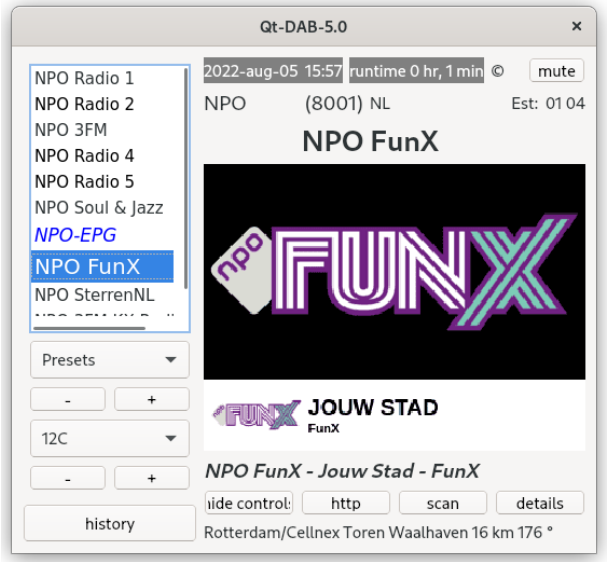
Qt-DAB 5*

User's guide for version 5.0 (revised)

Jan van Katwijk, Lazy Chair Computing
The Netherlands

August 8, 2022



Contents

1 Introduction

Qt-DAB is an advanced program for decoding terrestrial DAB (DAB+) transmissions. Qt-DAB, a program with a GUI, is designed to run on both Linux (x64) computers, on RPI 2 and up running Linux, and is cross compiled for Windows.

For *Linux (x64)* a so-called *AppImage* is available, a kind of container, an executable file that contains - next to the executable program - the libraries needed to run.

For *Windows*, an *installer* is available that will install the executable together with the required libraries.

These precompiled versions can be found in the releases section of the repository for Qt-DAB (https://github.com/JvanKatwijk/qt-dab/releases).

For creating an executable on an RPI 2 or higher or any other Linux system, section 5 of this report contains a pretty detailed description with scripts for Debian and Ubuntu.

Qt-DAB is implemented in C++, with extensive use of the Qt framework for its graphical appearance. It also uses a number of existing open source libraries, such as fftw, libsndfile, libsamplerate, libusb, and Qt-DAB is itself open source, available under the Gnu GPL V2.

The sourcetree for Qt-DAB contains - obviously - sources to generate an executable for Qt-DAB-5 and it contains sources for generating a Qt-DAB-4.x version.

The structure of this guide is simple, in section 2 the GUI and GUI widgets for the Qt-DAB program are discussed, and in section 3 the *configuration and control* widget is discussed. Some of the widgets that can be made visible under user control are discussed in section 4. in section 5 the supported devices and their control widgets for the Qt-DAB program are briefly discussed.

In section 6, a description is given on how to configure and build an executable for Qt-DAB.

2 The GUI

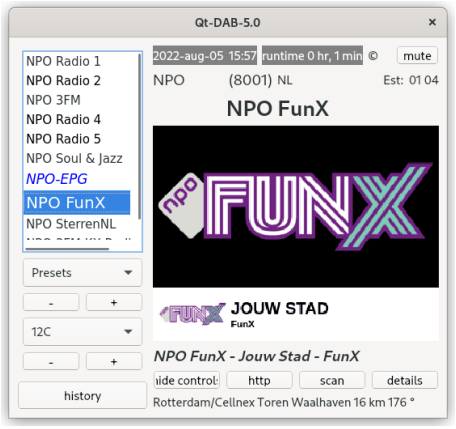


Figure 1: Qt-DAB-5

The Qt-DAB-5 GUI is simplified compared to the 4.4 version, most controls are moved to the *configuration and control* widget. What the interface does *not* have is a device selector. The software allows for selecting (and reselecting) a device, and will use the last selected device when starting the program. If the device cannot be opened, the *configuration and control* widget will be shown, that widget has a selector for one of the configured devices.

The main widget contains at the left side the display and selectors for channel and service selection. New is the button at the bottom, labeled *history*. Touching it will show the *history* list, i.e. the list of

all (channel, service pairs) that were seen since the last time the list was cleaned. Of course, selecting a service in that list is possible. Cleaning the list is by focussing the mouse on the list and touching the *right hand* mouse button.

The main part in the right hand side is - as can be seen on the picture - for the service label. Above this label there is the usual indication of ensemble name, service name (if a service is selected), country indication, and TII numbers. At the top there is the time, both current time and run time, a copyright symbol that, when it gets the focus, shows a.o which sources were used for constructing the executable. Toop right there is a *mute* button.

At the bottom of the right hand side of the widget there are 4 push buttons, believed to be important for quick interactions. The buttons are

- the button to show (or hide) the configuration and control widget;
- the button to start or stop the http server and start the webbrowser;
- the button to start or stop scanning the band;
- the button to show or hide the technical details of the currently selected service.

3 The configuration and control widget

As said, in Qt-DAB-5 a choice was made to make the main widget as simple as possible and move the controls to the configuration widet, hence the name *configuration and control*.

