus establishes both the graphics that implementation and relationships between these components.

The steps are:

1. Designing the "Graphical User Interface" or GUI, intuitive on an interactive and visual manner. Add the "Buttons", and objects to display information, such as text labels ("Label"), images ("graphicsView"), pictures or text editing boxes ("LineEdit") bars, vertical and horizontal scroll ("ScrollBars"), etc.
2. Specify or set the properties of the individual components. For example, change the size of objects, colors in a way, the font of a label, the background color of a tab, etc.
3. Develop and implement the code to execute when an action is performed on a component. For example make a "click" on a button tab.

4.1.5 Specification of the Laboratory "Hello World"

Let's create an application. The specification of this lab is:

"Make a window that has a button, and when clicked, a message is displayed in another window with the message: "Hello World ".

4.1.6 Development of the Lab: The Project

1.- To get started, open the Qt program from the drop down Windows startup (1) program: QT Creator (2) (Figure 20).

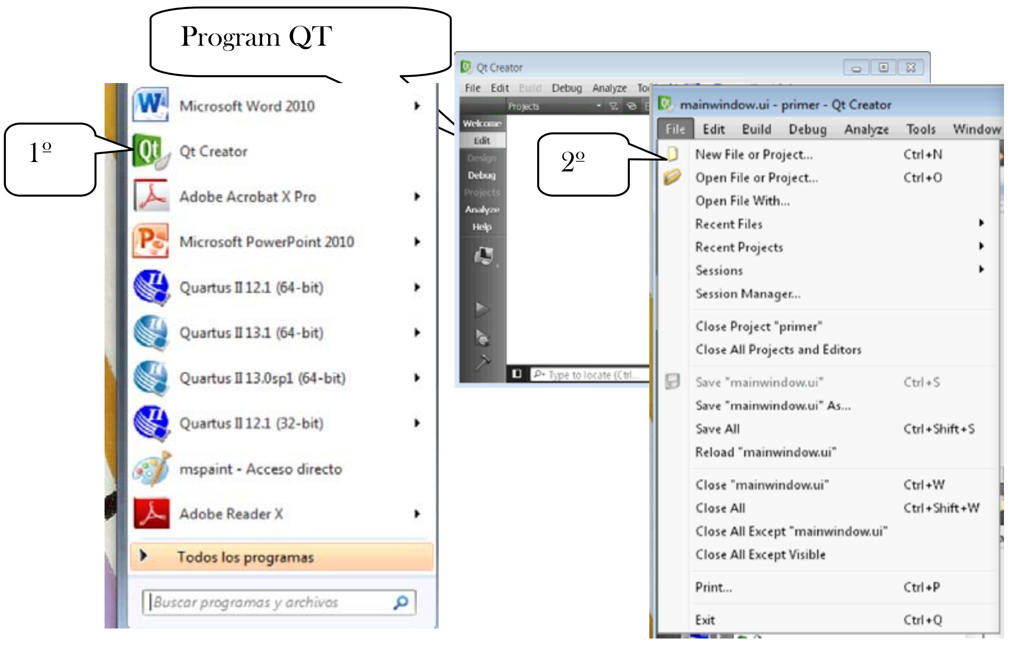


Figure 20: Starting with the project with Qt Creator

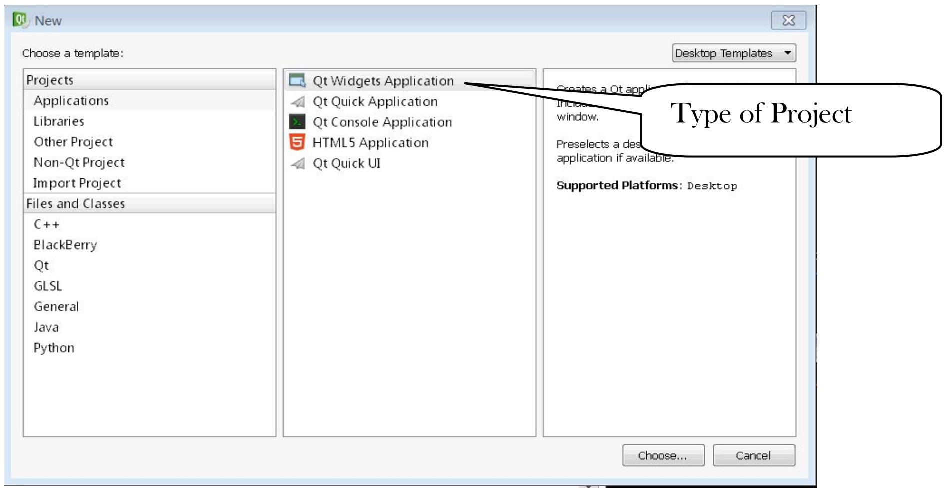
After opening the *QT Creator* **ALWAYS** the first thing to do is **CREATE** a project, to do this follow these steps:

2.- Click the "***File -> New File or Project***" menu, the following dialog box appears. To do this, we must **choose** or **create** an appropriate directory to save the project in it. (If otherwise, we have in Qt Creator, open another project, we close with "*File -> Close All Projects and Editors*."

Very important: Previous step to be performed in each lab: We named the folder of each practice with the order number: lab1, lab2, ..., etc). If we have even created folder practices of industrial computing subject (":\infi"), we can use Windows Explorer and with the mouse button, select the "*File*" menu, the "*Create New Folder*" or we can also be created using the Browser icon “”.To do this, find the directory of the subject "*C:\infi\*". Now, create a new folder, and put the name "Lab1".

3.- Select with the mouse, the new project: "*Applications*", and type of window, which is explained later: "*Qt Application Widgets*".

4.- Press the button: "*Choose*" (Figure 21):



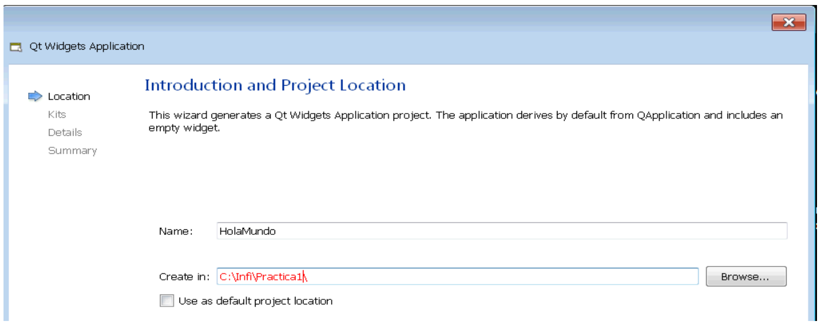
Creates a Qt appl... window.

Preselects a des... application if available.

Supported Platforms: Desktop

Figure 21: Selecting the type of project

5.- We can give the name in the edit box ("*Name*"). For example: *HelloWorld* (Figure 22)



This wizard generates a Qt Widgets Application project. The application derives by default from QApplication and includes an empty widget.

Name: HolaMundo

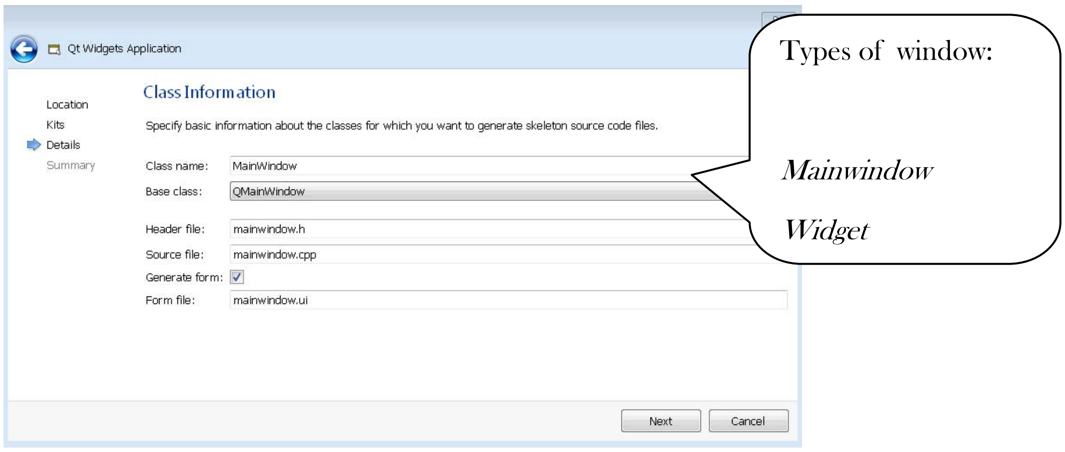
Create in: C:\Infi\Practica1\

Use as default project location

Figure 22: Project location and name

6.- If an error occurs. That is, it appears in **red** the edit box where the project is located, "*created in: C:\ infi\lab1\*", is because **you have not created the appropriate folder**. Then, you must return to the previous section, "*2*", and do all the steps as noted. And if there are no errors, continue!

