Configurable Radiation Testsuite
-
Documentation

Mattis Jaksch

April 13, 2020

Contents

1 Overview

This testsuite is developed to automate test procedures. It supports different devices (e.g. power supplies, ethernet clients, data acquisition units, ...) which can be configured to control the process in a desired way. The interaction between components is realized with the "qt-signal-slot"[1] mechanism. One may add signals to a device which are then send out to trigger actions on other devices. A trigger-device can be for example a power supply switching off to protect the DUT.

1.1 Mainwindow

Here in figure (1) an overview on the general program is given. The different options are described in the following sub-chapters.

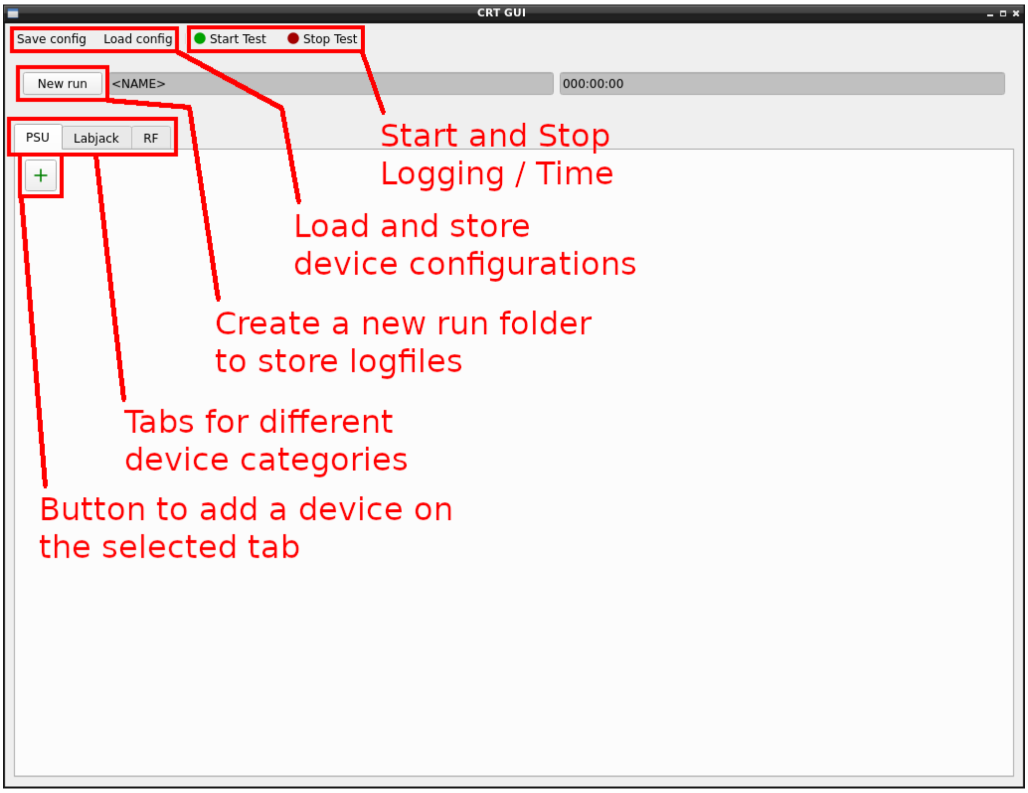

Figure 1: Main menu of the suite

1.1.1 Top Row

On the top row, on the left side, configurations for all the components in the various tabs can be either stored or loaded in a single file. The stored configuration file is also human readable and editable.

On the right side the test can be started or stopped. Starting and stopping the test sends a signal to all components. E.g. the power supply will turn selected channels on/off and start/stop the logging. Hence one can not click the start-button twice in a row!

1.1.2 Mid Row (Run Information)

In the row below is the run information. To start a run one should first create a "New run" by clicking the button and creating a folder to store all the devices log files. On the right of this

[1]Refer to `doc.qt.io/qt-5/signalsandslots.html`

button, the current file location and the time of the run are displayed. If one stops the run, the time also halts till the run gets restarted.

The log files are named after the individual components, so one should make sure to not have any names twice to avoid confusion.