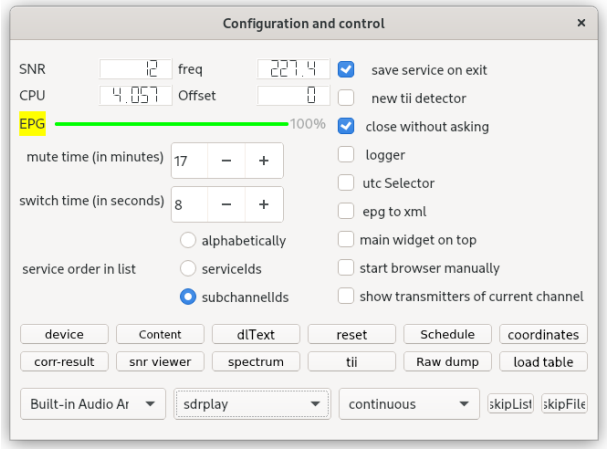


Figure 2: Configuration and control

Figure 2 shows the widget. At the left side, some general info is given, i.e. the selected frequency (in MHz), the correction on the frequency (in Hz), the SNR (in dB) and the CPU load (in percents). The latter is the overall load of the system.

Below these 4 numbers this is a progressbar telling the successrate of decoding the FIC. Since the FIC (i.e. the part containing the descriptions of the actual content) is (relatively) easy to decode, it is almost certain that if the bar does not show 100services is unlikely to succeed. To the left of this bar, a small label will tell if an EPG service is found.

The mute button is on the main widget, the time - in minutes - for muting when the button is touched, can be selected. Below the mute time selector, there another "time" selector can be set, the value of this selector tells the system how many seconds it might take after selecting a channel until some DAB data is seen.

The list of services is - as can be seen - displayed in a list on the main widget. There are three possible ordering for these services to be shown, the element below the selector for the switch time gives the opportunity to select one of three possibilities.

The checkboxes At the right hand side of the widget there is a column of checkboxes for a variety of settings. From the top to the bottom one sees

- *save service on exit*, if selected the software records the service that is selected, so that whenever the program is started again, the software "knows" what channel to choose and which service to select in that channel;
- *new tii detector*, if selected another algorihm is used to identify the TII data from the (spectrum of the) NULL period.
- *close without asking*, on normal program termination, a small widget will show, asking to confirm closing the program. If this option is selected, that part in the termination process of the program is skipped;
- *logger*, if selected, a log file will be opened in which some logging data, such as which service is selected and when, will be written;
- *utc selector*, if selected, UTC time is used rather than local time;
- *epg to xml*, an imported module - next to the built-in "processor" for EPG - is able to translate the EPG data into readable XML. However, on some input the imported module will crash the system, so it is not standard activated;
- *main widget on top*, when selected will ensure that the main widget is always on top. This comes in handy when there are a lot of widgets on the screen, or if larger widgets, such as a map, are shown. Note however that under Windows, new widgets are always put in the centre, so, as an example the main widget is still on top and you try to close the program, the confirmation widget will appear *behind* the main widget and widgets will not move, so you are stuck;
- *start browser manually*, is selected will prevent the system, when the *http* button on the main widget is touched and a http server is started, to start the "default" browser on the system, allowing you to 'start a preferrred browser;
- *show transmitters of current channel*, while by default, when selected a map to display the transmitters on a map, all transmitters in all channels selected are shown, selecting this option will clear the list on the map when changing the channel.

Control buttons The configuration and control widget shows two rows with push buttons, controlling a variety of configuration options.

- *device*, the button controls the visibility of the widget for the device control;
- *content*, when touched, a widget shows with a description of the content of the currently selected ensemble;
- *dlText*, when touched the texts of the dynamic label will be stored in a file;
- *reset*, when touched will perform the operations: stop and start;
- *schedule*, when touched a schedule can be specified;
- *coordinates*, when touched you will be asked to fill in the coordinates of your position;

- *corr-result*, when touched a widget shows with the result of the correlation;
- *snr viewer*, when touched a widget will appear showing the development of the SNR over time;
- *spectrum*, when touched a widget will appear showing the spectrum and the signal constellation;
- *tii*, when touched, a widget will appear showing the spectrum of the NULL part of the incoming DAB frames;
- *Raw dump*, when touched, the incoming samples are written to a PCM file (2 channels, int16 format, rate 2048000);
- *load table*, when touched *and the required functionality is installed* a fresh copy of the tii library is loaded and installed,

Selectors at the bottom Finally, at the bottom of the widget there are 5 selectors, 3 comboboxes and 2 push buttons. From left to right, they are

- a selector for the audio output. By default *default* is chosen, the setting will obviously be stored for use at the next program invocation;
- a device selector. A list of configured devices is shown. Note however, that - under Linux - your system has to have installed the support libraries for the selected device. For Windows, for all devices in the list the installed contains support libraries;
- a selector for the scan mode. Scanning is in one of three modes
  - *continuous* mode, indicating that the scanning will continue until explicitly stopped by touching the scan button (again) (or stopping the program);
  - *singla scan* mode, indicating that a single scan will be performed over all channels in the band, apart from thise declared hidden in a skiplist, starting at the first channel, stopping at the last channel. This mode generates the most detailed descriptions of the data in the channels seen;
  - *scan to data* mode, indicating that a scan - starting at the currently selected channel - will be performed stopping as soon as a channel is encountered with DAB data.
- a button labeled *skipList* will - when touched - show the default skipList. The entries in the list can be set to "+" or "-", with obvious meanings;
- button labeled *skipFile* will - when touched - show a file selection menu with which a skipList can be selected. If a non-existing file is selected, that file will be created as skipList.