7.- In the dialog box concerning the application type "*Qt Widgets Application*", we can select the window type (class) that we will use: "*MainWindow, widget or dialog*"(Figure 23). Select the default type is "*mainwindow*". Press the button below ("*Next*"), down to the last window, which shows all the files that your application (Figure 24) will have to be described in a little more detail below.

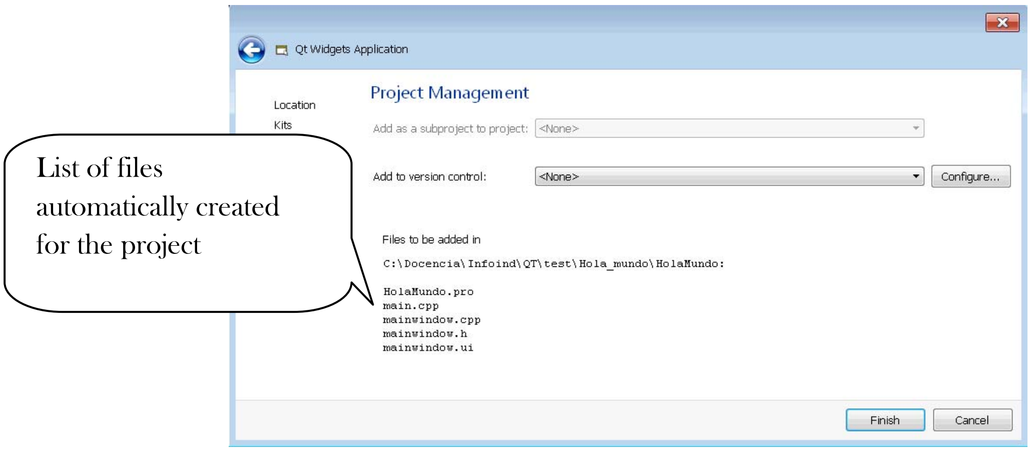


Class Information

Specify basic information about the classes for which you want to generate skeleton source code files.

Class name: MainWindow
Base class: QMainWindow
Header file: mainwindow.h
Source file: mainwindow.cpp
Generate form: ✓
Form file: mainwindow.ui

Figure 23: Class Information



Project Management

Add as a subproject to project: <None>

Add to version control: <None>

Files to be added in

```
C:\Docencia\Infoind\QT\test\Hola_mundo\HolaMundo:

HolaMundo.pro
main.cpp
mainwindow.cpp
mainwindow.h
mainwindow.ui
```

Figure 24: Files automatically created for the project

Now it is about to save the project properly!!

8.- Select the end: "*Finish*"

4.1.7 Classes, Objects, Properties and Events

According to the specification, we need two graphic objects:

- A window
- One button

Window

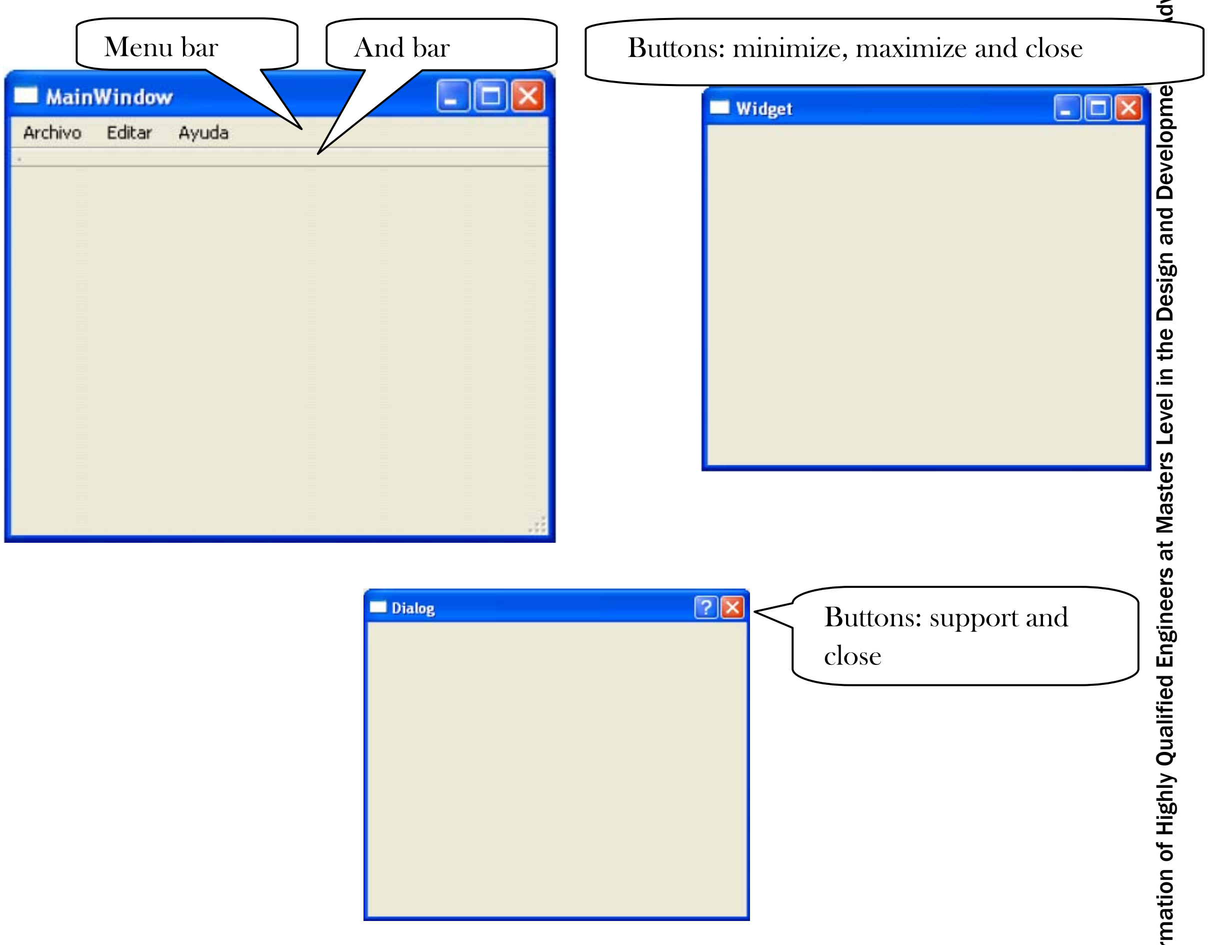
As we have seen above, Qt, windows can be of three types: "*mainwindow, widgets, or dialog*". The characteristics of each are:

The main window (or *mainwindow*) is a window that has a top menu bar and status bar below, plus buttons to minimize, maximize, and close ( ). It is interesting that for any of these three actions on the window, we must implement any code, since these functions are intrinsic to the windows operating system, and we are given (or inherited).

The window type "*widget*" is an empty window without menus or status bar, but with Windows buttons, maximize, minimize and close (,).

The window "*Dialog*" is a window with only the help buttons and close ( ).