1.1.3 Tabs

The tabs contain various devices of the same kind. Currently implemented are:

- Power-supply
- Labjack
- RF (Reception client for the Analog Devices IIO-daemon)

1.2 Testrun

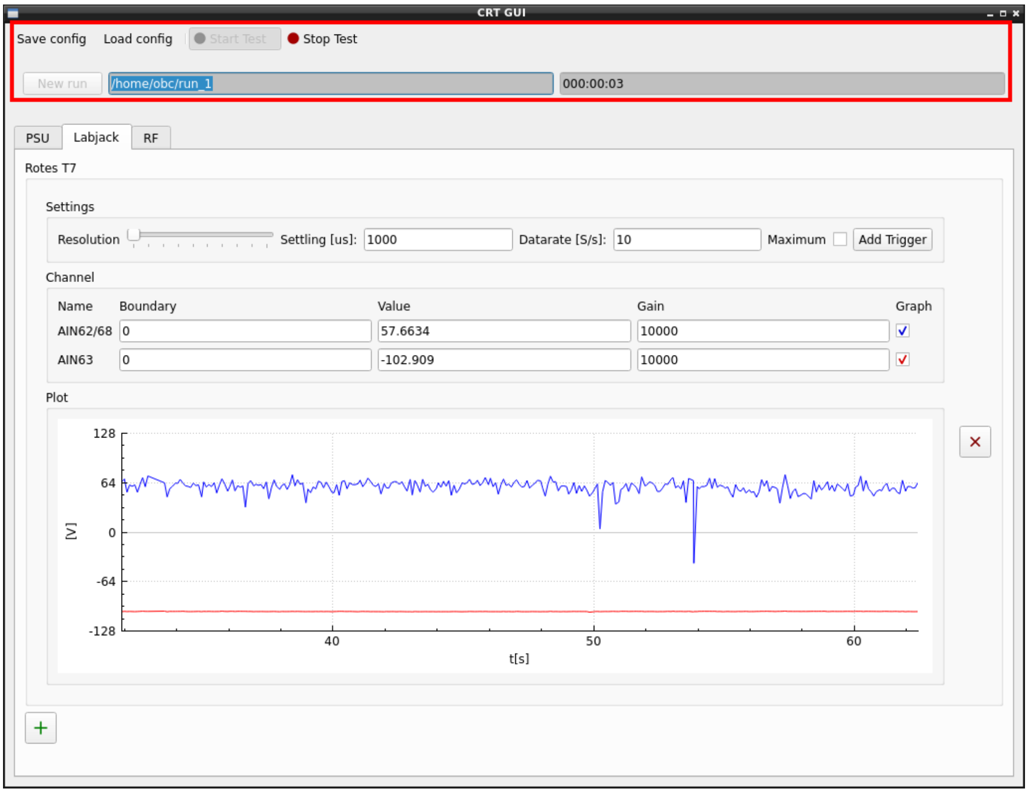

Figure 2: An active testrun in the suite

If a test run is started, the "Start" button and the "New run" button get deactivated and the time starts to count up. Meanwhile in the background, logfiles are generated with a UTC timestamp. The data itself is stored in a *.csv* format to be easily readable. To every datapoint or row being stored, a relative timestamp in milliseconds is added.

To stop the run and therefore also the logging and timer, the "Stop" button should be pressed. After that a run can also be continued by clicking the "Start" button again.

2 Components

Here the various components, corresponding to the tabs, are explained. The description includes a function overview, an example and two creation procedures (manual / config) as well as the currently supported devices.

2.1 Power-Supply

The power supply tab controls a given device remotely. An arbitrary maximum voltage / current can be set to avoid unwanted change and damage. In figure (3) one example is given. The psu can be easily controlled via voltage and current line edits. Also every channel can be enabled / disabled individually and also by a signal (e.g. with the test start / stop). Also a time-line of the voltage and current is given on a small graph for each channel.

