# easyLadder

The PLC engine for your **Raspberry Pi**

HMI library and sample

## Contents

## 1. Introduction

When building the easyLadder PLC system, you probably will need some kind of Human Machine Interface (HMI) to monitor PLC status, operate manual PLC commands, configure working parameters or manage process data (logging, graphs and so on). For this purpose, you can use any available industrial HMI providing a MODBUS TCP driver. Nevertheless, when creating cost-sensitive applications or having special HMI requirements, the best solution is to develop your own HMI application using a powerful language like C++, executed on the same Raspberry Pi. Doing so, you get an embedded HMI/PLC system with unlimited possibilities. This library is provided to help you in this progress.

**easyLadderHMI** is a library used to build your own HMI application to interface with easyLadder PLC. Using this library you can easily develop the HMI interface for your PLC program using the power of the Qt platform and C++ language, without worrying about knowledge of easyLadder PLC communication internals.

The sample provided is designed for the official Raspberry Pi 7" touch screen, but can be easily translated to other LCD. In this case, it connects to the local easyLadder engine (127.0.0.1), but it can also connect to a remote PLC simply changing this IP in the source code.

Using the Qt platform, this library can be compiled in any Linux platform, or even in Windows machines with small modifications. So it is possible to get your HMI working remotely in any desktop PC, for example.

## 2. Compiling sample in your Raspberry Pi

These instructions will let you to download the library and compile the sample using your Raspberry Pi with the official Raspbian Jessie (LITE edition can be used, since no X11 is required). This sample is designed for the official 7" touch screen.

Please note that the optimal development solution is to setup a cross compiler in your desktop computer, and transfer the compiled target to the Raspberry Pi system. Doing this requires some time since it is required to compile the Qt from source code, and it is not covered in this manual. This manual compiles the sample Hmi locally using the Raspberry Pi. There are a number of tutorials about cross compiling available in Internet.

First, download and extract the library:

```
cd
wget http://www.ferrariehijos.com/easyladderhmi.tar
tar -xvf easyladderhmi.tar
```

A directory called **easyLadderHmi** with the library and sample files will be generated in your home directory.

The next step is to install the Qt development package. The official Jessie repository includes a version of Qt but, at this time, it is not optimized for the Raspberry Pi video hardware, and requires the use of the X11 windowing system. We will try to install Qt from an alternative repository (apt.leandog.com). This Qt includes EGLFS support for video acceleration and does not requires X11.

Follow these instructions to install Qt in your Raspberry Pi:

```
sudo ln -s /opt/vc/lib/libEGL.so /usr/lib/arm-linux-gnueabihf/libEGL.so.1.0.0
sudo ln -s /opt/vc/lib/libGLESv2.so /usr/lib/arm-linux-gnueabihf/libGLESv2.so.2.0.0
echo "deb http://apt.leandog.com/ jessie main" | sudo tee --append /etc/apt/sources.list
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys BDCBFB15
sudo apt-get update
sudo apt-get install -y qt5
```

Now, we are ready to compile the sample:

```
cd easyLadderHmi
qmake
make
```

Once compiled, we can run the sample HMI.

```
./easyLadderHmi
```

Now you are ready to modify the source code and develop your own HMI application.

# 3. PLC COMMUNICATION FRAMEWORK

easyLadderHmi communication library is offered as a project include **easyLadder.pri**, located in the easyLadder directory. You must include this file in your .pro project. This library contains all classes needed to interchange information with the easyLadder PLC. All operations are done in the background using a different thread. You do not need to know details about these internals to build your program, but source code is provided if you need to modify it according to your needs.

easyLadder PLC communication is managed through class **easyLadderPlc**. To start communications simply create an object with this class and call **init** function with the IP address of the PLC. This class provides two signals to monitor the communication engine. Signal **connected (bool Connected)** is emitted when connection is stablished (**Connected=true**) or lost (**Connected=false**). Signal **refreshEnd ()** is emitted after each refresh cycle, so you can update values in the HMI when receiving this signal.