Based on the specification, we can choose the one that appears by default in the application, ie the type "*mainwindow*" and that allows us, among other things, minimize, maximize and remove window. It can also be considered for this simple application that would have been sufficient to choose a window type "*Widget or Dialog*" (Figure 25):



Menu bar
And bar
Buttons: minimize, maximize and close
MainWindow
Archivo Editar Ayuda
Widget
Dialog
Buttons: support and close

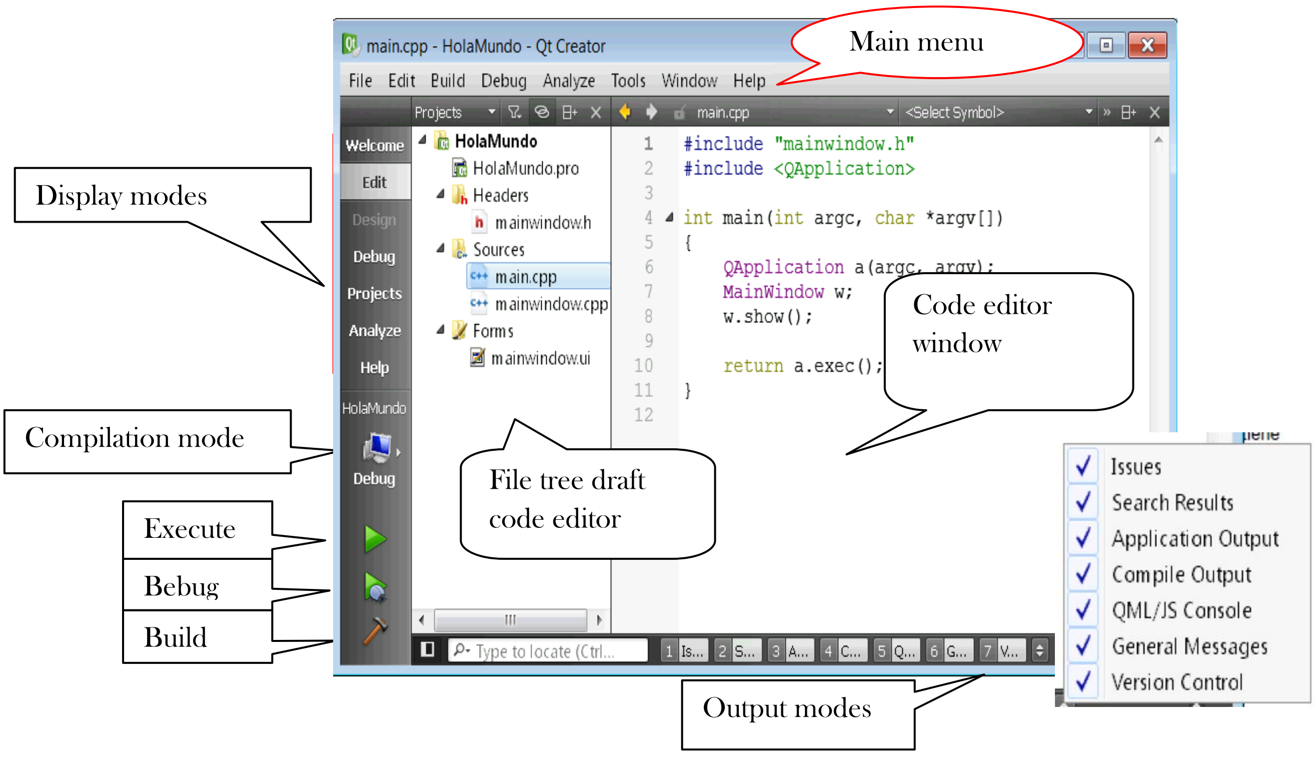
Figure 25: Window selection

Windows are called classes, and in our code will be declared subclass Qt: "**QMainWindow**, *QWidget* or *QDialog*". Observe the contents of the file "*mainwindow.h*".

Let's look at some features of the development environment for graphical applications Qt Creator. When starting the program several windows appear below the main menu (in the figure with a call in red).

The green box window is called workspace. This is divided into two parts: Left, and under "*Projects*" file tree project is located. Each of the items shown in the hierarchy can expand (or compress) simply "*by clicking*" repeatedly on the icon on the left in a triangle: gray (or black).

In the Figure 26 the main file "*main.cpp*" making the application is displayed. Part of projects, different types of files it has generated Qt Creator for these applications are observed:



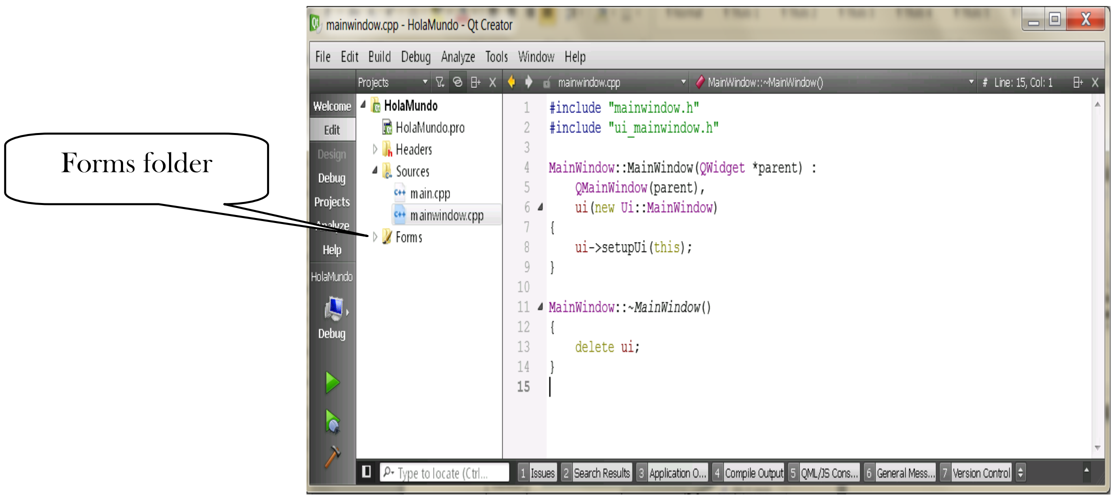
```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Figure 26: Content of the file *main.cpp*

- HelloWorld.pro
- mainwindow.h
- main.cpp
- mainwindow.cpp
- mainwindow.ui

The file "*.pro*" is a text file containing: the chosen type definition window, all files and project elements. Also, the name of the forms of the project (or "*forms*") of header files (or "*headers*" with the extension "*.h*"), the filenames of the implementation (or "*sources*" with extensión.cpp). It may also contain other information such as the path to the files "*include*" ("INCLUDEPATH"), external libraries used (with the extension "*.lib*") and other resource files (or "*resources*").

The forms directory "*Forms*". In this directory all forms (windows) containing the application is included. The application can have one or more windows, but only one can be the main window, which is shown to run the program.



```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

The file in the main window: "*mainwindow.ui*". It is the main application window. Viewports always have extension "*ui*". Are text files in XML (eXtensible Markup Language) is a language extension of marks adopted by the consortium "WWW" Internet, which is used to exchange information between applications of any kind. In Qt is used for the description of the graphical elements inserted in the window, such as buttons, labels, menus, etc. These files can be edited to implement a graphical object manually, but requires knowledge of XML, which is outside the scope of this course. In our case we use the graphical components of the palette editor windows Qt Creator in design mode: "*Design*".

Directory header files: "*Headers*". Within this directory are all header files (or "*headers*" which have the extension "*.h*"). These files are definitions of classes, constants, variables and public procedures. There are generally two types of headers: An associated with the windows, and other associated non-graphical modules.

The header files "*mainwindow.h*". Contains the declaration of the main window of our application. Our windows are classes, subclasses declared "*QMainWindow*", "*QWidget*" or "*QDialog*". Observe the contents of the file "*mainwindow.h*".

The "*mainwindow.ui*". File This is not C/C++ but XML code that is compiled to automatically generate the file "*ui_mainwindow.h*" where a class is defined: "*Ui::MainWindow*", which is containing buttons, labels, etc. Then, within "*mainwindow.h*" the "*MainWindow*" class that includes an attribute is defined, (a pointer to the "*ui*" window): "Ui :: MainWindow * ui".