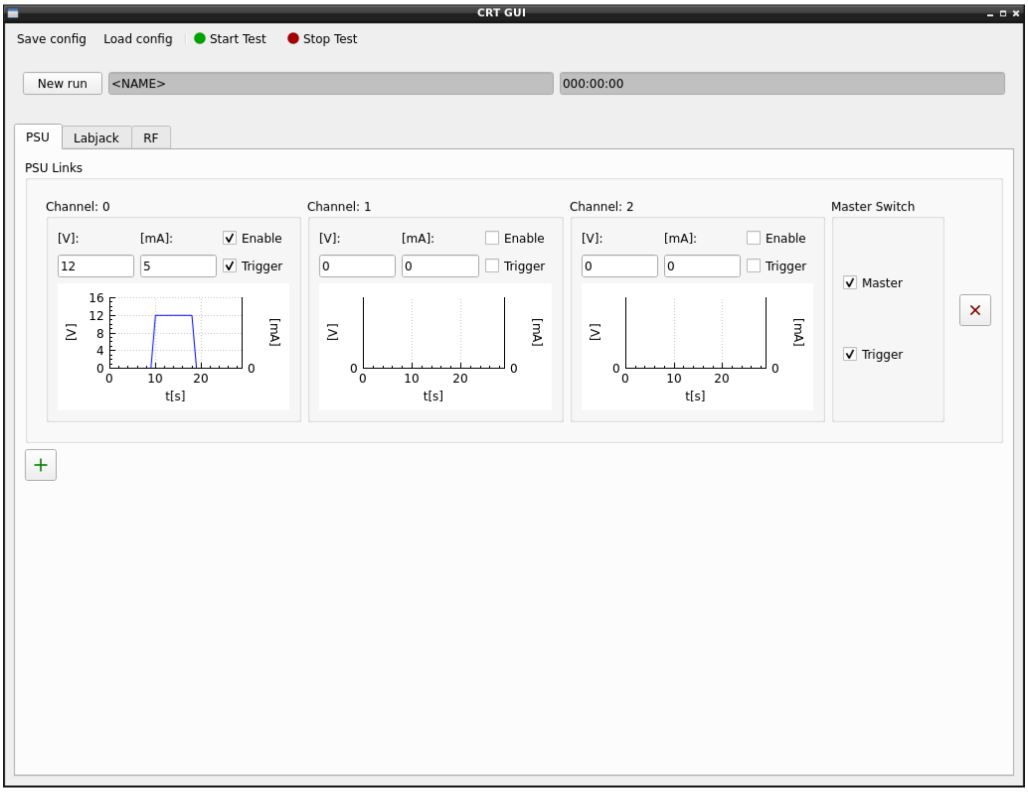

Figure 3: An example with a Rohde&Schwarz 3-channel power supply

2.1.1 Manual Creation

The window for manual creation is presented in figure (4). In the first row an individual (meaningful) name can be chosen. In the second row the vendor is put in (case insensitive). Then follows the address which has a IPv4 part and a port number after the double dot. In the last three rows a description of the power supply is given with the number of channels and the maximum voltage / current the supply either has or one wants to apply.

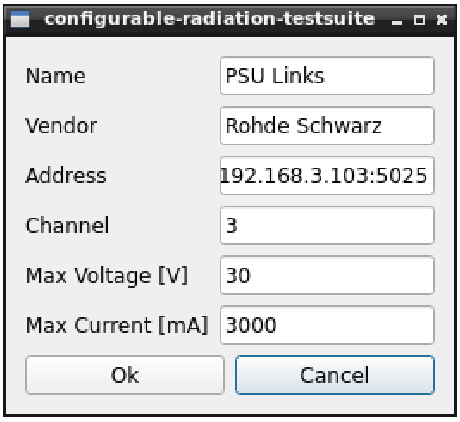

Figure 4: Creation menu for a power-supply

2.1.2 Config Creation

The power-supply is denoted in the configuration file in the "PSU" section. The first four lines correspond to the manual creation in chapter 2.1.1. After that the latest channel settings are brought up. They have a prefix *c*, a number X as identifier and a suffix *v* for voltage and *c* for current.

```
Section PSU
        name=PSU Links
        address=192.168.3.103:5025
        channel=3
        max_voltage=30
        max_current=10
        c0v=12
        c0c=500
        c1v=0
        c1c=0
        c2v=0
        c2c=0
        signal=
EndSection
```

2.1.3 Supported devices

Support for other devices can be easily added. One only needs to edit the *psu.** files and add a few lines for the correct SCPI[2] Code.

Supplier	Model
Rhode&Schwarz	HMC8043
TTI	(?)

[2]Standard Commands for Programmable Instruments

2.2 Labjack

The Labjack is a data acquisition device to measure voltages in a range of 10 mV to 10 V. As a consequence available the resolution is dependant on the chosen measurement range. The same goes for the data rate which depends on the settling time (see low-pass filter) and on the chosen resolution. In the figure (5) a Labjack with two channels (one differential, one single-ended) are shown. In the top row the mentioned data-rate, resolution and settling time can be chosen. The "Maximum" check box allows to set the data-rate to the available maximum. Below are presented the two channels with a boundary ("0" means "not set"), their latest value and the chosen gain (only post-processing). The display of the channel in the graph can be switched on or off by the corresponding check box. At the bottom the graph is presented with all channels plotted on one graph.