4 Some widgets

4.1 Technical details

The main widget has as - said - a button *details* which which a widget can be made visible that shows some data on the currently selected service, see figure 3. The widget contains, next to the technical data for the service and the progressbars telling the quality of the data for the service, two pushbuttons. As the names suggest, the button *frame dump* - when touched - instructs the software to dump the AAC frames of the audio service into a file. Such a file can be played by e.g. VLC. The second button, *dump audio* - when touched - instructs the software to dump the audio output into a ".wav" file with a samplerate of 48000 samples/second.

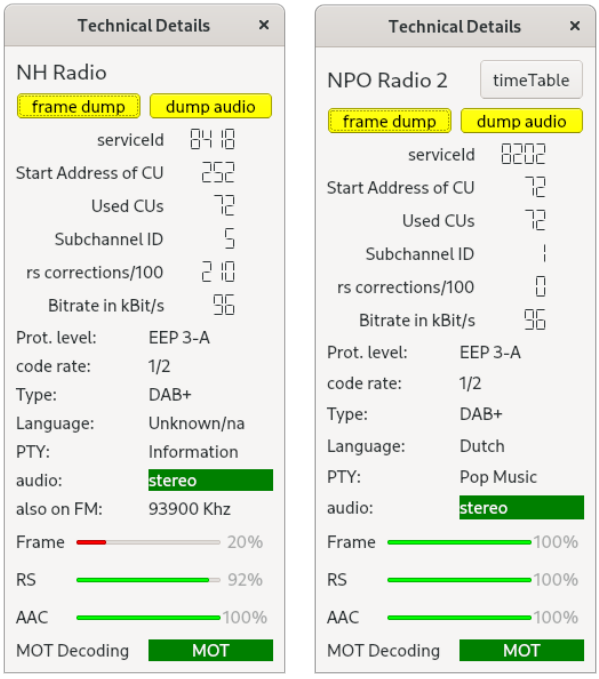


Figure 3: Technical details

4.2 Widgets for inspecting the signal

The configuration and control widget contains 4 buttons, each controlling the visibility of a widget for showing some aspects of the signal.

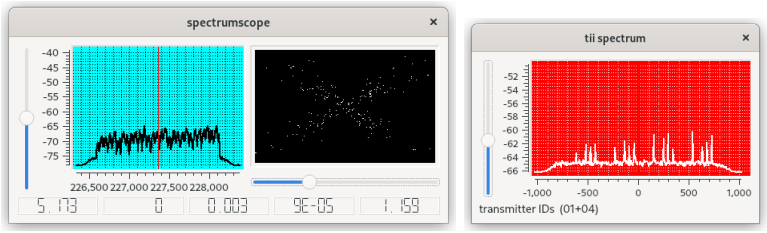


Figure 4: Spectrum, signal constellation and TII data

The spectrum widget and TII data Figure 4 shows in the widget left, the spectrum of the incoming signal and the signal constellation. At the bottom, the widget contains a number of "quality indicators", tooltips tell their meaning.

The widget right in figure 4 shows the spectrum s of NULL period of the incoming DAB frames, the NULL period contains encoded data identifying the individual transmitters. As can be seen in the picture, a group of 4 "peaks" is repeated 4 times. The encoding is apparently (01 04), which happens to be a transmitter near my location.

Correlation result Figure 5 shows the result of the correlation to compute the first sample of the first data block. Since one usually receives data from more than a single transmitter, the picture often

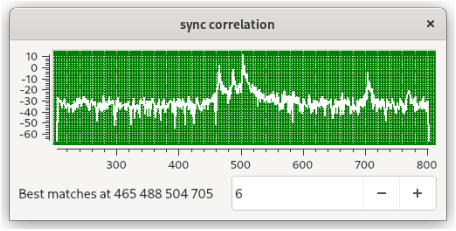


Figure 5: Correlation result

shows more than a single peak. At the bottom of the widget the sample numbers for the best matches are shown,

The peak closest to sample 504 is the strongest one, the data from the transmitter responsible for that peak is the data used in the decoding process. Apparently the transmitters responsible for the peaks at 465 and 488 are a little bit closer to the receiver than the transmitter for sample 504 (as a matter of fact, since the samplerate is 2048000 it is easy to compute how much closer they are).

The ength of the segment shown, in units of 100 samples, can be set in the spinbox, the picture shows a segment of 600 sampples.