The source files folder "*Sources*". Within this directory is implementing classes and program functions. Contains C++ code files (with the extension "*.cpp*"). They may be associated with windows or modules that are not graphics.

File main window: "*mainwindow.cpp*". Here is where you implement the functions (or "*slots*"). These functions are called methods and are associated with events that may occur on an object. The structure of these functions is automatically generated in the vicinity of Qt. The designer must complete the implementation of the code you want to perform the function: for example, that when you "*click*" with the button sale appears displaying a text. The designer uses this file to implement any other method you want to add to the class. Also, if needed, you can define procedures outside the classroom.

Main application file "*main.cpp*". It is the main program application, where the main function or procedure "*main*" is. This code is generated automatically. We can visualize a double "click" on the file, and what it contains is:

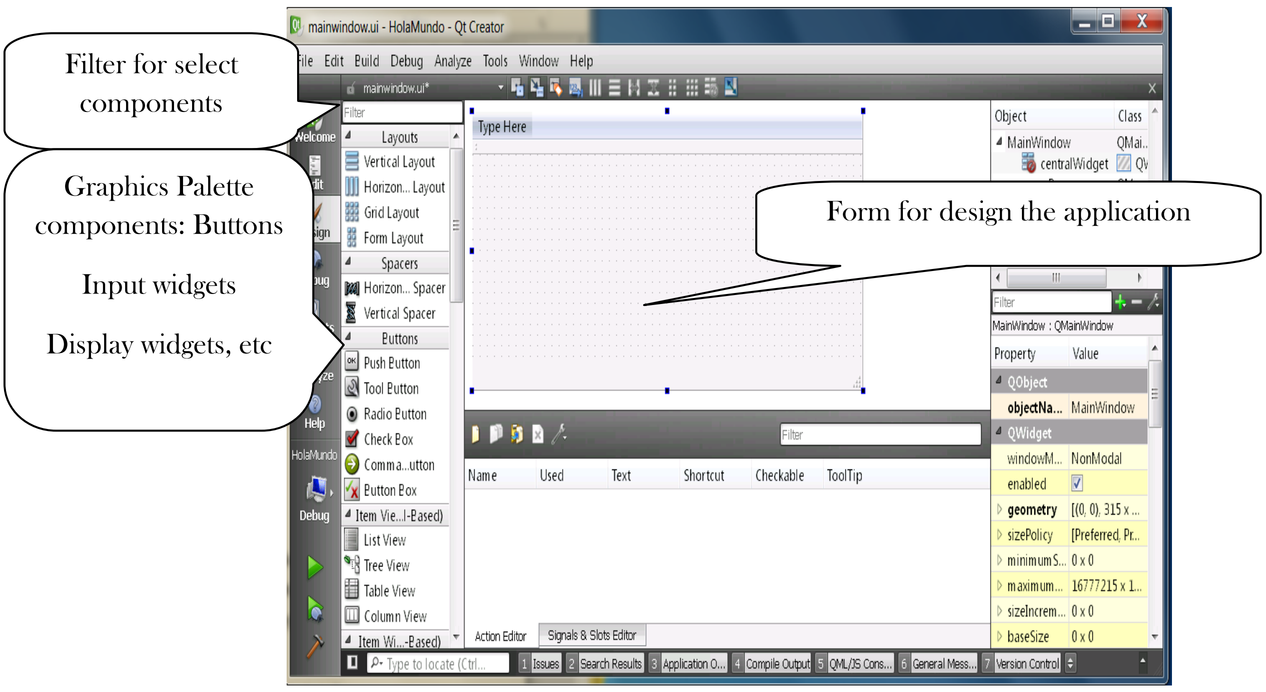
An application "to": ("*QApplication to*"), which uses a window "w" ("*MainWindow w*"), then displays the "w" window ("*w.show ()*"), and finally running the "a" ("*return a.exec ()*").

The button

1. Select in the project tree form in directory "*Forms*", expand the hierarchy on the triangle icon with a "*click*" the mouse. The file "*mainwindow.ui*" appears. Select the window design work in design mode: "*Design*". We describe some elements of the design window. In the left column palette graphic components, which are used for placing on the form is. Among others are the different types of buttons "*Buttons*", and other objects for user interface input/output "*Input Widgets*" or "*Display Widgets*", etc.

   To the right is the form, also known record of graphic design or canvas. The form is where the graphic objects palette of components that wish will be placed. To do this, a graphical component of one of the lists is chosen, and holding the left mouse button, drag it over the form and dropped into the desired position, press the left mouse button. All objects put on this tab will be those who display our application as placing the go. Pushbutton, *radioButton*, *ToolButton*, link command button, etc.: The different components of the graph palette, for example in the "*Buttons*" list different types of buttons appear. To select the button that pressed either we choose from the list of "Buttons": "*Push Button*", select it with the mouse and place it on the record, and place where we see fit. Or you can also choose by component selection filter that is located above the component palette.

2. To the right of the form, there are two more windows. In the upstairs window where objects and classes ("*Object*, *Class*") are shown, they are appearing each object we deposited on the form. Column object "*Object*" We can see each other, the object just put on the form: "*Pushbutton*" belonging to the "*QPushButton*" class (Figure 27).



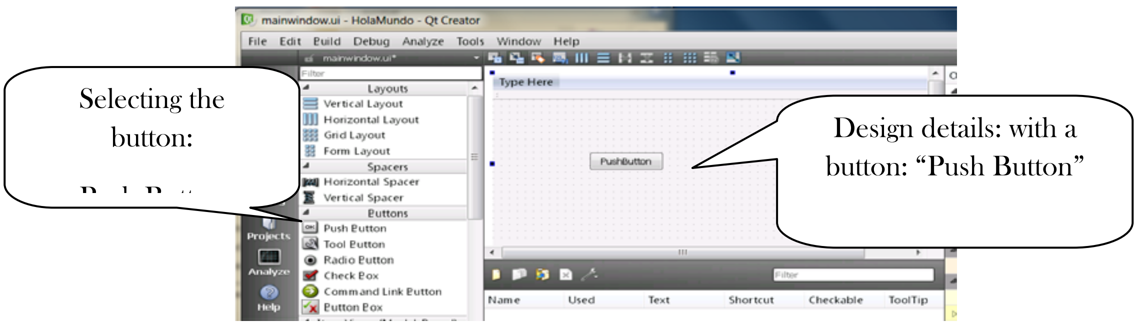


Figure 27: Windows in the form

3. The properties of an object (Figure 28) such as size, text, etc., can be modified in two ways: at design time or at runtime program:

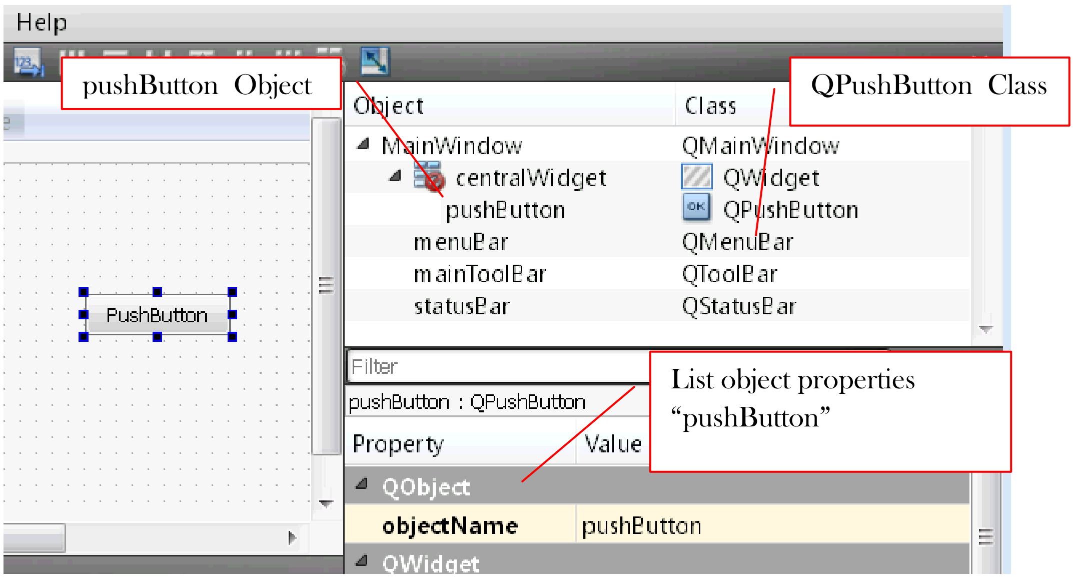


Figure 28: Properties of an object

- At design time: We can change the values of the properties of an object at the time of design in two ways. By selecting the object and displaying the menu dialog, select the property to modify. The other way is through the list of object properties. For example, in the case of the button, change the text using the property called "*text*" and in the edit box in the column "*Value*", we can change the text that is given by default object ("*PushButton*"), with a new one, for example: "Salute" (Figure 29), we see that it changes at the same time we do this action design.