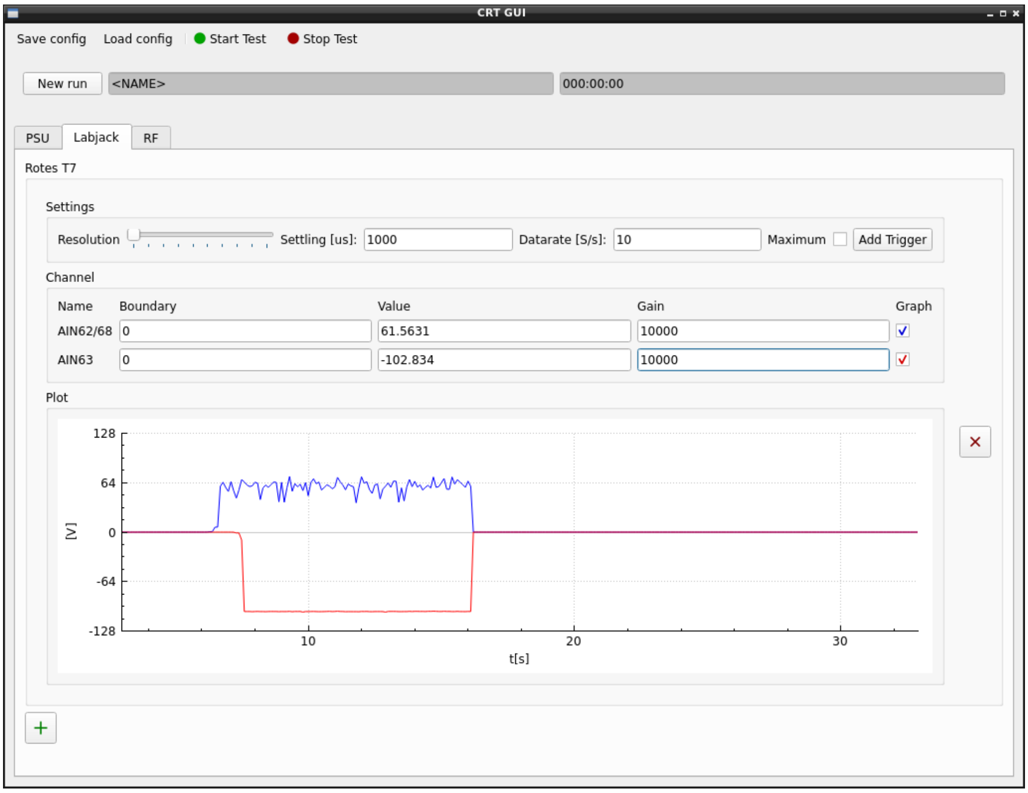

Figure 5: An example with a T7 Labjack data acquisition device

2.2.1 Manual Creation

The window for manual creation is presented in figure 6. In the first row an individual name can be chosen. The channel names in the second row can also be chosen individually. The third and fourth row determines the used channels[3] and if they are differential or not. A differential channel is given by a certain positive and negative address, whereas single ended just use 199 as negative channel.

[3] Refer to `labjack.com/support/datasheets/t-series/ain`

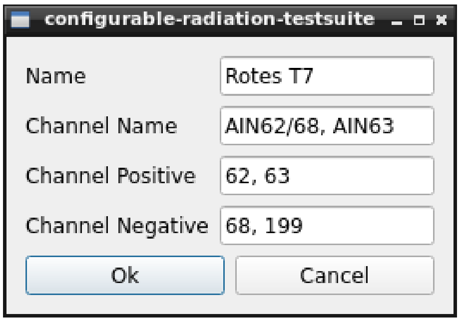

Figure 6: Creation menu for a Labjack

2.2.2 Config Creation

The LabJack is denoted in the configuration file in the “LabJack” section. First the name is defined and after that the number of channels as a validity check. The individual channels are written with a prefix *c*, a number *X* and their features as prefix:

- *n*: Name
- *pc*: Positive Channel
- *nc*: Negative Channel
- *b*: Boundary value
- *g*: Gain value

```
Section LBJ
          name=Rotes T7
          channel=2
          c0n=AIN62/68
          c0pc=62
          c0nc=68
          c0b=0
          c0g=1
          c1n=AIN63
          c1pc=63
          c1nc=199
          c1b=0
          c1g=1
          signal=
EndSection
```

2.2.3 Supported devices

To support other devices the addresses in the *Labjack.** files have to be extended.