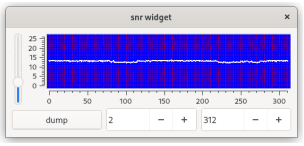


Figure 6: SNR over time

SNR Figure 6 shows the development of the SBR (Signal Noise Ratio) over time. While the image - showing the development over a few minutes in a stable environment - might not be overwhelming ingteresting, it comes in handy with setting up an antenna and it might be interesting to see the development over 12 to 24 hours. For that purpose the SNR can be stored into a file and a separate small program is available to view the dump/

4.3 Scheduling

Touching the *schedule* button on the configuration and control widget initiates input handling for setting a schedule element. *Scheduling* of certain events is possible for a period of up to 7 days. Figure **??** shows 3 widgets. The first one appears when touching the schedule nutton, it shows the services and commands from which one may be chosen. The list of services is composed from the services in the current ensemble, and a few commands. Commands are

- *nothing*, with the intuitive semantics;
- *exit*, with the intuitive semantics;
- *framedump* indicating that the AAC frames of the audio service active at the time of the execution of the command are to be written into a file. The softwware will generate a filename for this file. Any - in time - second occurrence of the command will stop the activity, selecting a different service or channel will also stop the activity.
- *audiodump* indicating the the resulting PCM samples of interpreting the audio of the service active at the time of executing the command, are written into a ".wav" file. The software will generate

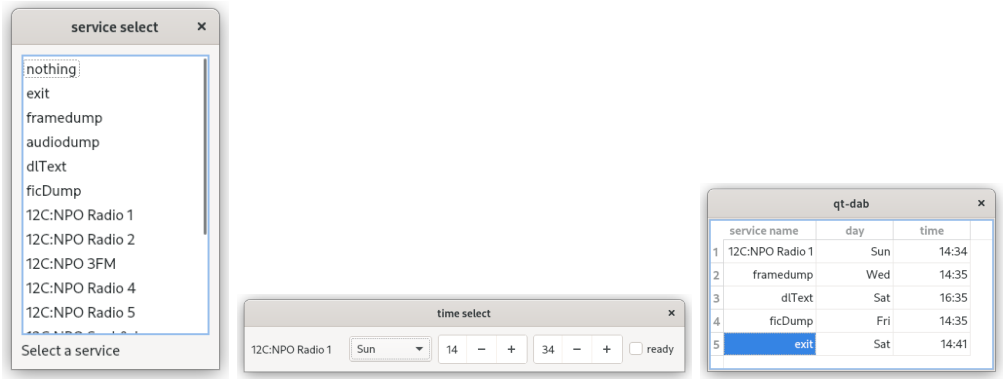


Figure 7: Scheduler widgets

an appropriate filename; Any - in time - second occurrence of the command will stop the activity, selecting a different service or channel will stop the activity;

- *dlText* indicating that the dynamic label text of any service running at the time of the execution of the command or later as long as the command is active, is stored in a file, the name of which is generated by the software. Selecting another service will *NOT* stop the activity.
- *ficDump* indicating that at the specified time the bits being the input to the FIC decoding process will be stored in a file. A second occurrence of the command will stop the activity.
- *a service name* indicates that at the given time the service will be selected and started.

The second widget in the picture shows the widget on which the day and time can be specified for the event to happen. The third widget is an example of the resulting list of commands.

Note that on restarting the Qt-DAB program after it was stopped the schedule list will be searched for obsolete commands, these will be removed.

4.4 Showing the content

Touching the *content* button causes the program to show a widget with a description of the content of the current ensemble Figure 8 shows a description of the content of the current ensemble. The top

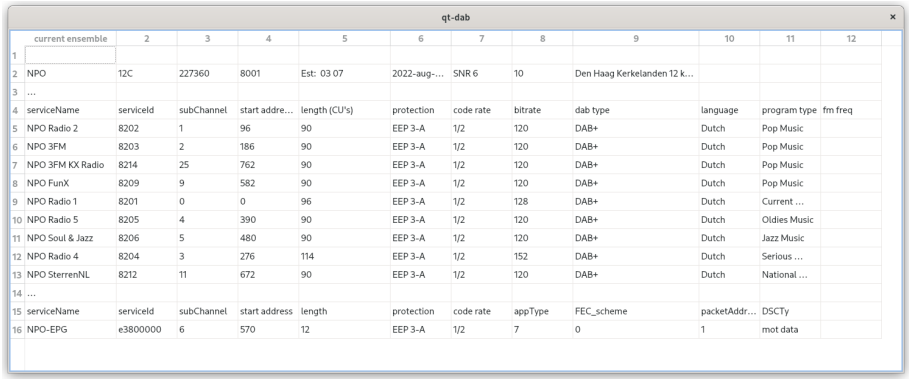


Figure 8: Content of current ensemble

line gives some general information, including the frequency (in KHz, here 227360), an estimate of the

identification of the transmitter whose data is receiver, as well as the name of that transmitter, the SNR, the ID and the channel name.

Saving the description into a file is done by *double clicking* on the widget, the output is a ".csv" file.

4.5 Scanning

Scanning is started (stopped) using the *scan* button on the main widget. Scanning is - as mentioned before - in one of the modes *continuous*, *single scan*, *scan to data*. While scanning in *single scan* mode, an extensive description of each ensemble found is given, similar to the description generated by the application of the *content* button, see figure 9.