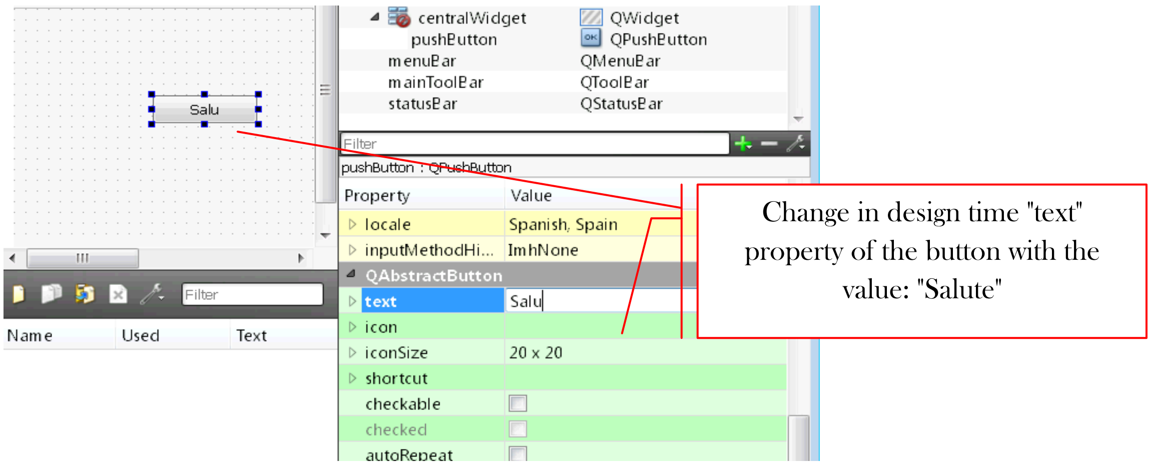


Figure 29: Changing text property

- At run time: We can change the properties for a program at the time of execution. For this we associate that action to an event. To do this we have to write a few lines of code inside the function (also known as method or Qt by "slot"), which establishes event (or "signal") associated with the object, in this case the button.

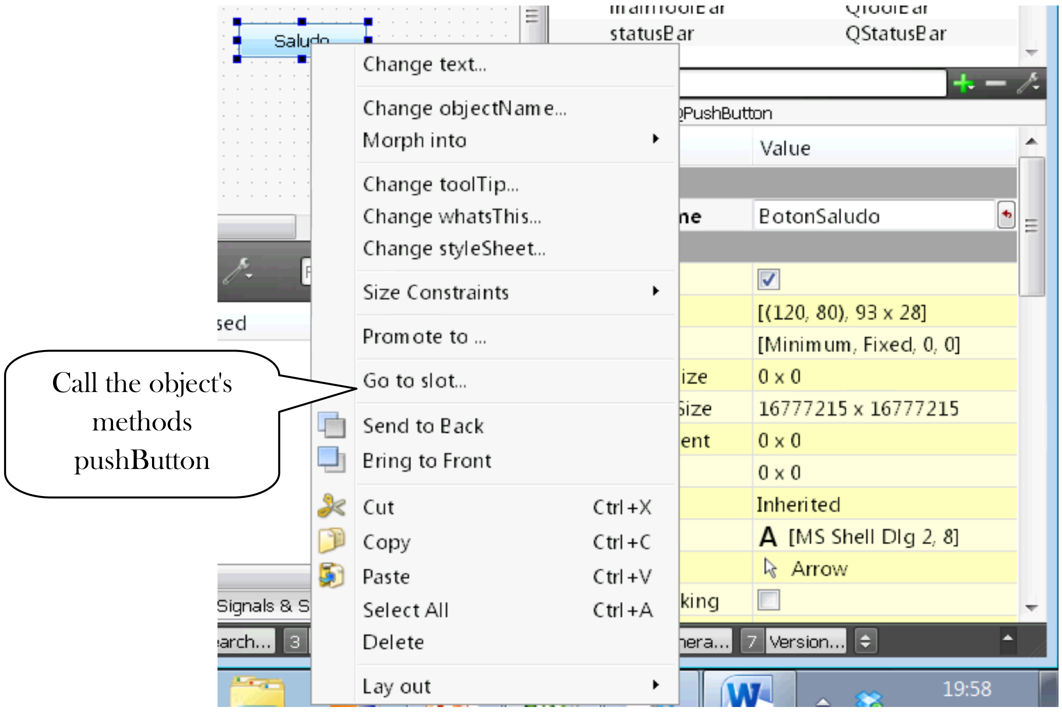
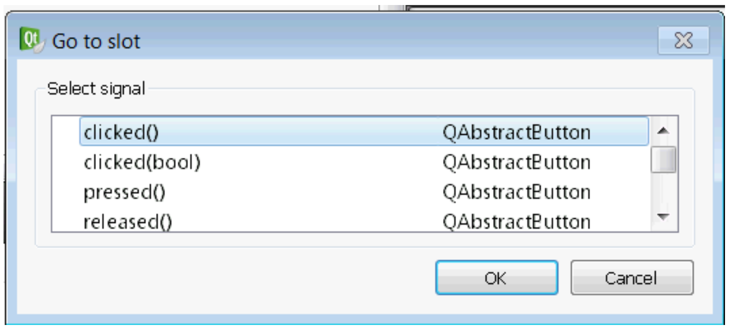


Figure 30: Object´s methods

We want to "click" on the button, a message "Hello World" is displayed. To do this we have to refer to the associated button method. Select the button on the design tab and clicking the right mouse button, a menu appears. We chose "Go to slot" (Figure 30). This action opens a dialog.

Implementation Code

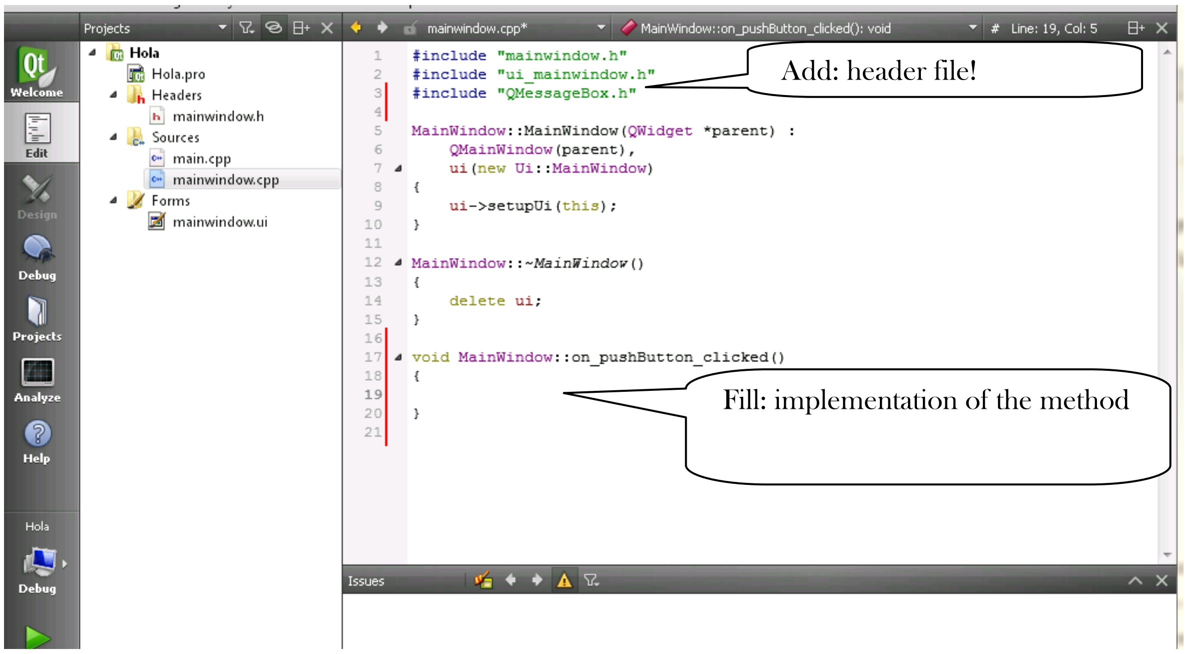
The window (or “Dialog”) appears all signs that can be associated to the source object, in this case with the button shown. Now, select the associated event "click" signal (which is listed as "*clicled ()*") (Figure 31). This will be the event to be associated with the object "*PushButton*".