Supplier	Model
Labjack	T7

2.3 RF Signals

In order to verify RF signals, the RF tab can be used. The current implementation supports only sine signals and requires IQ data with 16 bit. A evaluation margin in exponents of 2 can be chosen as an upper / lower boundary. Once this boundary is crossed (and the evaluation is active) an error signal gets emitted.

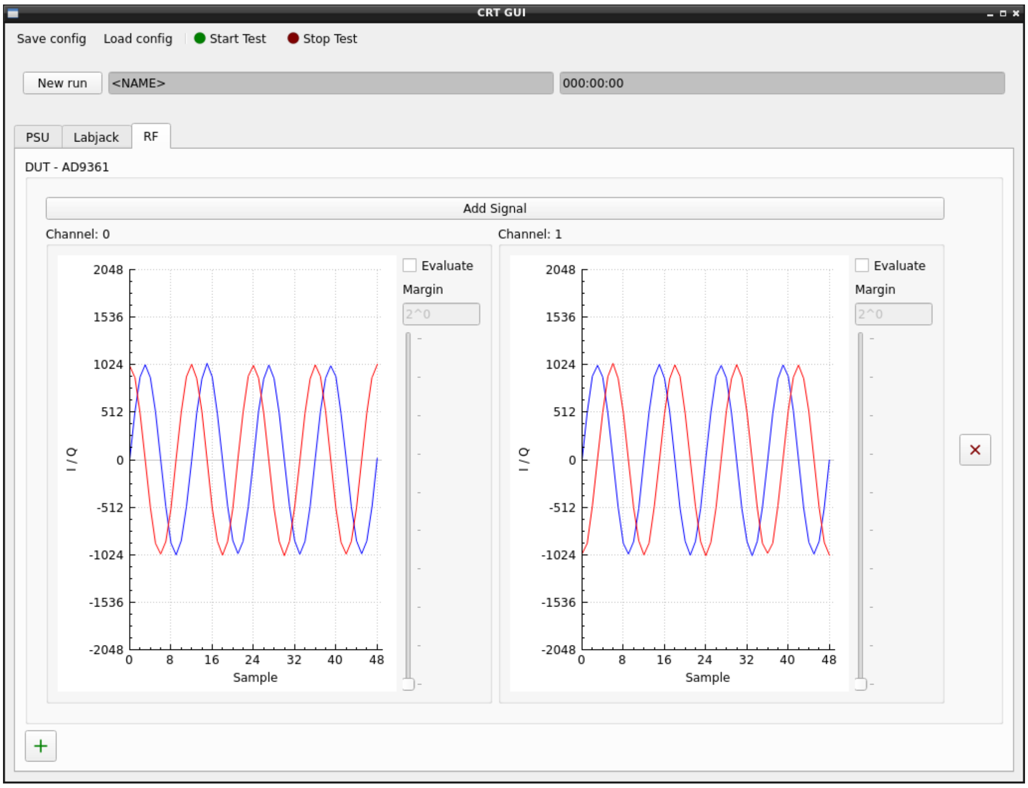

Figure 7: An active RF testrun in the suite

2.3.1 Manual Creation

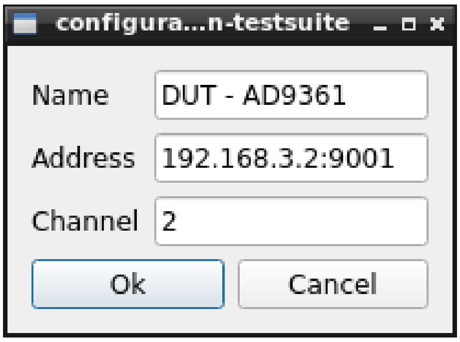

Figure 8: RF creation menu

2.3.2 Config Creation

Not implemented

2.3.3 Data Reception

The current implementation uses the simple *ncat*[4] command to receive raw data over Ethernet. This makes it rather independent. The current data protocol is given in table (1):

Channel n	8 Byte							
IQ-Data	4 Byte				4 Byte			
I	2 Byte		2 Byte					
Q					2 Byte		2 Byte	
Byte	LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB

Table 1: Data structure of the RF IQ-data

[4]Refer to `de.wikipedia.org/wiki/Netcat`

2.4 Ethernet

Not implemented yet

3 Signal Event Mechanism

The signal event mechanism allows the user to define interactions between components. For example if a certain event happens in component A it can trigger an action in component B. Signals are added via the "Add Signal" button in the components window. If no such button is available it means that this component is not able to trigger any signal.