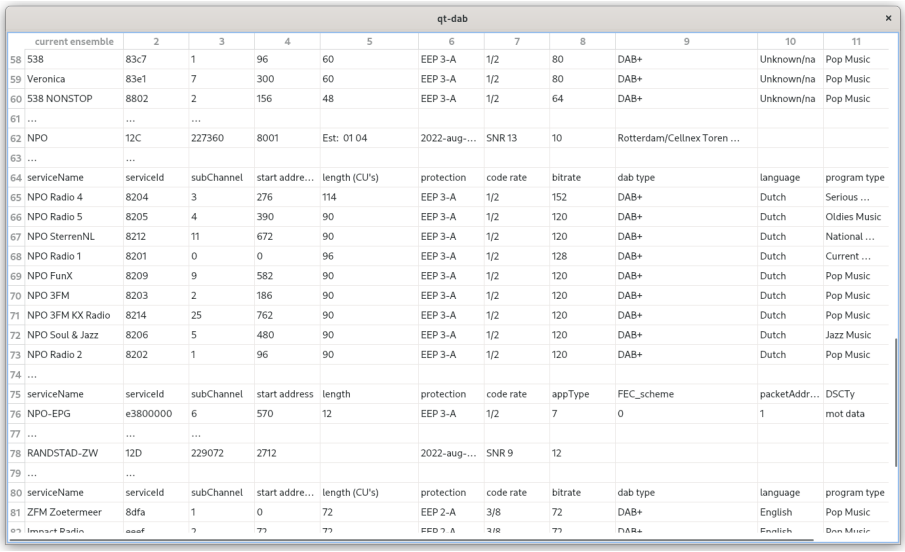


Figure 9: Output of scan

The output is written as a ".csv" file, and can be viewed by e.g. Libre Office.

4.6 Showing the map

Qt-DAB-5 has - as Qt-DAB-4.x does - an option to generate info for a *map*, showing the positions of transmitters received.

The main widget has a button labelled *http*, when touched, it will (try to) start a simple http server. Based on user requests, a default option was added, i.e. to start the *local* browser on the machine the program is running on. Touching the button again will stop the http server, but it has no influence on the browsser.

The configuration and control widget contains a selector *start browser manually*, that - when turned on - will cause the software *not* to start the default browser, allowing the user to select a browser.

A picture of a map is shown in figure 10, the map shows that transmitters were received from 3 locations, and the column at the right hand side of the picture shows which channels were received with what receivers.

5 Supported input devices

Qt-DAB supports a variety of input devices, the Adalm Pluto, the AIRspy, the hackrf, the limeSDR, RT2832 based sticks and SDRplay RSP devices. Furthermore, there is support for the rtl_tcp server,

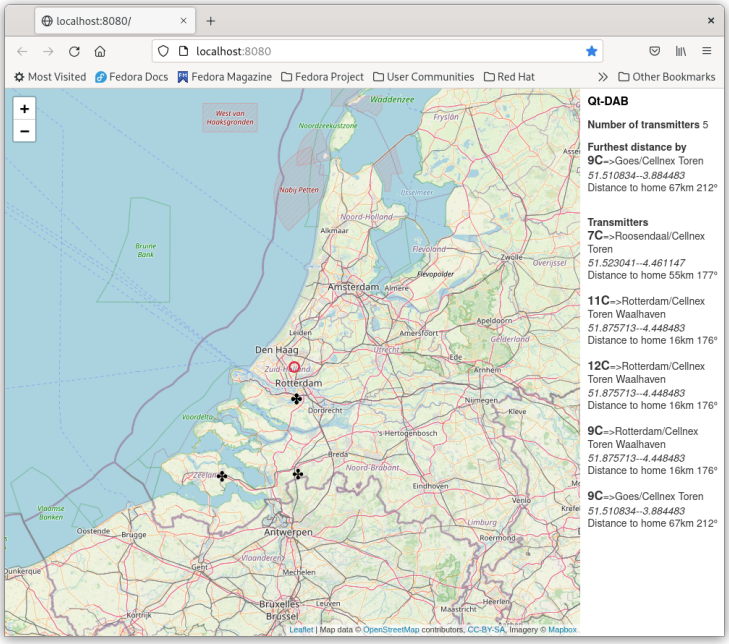


Figure 10: Map

for file input (raw, wav and xml), and for devices for which a *Soapy* interface library exists,

Both the *appImage* and the *Windows installer* are configured with (almost) the whole range of devices: SDRplay RSP (different versions for the 2.13 and 3.0X library versions), the Adalm Pluto, the AIRspy, the hackrf, the LimeSDR, and - of course - the RT2832 based dabsticks.

5.1 The SDRplay RSP

The Qt-DAB software supports all RSP's from SDRplay. Qt-DAB provides two different device handlers for the RSP's, one for devices using the 2.13 SDRplay interface library, the other one supports devices using the 3.0X SDRplay interface library. Note that API 3.10 on Windows prohibits the use of the 2.13 library on Windows.

