**Point of Care**

**Invitro Diagnostics**

**Introduction**

This application was created to control the Point of Care Invitro Diagnostics (POCIVD) Demo and was developed in C++ using QT, which is a Cross-platform application development platform. The purpose of this application is for the user (physician) to present their ID badge (RFID card) and then perform an analysis of a test strip. The test strip consists of a patient’s ID in the form of a barcode on one end. On the other end, there are test containing a line labeled “C” or “T”. Depending on the sequence of lines, the test results can either be negative, positive, or invalid.

**Devices**

Reach G3 4.3” PCAB,

- touch screen display and device controller

[4.3" G3MSB Projected Capacitive (Part Number: 51-0603-011) | Reach Technology](https://www.reachtech.com/products/touchscreen-display-modules/4-3-modules/51-0603-011/)

Thing Magic RFID M3e HF/LF RFID Module,

- secure access RFID reading

[ThingMagic® M3e HF/LF RFID Secure Module - Jadak - A Novanta Company (jadaktech.com)](https://www.jadaktech.com/product/thingmagic-m3e-hf-lf-rfid-secure-module/)

JADAK Allegro IVD Penny-Whistle,

- machine vision image processing

[Allegro IVD Smart Camera Series - Jadak - A Novanta Company (jadaktech.com)](https://www.jadaktech.com/product/allegro-ivd-smart-camera-series/)

JADAK VIBE barcode reader,

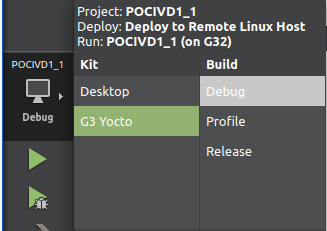
- reading barcode data

[VIBE® Fixed Mount Machine Vision Camera - Jadak - A Novanta Company (jadaktech.com)](https://www.jadaktech.com/product/vibe-fixed-mount-machine-vision-camera/)

**Properties**

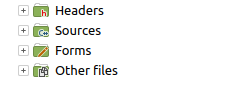
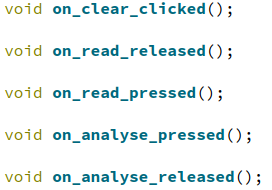
The QT framework was used to take advantage of its cross-compiling ability which allows the development of C++ applications that can be built and hosted on the G3 running a Linux-based OS.

**Cross Compilation to G3 Display:**



The POCIVD consists of three main directories, headers, sources, and forms. The headers directory contains the “**mainwindow.h**” file that declares the functions used in “**mainwindow.cpp**”. The M3e module requires the use of its own SDK, “Mercury API” to fully utilize the RFID module's capabilities. As a result, all necessary header files, along with their corresponding .c files required to control the M3e have been stored in the header and sources folders respectively. See the directories and major functions below:

**Directories:** **Functions:**

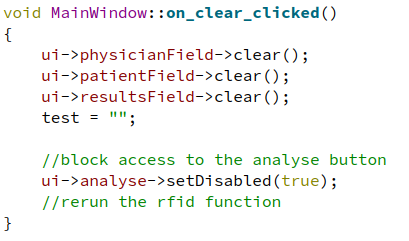
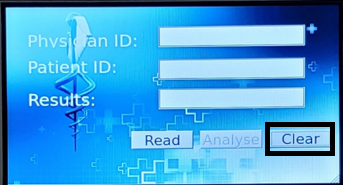
 

Both the Pennywhistle and the Vibe utilize the “**QSerialPort**” class to interface with. In the Linux environment serial ports are recognized in the following format “/dev/ttyxxx#” instead of the windows format “com#”. For the Pennywhistle to process the image for the desired data, a python script was created to identify patterns on test strips used in the demo. A string is then sent to the read buffer for access based on the lines present in the image collected. The version of Mercury API loaded on the M3e used in this application is, ”**mercuryapi-1.37.3.10**”.

**Functions**

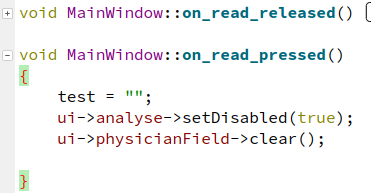
Each of the functions referenced above are used in the GUI displayed on the G3 to perform actions to either the GUI itself or the devices in the demo. The Clear button has one function, “**on\_clear\_clicked**”. The function clears all available fields on the GUI and sets the variable “test” to an empty string, then disables the analyze button.

**Clear Function: Results On GUI:**

The Read button has two functions “**on\_read\_pressed**” and “**on\_read\_released**”. “**on\_read\_pressed”** is responsible for initializing the “**test**” variable to an empty string, disabling access to the analyze button and clearing the physician field. **“on\_read\_released”** is responsible for connecting to and controlling the M3e and processing the data collected from the HF RFID tag presented. For this demo, two tags with the following UID’s “76B2” and “56D6” are used to represent two physician’s "G.McIntyre" and "S.Dixon" respectively. After establishing a connection to the M3e using the Mercury API’s functions, iteration is used to parse the read data to the variable “**test**”, followed by a conditional that compares the UID value obtained and stored in test to either of the possible values. Depending on the presented tags UID, the corresponding physicians name will be poplated and the Analyze button will become accessible.if the UID is not recognized, the Analyze button remains inaccessible and an Invalid User message is output to the physician field.

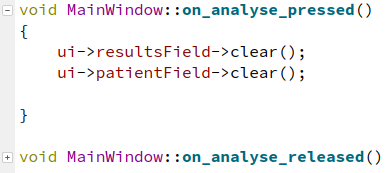
**Read Functions: Results On GUI:**

- note, for “**on\_read\_released**”, please refer to the function definition in “**mainwindow.cpp**”.

The Analyze button has two functions, and, like the read functions utilizes the pressed and released events. “**on\_analyse\_pressed**”, clears both the results field and the patient field. “**on\_analyse\_released**” connects to both the Vibe and the Pennywhistle. The Vibe has a serial command sent to scan the barcode within its line of sight. The data collected is then read from the read buffer for processing. The processed data is either mapped to the corresponding patients name then output, or an Invalid patient message is output. The Pennywhistle is also sent a serial command which directs the Pennywhistle to execute a python script allowing the image collected to be processed. After the image is processed and a valid response is read from the Pennywhistle, one of three messages are styled and output to the results field. Invalid is orange, Positive is red, and Negative is green.

**Analyze Functions: Results On GUI:**

- note, for “**on\_analyse\_released**”, please refer to the function definition in “**mainwindow.cpp**”.