Go to slot
Select signal
clicked() QAbstractButton
clicked(bool) QAbstractButton
pressed() QAbstractButton
released() QAbstractButton
OK
Cancel

Figure 31: Signal selection

Now any function (or "slot") that we use, and to be associated with any of the objects of design, we instantiated by declaring its prototype. This is done by including the header file in the source object in the main window. That is, should be included in the file source ("*mainindow.cpp*") where we have inserted the object (button), the header file "*QPushbutton.h*" (Figure 32).



```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "QMessageBox.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_pushButton_clicked()
{

}
```

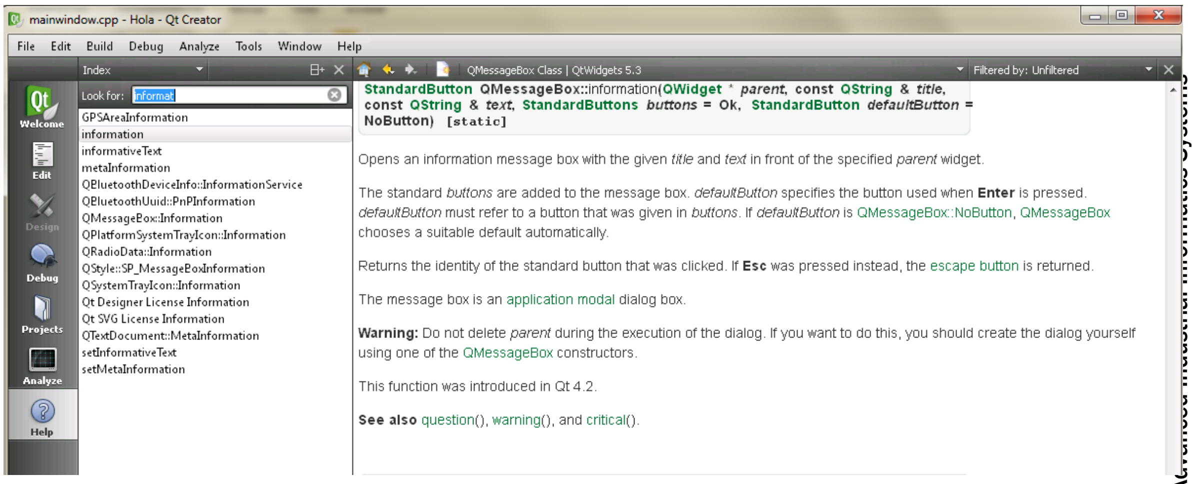
Figure 32: Header file and Method

- The code added between the two braces ("{" "}") of the event "*on_pushButton_clicked ()*" is another function that implements a text message. This function is of the "*QMessageBox*" and the method is called, "*information*", and has several parameters separated by commas (Figure 33). All methods that we use for graphics components can be obtained by a call to the help of the program with the "*F1*" key.

```

void MainWindow::on_pushButton_clicked()
{
    QMessageBox::information(this,"Message","Hey world: QT is comming!!");
}
```

Figure 33: Code for *QMessageBox* function

Pressing the "F1" key, which shows the help (Figure 34):


mainwindow.cpp - Hola - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Index
Look for: informat
QMessageBox Class | QtWidgets 5.3
Filtered by: Unfiltered
Welcome
Edit
Design
Debug
Projects
Analyze
Help

GPSAreaInformation
information
informativeText
metaInformation
QBluetoothDeviceInfo::InformationService
QBluetoothUuid::PnPInformation
QMessageBox::Information
QPlatformSystemTrayIcon::Information
QRadioData::Information
QStyle::SP_MessageBoxInformation
QSystemTrayIcon::Information
Qt Designer License Information
Qt SVG License Information
QTextDocument::MetaInformation
setInformativeText
setMetaInformation

```
StandardButton QMessageBox::information(QWidget * parent, const QString & title,
const QString & text, StandardButtons buttons = Ok, StandardButton defaultButton =
NoButton) [static]
```

Opens an information message box with the given *title* and *text* in front of the specified *parent* widget.

The standard *buttons* are added to the message box. *defaultButton* specifies the button used when **Enter** is pressed. *defaultButton* must refer to a button that was given in *buttons*. If *defaultButton* is QMessageBox::NoButton, QMessageBox chooses a suitable default automatically.

Returns the identity of the standard button that was clicked. If **Esc** was pressed instead, the escape button is returned.

The message box is an application modal dialog box.

Warning: Do not delete *parent* during the execution of the dialog. If you want to do this, you should create the dialog yourself using one of the QMessageBox constructors.

This function was introduced in Qt 4.2.

See also question(), warning(), and critical().

Figure 34: Help information

You are now ready to run the application. !!

4.1.8 Executing the Application

1. Save the project with the option in the dropdown menu: "*File→Save All*" (Figure 35)

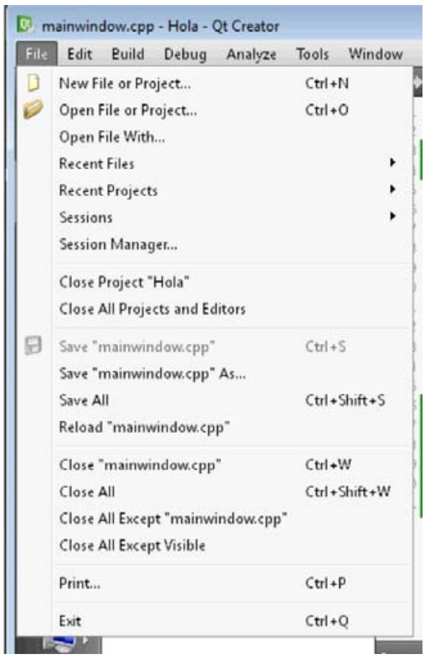

mainwindow.cpp - Hola - Qt Creator
File Edit Build Debug Analyze Tools Window
New File or Project... Ctrl+N
Open File or Project... Ctrl+O
Open File With...
Recent Files
Recent Projects
Sessions
Session Manager...
Close Project "Hola"
Close All Projects and Editors
Save "mainwindow.cpp" Ctrl+S
Save "mainwindow.cpp" As...
Save All Ctrl+Shift+S
Reload "mainwindow.cpp"
Close "mainwindow.cpp" Ctrl+W
Close All Ctrl+Shift+W
Close All Except "mainwindow.cpp"
Close All Except Visible
Print... Ctrl+P
Exit Ctrl+Q

Figure 35: Options in the dropdown menu

2. Run the program from the main menu: "*Build* → *Run*" or with the program icon ("▶"), or with the keys "*Ctrl* + *R*" (Figure 36).