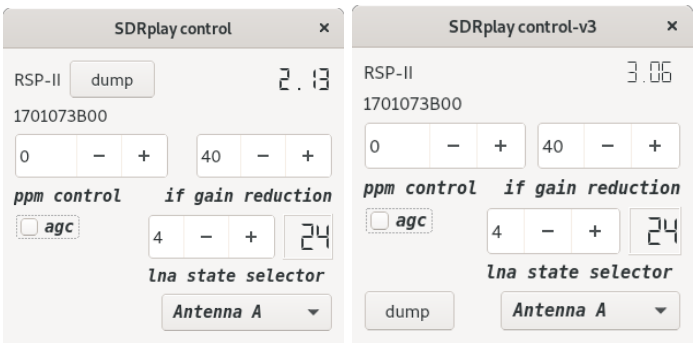


Figure 11: Qt-DAB: The two control widgets for the SDRplay

As figure 11 shows, the control widgets for the two different versions resemble each other, their

implementations differ considerably though. Both have spinboxes for setting the *if gain reduction*, the *lna state* and a *ppm offset*.

An optimal value for the *ppm offset* is to be determined experimentally, the RSP II, as used here, is happy with a ppm offset 0, the oscillator offset is almost zero in the region of Band III.

The spinbox for the *if gain reduction* is programmed to support the range of values between 20 and 59. The range of values for the *lna state* depends on the model of the RSP. The software will detect the model and fill in the range accordingly.

If the *agc* is selected, the *if gain reduction* spinbox will be hidden, its value is then irrelevant.

The RSP II has two (actually 3) slots for connecting an antenna. If an RSP II is detected, a combobox will be made visible for *antenna selection*.

A similar combobox exists for selecting a tuner in the widget for the 2.13 library controller. The SDRplay duo has two tuners. If the software detects the duo, a combobox will be made visible for selecting a tuner (note that this feature is not tested, I do not have a duo).

Finally, both versions of the control widget contain a *dump* button. If touched, the raw input from the connected device will be stored in a so-called xml formatted file. First a menu is shown for selecting a filename, a suggestion for the name of the file *device name* - *date* is given. Touching the button again will stop dumping and the file will be closed.

If more than one connected device is detected, a widget appears on which a selection can be made which device to use.

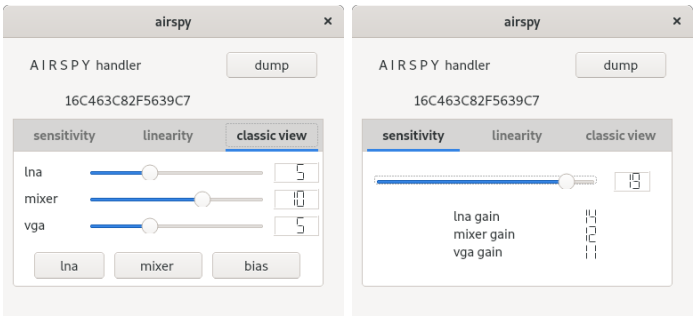


Figure 12: Qt-DAB: Widgets for AIRspy control

5.2 The AIRSpy

The control widget for the AIRspy (figure 12, left) contains three sliders and a push button. The sliders are to control the lna gain, the mixer gain and the vga gain.

To ease balancing the setting of the sliders, two combined settings are included in the widget, selectable by the tab *sensitivity* and *linearity*. Figure 12 right side, shows the setting at selecting the tab *sensitivity*.

Touching the button labeled *dump* instructs the software to dump the raw stream of samples into a file in the xml format (Note that while processing DAB requires the samplerate to be 2048000, that rate is not supported by the AIRspy, implying that the driver software has to do some rate conversion. The xml file though will just contain the samples on the rate before conversion).

If more than one connected airspy is detected a widget will appear with which the device to use can be selected.

5.3 The hackrf

The control widget for hackrf (figure 13) shows, next to the Serial Number of the device, a few sliders, a few checkboxes, a spinbox and a push button.

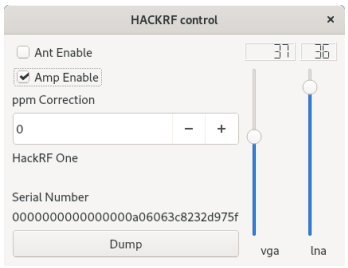


Figure 13: Qt-DAB: Widget for hackrf control

- the *sliders* are there for controlling the lna and vga gain, the slider values are limited to the range of possible values;
- The *Ant Enable* checkbox is for Antenna port Power control (not used in this controller);
- The *Amp Enable* checkbox is - if enabled - for additional gain on the antenna input;
- the *ppm correction* spinbox can be set to correct the oscillator (on 227 MHz, the Qt-DAB software reports an offset of somewhat over 3 KHz);
- the *Dump* push button when pushed, starts dumping the raw input in xml file format. Touching the button again will halt the dumping and close the file.

5.4 The LimeSDR

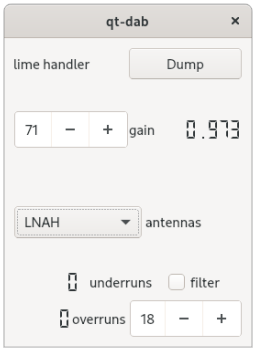


Figure 14: Qt-DAB: Widget for Lime control

On selecting the LimeSDR (if configured), a control widget for the LimeSDR is shown (figure 14). The widget contains five controls:

- *gain* control, with predefined values;
- *antennas*, where *Auto* is usually the best choice;
- *Dump*, if touched, the raw input from the connected device will be written to a file in the so-called xml format.