3.1 Usage Example

If one has a power-supply and a LabJack set up, a signal interaction can be created. The LabJack has a button to add signals (see figure 9). Here the user can choose what action should be taken if a certain value boundary is crossed. In this case the power-supply would switch off if a boundary crossing would occur.

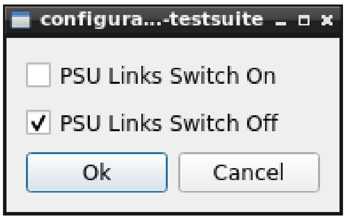

Figure 9: "Add Signal" menu with available signals

A Diagrams

B Signal Event Mechanism

To understand the signal event mechanism an overview of the participating classes is presented in figure 10.

In the *SubWindow* are general cases of signals and corresponding slots defined. The signals can be added to the *EventManger* in the derived *SubWindow* classes. Theses signals are then official and can then be used to trigger actions. Either internally in the program code or externally by adding signals in the GUI via dialog.

Figure 10: Signal Event Mechanism with the participating classes

B.1 Event Manager

The *EventManger* class object exists only once in the program. It presents an interface to all the other classes to handle signals and slots and especially the interaction between *SubWindows* and their components. For this to work, the signals from the *SubWindow* have to be added to the *EventManager*.

B.1.1 Registered Signal Structure

A signal from the *SubWindow*, which is added to the *EventManager*, holds the following properties:

```
struct RegisteredSignal {
        QString name;
        SignalType st;

        SubWindow *sub;
        void (SubWindow::*sp)(void);
```

```
};
```

The name is a descriptive name and should consist of the components name (*cfg_element_name*) and info of the action (e.g. turn on, turn off). The signal type is one of the predefined types / actions listed in the *EventManager*. The next two rows are the pointers to find the signal later on and to emit them via the call mechanism.

B.1.2 Call Mechanism

To now emit the registered signals and cause an action, the *call_tigger()* method is used which needs the type of signal to be triggered and a signal list to choose from:

```
void EventManager::call_trigger(enum SignalType st,
const QVector<struct RegisteredSignal*> &signal_list) {

    RegisteredSignal * signal;
    foreach (signal, signal_list)
        if(signal->st == st)
            emit ((signal->sub)->*(signal->sp))();
}
```

B.2 Sub Window

In the *SubWindow* a general set of signals and slots is available. These can be implemented in the derived classes for easy interaction internally as well as externally. The class holds a signal list with signals which have been added either internal or external. All the available signals are stored in the signal list of the *EventManager*.

B.2.1 Code Example

First, in the derived subclass the used signals have to be registered to inform the *EventManager* (and therefore the outer world) about it. In this example we announce from our power-supply the signals "on" and "off". Which means that our power-supply is listening and will turn itself on or off if one of these signals are emitted:

```
PSU::PSU() {
  connect(this, SIGNAL(signal_on()), psu, SLOT(switch_on()));
  connect(this, SIGNAL(signal_off()), psu, SLOT(switch_off()));

  eventManager->add_signal(psu->get_element_name() + " Switch On",
   SignalType::on, this, &SubWindow::signal_on);
  eventManager->add_signal(psu->get_element_name() + " Switch Off",
   SignalType::off, this, &SubWindow::signal_off);
  }
```

If one now wants to not only react, but also to trigger other components, the predefined slots have to be used. They can be connected to certain internal class signals.
For example if we track various sub-voltages on a board via LabJack, we can then connect an off-signal if we cross a set boundary:

```
connect(channel, SIGNAL(boundary_check_failed()),
 this, SLOT(trigger_signal_list()));
```

If a voltage now crosses a boundary, all signals that have been added to the signal list are now emitted. The action could be also limited by calling only off-signals with the other slots.