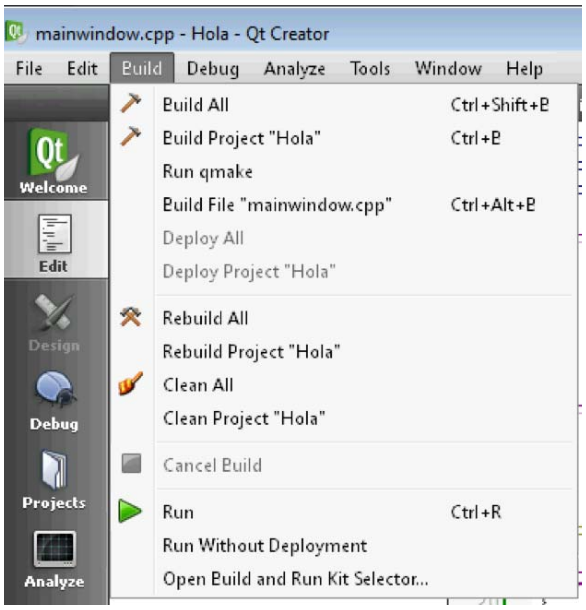


Figure 36: Running the program

3. What we displayed is the result of our application: A window with a button, and when we "*click*" with the mouse on the button "*Salute*" gives us the output in another window ("*QMessage*") with the message we wrote: "Hey world ..." (Figure 37):

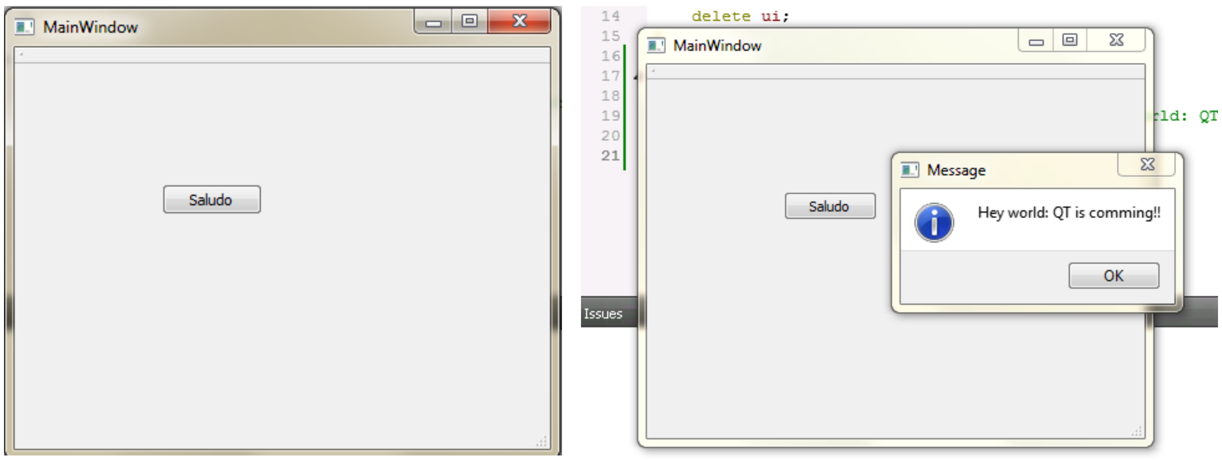


Figure 37: Button and message shown

4.1.9 Exercises and Activities

1. Repeat the above steps and add another button that takes a different message. Change the following properties of the button: Name, Caption and Font.
2. Change the font of the title of the button (example Arial 12, Bold-Italic)
3. Remove the button corresponding to the first function entered, and subsequently removing the button of the form, the application goes only added in the previous activity button. Then save, compile and run the project.
4. Close Qt and search with Windows Explorer the folder containing the project. Double-click the executable to test it out.
    - Check the disk space occupied by the project files. Remove any leftover files and check back the occupied space.
    - Drag the folder with the mouse to the icon of the USB disk to be copied into it. Rename the folder to diskette Practica1_v2. Remove the hard drive folder.
    - Go to the adjacent computer diskette, drag the folder from the USB disk to the hard disk and double click on the "*.pro*" file. Qt opens with the project. Run the project to make sure everything works well.

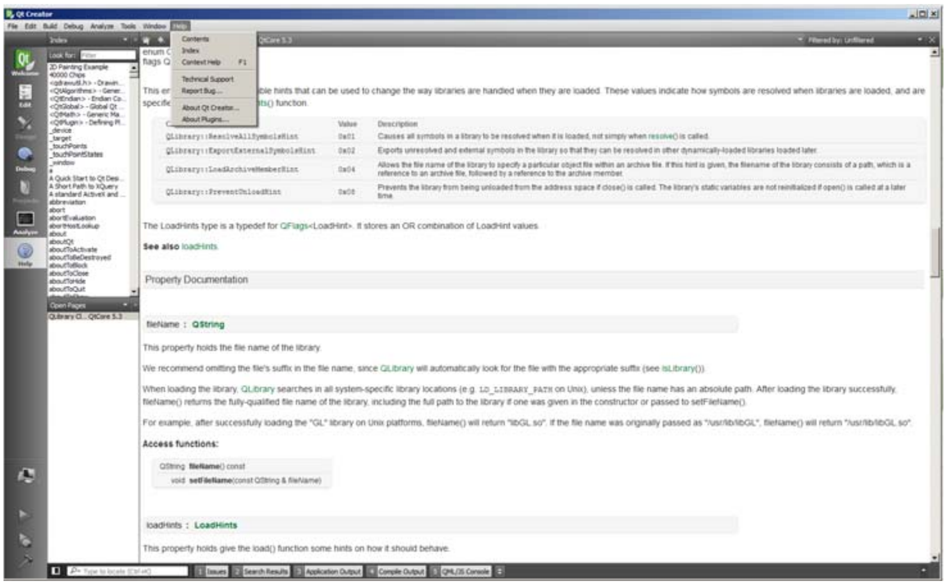
5 Seminar

The objective of this seminar is to better understand the development tool that we are using in this course: *Qt Creator*. Also we will meet other development tools and comparing them with that we are going to use this course.

The seminar is organized for students to work in groups in order to obtain the requested information and then present their results to the rest of groups.

It is proposed to break the class into groups of maximum size of 5 students. Then the groups will work on the environment **Qt Creator** on the following topics:

Use the Help environment that provides the tool for ease of use to programmers. Subsequently shown in Figure 38how you can access help. Student groups will tour the different themes that offers support tool



This er… …ble hints that can be used to change the way libraries are handled when they are loaded. These values indicate how symbols are resolved when libraries are loaded, and are specifie… …ts() function.

	Value	Description
QLibrary::ResolveAllSymbolsHint	0x01	Causes all symbols in a library to be resolved when it is loaded, not simply when resolve() is called.
QLibrary::ExportExternalSymbolsHint	0x02	Exports unresolved and external symbols in the library so that they can be resolved in other dynamically-loaded libraries loaded later.
QLibrary::LoadArchiveMemberHint	0x04	Allows the file name of the library to specify a particular object file within an archive file. If this hint is given, the filename of the library consists of a path, which is a reference to an archive file, followed by a reference to the archive member.
QLibrary::PreventUnloadHint	0x08	Prevents the library from being unloaded from the address space if close() is called. The library's static variables are not reinitialized if open() is called at a later time.

The LoadHints type is a typedef for QFlags<LoadHint>. It stores an OR combination of LoadHint values.

See also loadHints.

Property Documentation

fileName : **QString**

This property holds the file name of the library.

We recommend omitting the file's suffix in the file name, since QLibrary will automatically look for the file with the appropriate suffix (see isLibrary()).

When loading the library, QLibrary searches in all system-specific library locations (e.g. `LD_LIBRARY_PATH` on Unix), unless the file name has an absolute path. After loading the library successfully, fileName() returns the fully-qualified file name of the library, including the full path to the library if one was given in the constructor or passed to setFileName().

For example, after successfully loading the "GL" library on Unix platforms, fileName() will return "libGL.so". If the file name was originally passed as "/usr/lib/libGL", fileName() will return "/usr/lib/libGL.so".