PLC device values are managed through class **easyLadderDevice** and children of this class:

- **easyLadderBitDevice**
- **easyLadderWordDevice**
- **easyLadderSignedWordDevice**
- **easyLadderDwordDevice**
- **easyLadderSignedDwordDevice**
- **easyLadderFloatDevice**
- **easyLadderDeviceBlock**

If you need to get a device value, you must create an object of the desired class, according to the type of device needed, and specify the corresponding PLC object, a friendly device name and the device name in the PLC (D0, M10, K4M0...), in addition to other optional parameters. When object is created, value refresh is done automatically in the background. When object is destroyed, refresh is stopped. PLC refreshing algorithm is intelligent: when you create several device objects referring to the same PLC device, only one device will be queried to the PLC.

When creating a PLC device object, you can specify additional parameters.

Parameter **decimals** is provided to manage value display and edition. For example, when using an **easyLadderWordDevice** over D0 with **decimals** parameter to 2, when D0 is 1234, this device will be displayed and edited as 12.34.

Parameter **refreshRatio** controls the desired refresh timing for the device. When setting a value of 0 (default), this variable will be refreshed on every communication scan. When setting a value of 4, the device will be refreshed skipping 4 scans. A value of -1 indicates manual refresh. In this case, the device will be refreshed using the function **easyLadderDevice::refresh()** and you can query **easyLadderDevice::isRefreshed ()** to know when refresh was accomplished.

# 4. SCREEN MANAGEMENT FRAMEWORK

easyLadderHmi windowing system is based on class **hmiBaseWindow** provided by the library, and your own **hmiWindowX** classes. The sample includes **hmiWindow1** and **hmiWindow2** as a startup point for your own screens. **hmiBaseWindow** is the only real Window in the system, and it is the base for all other windows (HMI screens). Generally, you can use this class without any modification. This window is intended to be blank, so all other windows are created as children of the **hmiBaseWindow** and displayed as regular widgets over this blank window base.

In the sample, the **hmiBaseWindow** is created in the **main ()** function. The **hmiBaseWindow** includes the **easyLadderPlc** object and it is then **init ("127.0.0.1")** in the class constructor.

You need to create as many **hmiWindowX** as needed. Each **hmiWindowX** represents a HMI screen, and contains the required number of **easyLadderDevice** used by the screen. To change the currently visible screen, you must call **BaseWindow->changeToWindow (new hmiWindowX (Plc, BaseWindow))**. Do not care about **hmiWindowX** destruction, since it is done for you in the **hmiBaseWindow** class.

Each **hmiWindowX** includes all **easyLadderDevice** as member variables, and **easyLadderDevice** constructor parameters are specified in the **hmiWindowX** constructor (see samples). When **hmiWindowX** object is deleted, all devices included in this class will be also deleted, so no further device refreshing will be done.

To ease device monitoring and editing, some **hmiWidget** are included in the sample (hmiwidgets.h). You can use these standard widgets to provide automatic device value refreshing and editing. Also, you can use these widgets as a startup point for your own widgets. In order to use these **hmiWidget**, you must create them in the Qt form editor as a normal Widget and promote to the desired **hmiWidget** class. After that, you need to assign the device and desired working parameters using the **hmiWidget::setDevice (...)** function in the **hmiWindowX** constructor. To refresh all **hmiWidget** for the current screen you can use the convenient **hmiWidget::refreshAll (this)** static function in the slot responding to the refreshEnd () PLC signal.

Some **hmiWidget** can be configured to change the StyleSheet depending on the assigned device value. For example, a **hmiLamp** widget can show a different image depending on the set or reset state of the device. You can set these StyleSheet using the function **setStyleList**. Please see the sample source code for details about this and other available functions.

To start with your own application, you can delete **hmiWindow1** files and use the **hmiWindow2** class as a startup skeleton for your own screens. You can copy **hmiWindow2.h**, **hmiWindow2.cpp** and **hmiWindow2.ui** files as **hmiWindowX.h**, **hmiWindowX.cpp** and **hmiWindowX.ui** to begin your own HMI project.

## 5. SUPPORT

For support contact info@ferrariehijos.com or http://www.ferrariehijos.com/easyLadder.

ferrari e hijos, s.a.

http://www.ferrariehijos.com/easyLadder
info@ferrariehijos.com