New is the inclusion of a filter. Note that the limeSDR reads samples with a bandwidth of 204KHz, while the frequency distance between successive channels is 1712KHz. So, DX-ing in adjacent channels, where e.g. the first channel contains a strong DAB signal and the next one a weak, is difficult.

Therefore the control widget for the limeSDR has two additional controls,

- switching a software FIR filter on-off (the checkbox labeled *filter*),
- setting the size of the FIR filter (the spinbox below the checkbox).

Note that using the filter is not free, for a filter with a size of N, N * 2048000 complex additions and multiplications are performed. While on a modern PC that is not an issue, it certainly is on ARM based micros like the RPI 2.

5.5 The RTLSDR stick

On selecting the dabstick (i.e. RT2832 based devices) (if configured), a control widget for the device appears (figure 15).

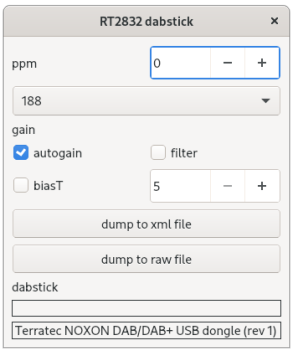


Figure 15: Qt-DAB: Widget for rtlsdr device

The widget contains just a few controls:

- a *spinbox* for setting the ppm. Note that on average the offset of the oscillator with DABsticks is (much) larger than that with devices like the SDRplay. The DAB software is able to correct frequencies to up to app 35 KHz, for some sticks the frequency error was large and correction using the ppm setting was required.
- a *combobox* for setting the gain. The support software for RT2832 based devices generates a list of allowable gain settings, these settings are stored in the combobox;
- a *combobox* for setting the autogain on or off;
- a *push button* that, when touched, will instruct the software to dump the raw input in the aforementioned xml format. At first a menu appears for selecting a file. Touching the button again will stop dumping and close the file.

New is the inclusion of a *filter*. Note that the DABstick reads samples with a bandwidth of 2048 KHz for a signal with a bandwidth of app 1.536 MHz, while the frequency distance between successive channels is 1712KHz. So, DX-ing in adjacent channels, where e.g. the first channel contains a strong DAB signal and the next one a weak, is difficult.

The controller therefore contains an optional FIR filter, for which the rtlsdr control widget has two additional controls:

- switching a software filter on-off (the checkbox labeled *filter*),
- setting the size of the FIR filter (the spinbox below the checkbox).

Note that switching the filter on is not free, for a filter with a size of N, N * 2048000 complex additions and multiplications per second are performed. While on a modern PC that is not an issue, it certainly is on ARM based micros like the RPI 2.

If more than one connected RTLSDR based device is detected, a widget appears on which the device of choice can be selected.

5.6 The Pluto device

When selecting *pluto*, a widget (figure 16) appears with a spinbox for selecting the gain, and a checkbox for selecting the agc. If *agc* is enabled, the spinbox for the gain setting is invisible. The widget contains

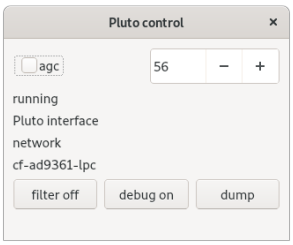


Figure 16: Qt-DAB: Widget for Adalm Pluto device

furthermore three buttons:

- the *debug control* button, when activated, instructs the software to show output on each step in the initialization process (note that the setting of the debug button will be maintained between invocations);
- the *dump* button will cause the original input - with a samplerate of 2100000 samples per second[1] - to be stored in an xml file.
- the *filter* button. The adalm pluto has as option specifying a fir-filter, to be executed within the Pluto device. This implementation of the controller for pluto will load a predefined filter onto the Pluto device which is enabled by default. With the filter button the filter can be disabled or enabled. Note that the button text indicates the action when touching, not the current state.

5.7 Support for Soapy

Soapy is a generic device interface, a kind of wrapper to provide a common interface to a whole class of devices. Qt-DAB supports Soapy, and its use is tested with the Soapy interface for the SDRplay.

The widget for soapy control (see figure 18) when applied to the Soapy interface for the SDRplay contains the obvious controls, similar to that of the regular control for the SDRplay.

5.8 rtl_tcp

rtl_tcp is a server for rtlsdr devices, delivering 8 bit IQ samples.

In the small widget, the ip address of the server can be given. Since the default port for the server is 1234, that port number is the one used by the client. After clicking the *connect* button, the client will look for a server and pass some parameters.

However, the port number can be set in the ".ini" file, by setting

[1] The smallest samplerate that pluto gives is slightly larger than the required 2048000, 2100000 is chosen since it is easy to handle

Figure 17: Qt-DAB: Widget for soapy

`rtl_tcp_port=XXX`

where XXX is to be replaced by the portnumber of choice.