Access functions:

```
QString fileName() const
   void setFileName(const QString & fileName)
```

loadHints : **LoadHints**

This property holds give the load() function some hints on how it should behave.

Figure 38: Qt Creator Help Menu

Besides the environment Qt Creator provides a collection of already developed projects as examples to facilitate project development comprehension for the programmers (Figure 39)

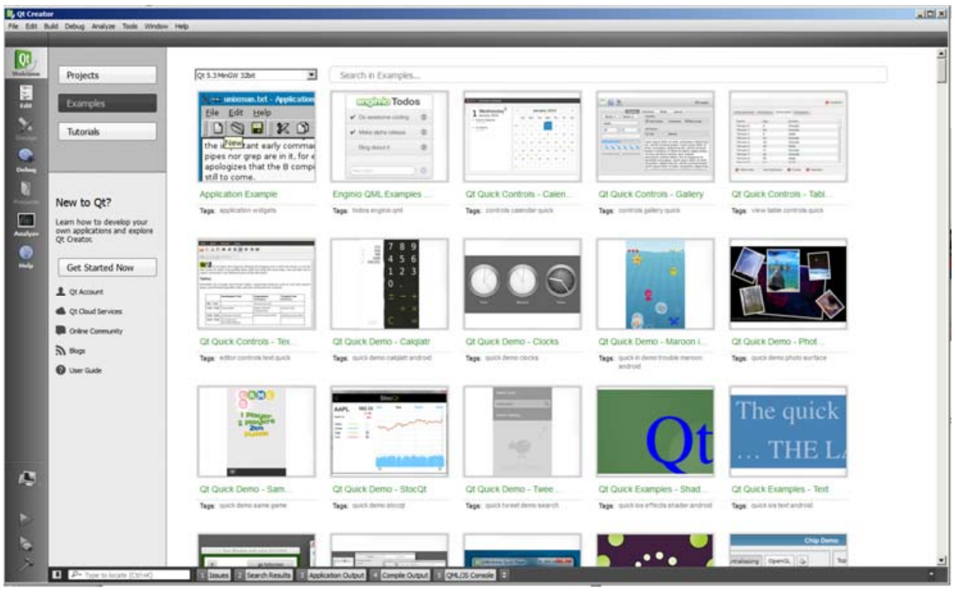


Figure 39: Project examples provided by Qt Creator

Furthermore it is proposed to groups of students to research the concepts and differences between "*Release*" and "*Debug*", showing its findings to other groups

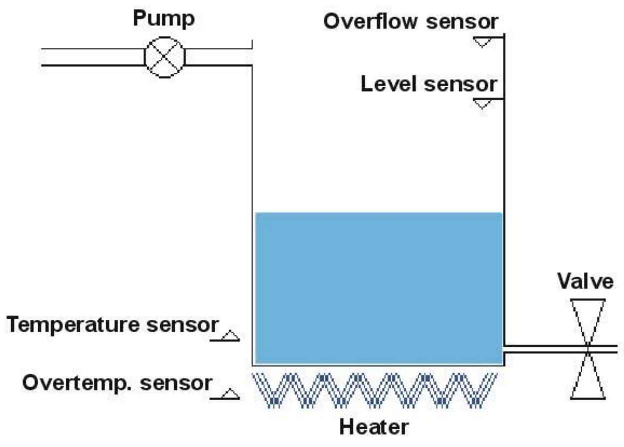
It should also be investigating elements used in the debugger. For example: Stepping, breakpoints, display values of variables and other possibilities to be discover

Should also work on selecting and configuring a kit

Once we have worked on the Qt Creator tool, it is proposed to search the internet similar tools, both as freeware and paid.

In search will be conducted online and once located should get their benefits and then be able to present to the other groups a comparison of the results obtained

6 Mini-project



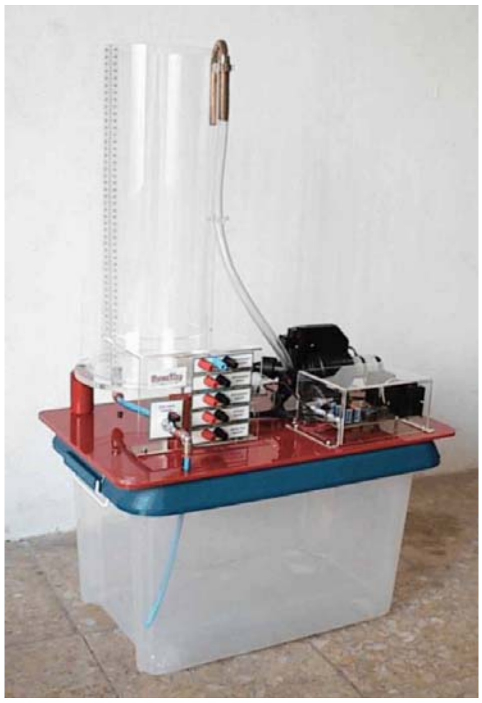


Figure 40: Diagram of the process

6.1 Requirements

The main objective of this part of the course is the programming of the control system and monitoring system, to be run on a personal computer type PC.

6.1.1 Minimum Functional Requirements

Minimum functional requirements that must be met are:

- The control must consider all the sensors and actuators specified in **¡Error! No se encuentra el origen de la referencia.**. It must be able to automatically maintain the temperature and level values desired by the application user.

Signal	NI USB-6008 Connection
Overflow	P0.0
Overtemperature	P0.1
Electrovalve	P1.0
Heater	P1.1
Temperature	ai0, GND reference
Level	ai1, GND reference
Pump	ao0

Table 2: Signals connections between process and data acquisition board

In any case, it shall submit the minimum requirements or expanded, and for each group will be evaluated:

1 The source code that will be developed in C ++ language, the resulting software must run on a PC type computer with Microsoft Windows operating system.
2 The process interface must be programmed.
3 The user interface must be programmed, using the basic elements of the computer interface (screen, keyboard and mouse) and will be able to establish the different parameters and will display the status on real-time.
4 Also all the remaining modules that may be required to implement the system must be designed and programmed.
5 All the development of the project and the result must be reported in a formal document.
6 At the end the project will be exposed including a practical demonstration of the result.

6.1.2 Expansion Requirements

To improve the mark, every aspect of the project can be improved. As improvement ideas a number of guidelines are listed below:

- Design and build the hardware to connect to the data acquisition card.
- Improving control loop considering more complex scenarios such as heating cycles/download, doses, etc.
- Improve user interface by incorporating graphical representations of process status, historical data of temperatures, relative humidity, etc.
- Record the time evolution, permanently storing values in files.
- Allow configuration of different operating modes (manual, automatic, etc.)
- Manage other sensors/actuators in order to improve the system.
- Use automatic documentation generating systems such as doxigen.
- Use a software repository on the Internet (sourceforge, gitorius, github, etc.).
- Etc.

7 Bibliography

Industrial PC, http://en.wikipedia.org/wiki/Industrial_PC

H. Wortmann, H.S. Jagdev, Computers in Industry, Elsevier, ISSN: 01663615

Raj Kamal, *Embedded Systems: Computer Architecture, Programming and Design*, Ed: McGraw Hill, 2010, ISBN: 9780070667648 0070667640

Irv Englander, *The Architecture of Computer Hardware, Systems Software, and Networking: An Information Technology Approach*, Ed: Wiley, 2014, ISBN-13: 978-1118322635

Wikibooks, *Introduction to Computer Information Systems/Information Systems*, http://en.wikibooks.org/wiki/Introduction_to_Computer_Information_Systems/Information_Systems

Krishna Kant, Computer-Based Industrial Control, Ed: PHI Lerning, 2010, ISBN: 812031123X, 9788120311237

Mohamed Khalgui, Olfa Mosbahi and Antonio Valentini, *Embedded Computing Systems: Applications, Optimization, and Advanced Design*, IGI Global, 2013, ISBN: 9781466639232|

Jim Ledin, *Embedded Control Systems in C/C++*, Jim Ledin, 2010, ISBN: 1578201276

Qt Project, http://qt-project.org/