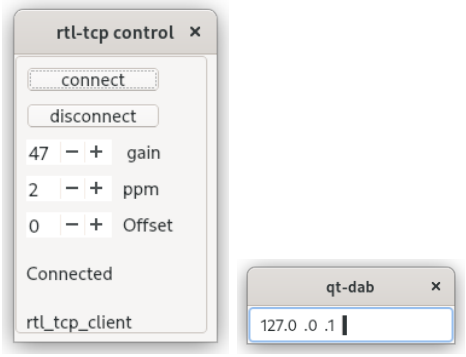


Figure 18: Qt-DAB: Widget for rtl_tcp

5.9 File input

Qt-DAB supports both *writing* raw input files and *reading* them back. Writing a file as PCM file is initiated bythe *Raw dump* button on the main GUI, writing a file as xml file by the *dump* button on the various device widgets. Qt-DAB differentiates between reading

- raw 8 bit files as generated by e.g. Osmocom software (usually files with an extension ".raw" or ".iq");
- PCM (i.e. ".wav") files, provided the data is 2 channels and with a samplerate of 2048000, generated by Qt-DAB and with an extension ".sdr";
- xml files. The xml file format was defined by Clemens Schmidt (author of QIRX) and me and aims at saving files in the original format, so to allow easy exchange between different DAB decoder implementations. In order to support proper decoding of the contents, the data in the file is preceded by a detailed description in xml, hence the name xml file format.

When selecting file input ".raw" or ".wav", a simple widget is shown (figure 19), with as indication the number of seconds the file is being played.

Since processing an xml file implies some interpretation, the widget (figure 20) for control when reading an xml file is slightly more complex. It contains - next to the progress in reading the data - a

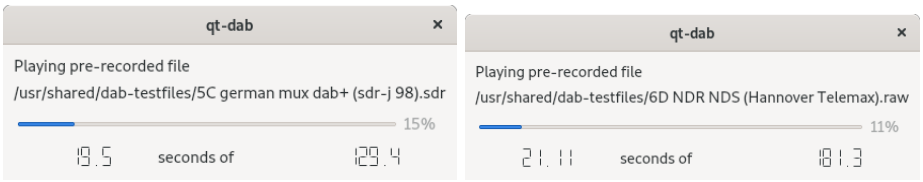


Figure 19: Qt-DAB: Widgets for file input

description of the contents of the file. So, the program that generated the file as well as the device used in that program are displayed, the number of bits of the samples, as well as the number of elements is displayed as is the samplerate of recording and the frequency of the recording.

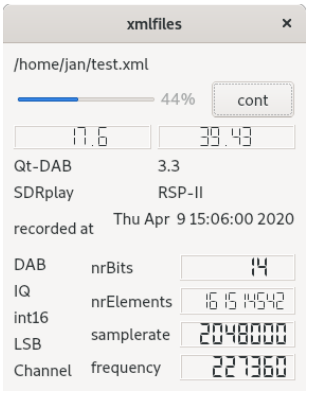


Figure 20: Qt-DAB: Widget for xml file input

Touching the *cont* button will instruct the software to restart reading at the beginning of the segment in the file after reaching the end.

5.10 The Qt-DAB device interface: adding support for a device

The Qt-DAB software provides a simple, well-defined interface to ease interfacing a different device. The interface is defined as a class, where the actual device handler inherit from.

```
class deviceHandler {
public:
         deviceHandler  ();
virtual ~deviceHandler ();
virtual bool restartReader (int32_t);
virtual void stopReader ();
virtual int32_t getVFOFrequency ();
virtual int32_t getSamples (std::complex<float> *, int32_t);
virtual int32_t Samples ();
virtual void resetBuffer ();
virtual int16_t bitDepth ();
virtual void show ();
virtual void hide ();
virtual bool isHidden ();
virtual QString deviceName ();
private:
int32_t lastFrequency;
};
```

A device handler for a - yet unknown - device should implement this interface. The semantics of most functions are obvious, a few are more specific

- if opening the device fails, the constructor should give up by throwing an exception;
- *bitDepth* tells the number of bits of the samples. The value is used to scale the Y axis in the various scopes and to scale the input values when dumping the input.
- *deviceName* returns a name for the device. This function is used in the definition of a proposed filename for *dump*s.
- The GUI contains a button to hide (or show) the control widget for the device. The implementation of the control for the device will implement - provided the control has a widget - functions to *show* and to *hide* the widget, and *isHidden*, to tell the status (visible or not).

What is needed for another device Having an implementation for controlling the new device, the Qt-DAB software has to know about the device handler. This requires adapting the configuration file (here we take qt-dab.pro) and the file radio.cpp, the main controller of the GUI.

Modification to the qt-dab.pro file Driver software for a new device, here called *newDevice*, should implement a class, e.g. *newDevice*, derived from the class *deviceHandler*.

It is assumed that the header is in a file *new-device.h*, the implementation in a file *new-device.cpp*, both stored in a directory *new-device*.

A name of the new device e.g. *newDevice* is to be added to the list of devices, i.e.

```
   CONFIG += AIRSPY
...
   CONFIG += newDevice
```

Next, somewhere in the qt-dab.pro file a section describing XXX should be added, with as label the same name as used in the added line with CONFIG.

```
newDevice {
   DEFINES         += HAVE_NEWDEVICE
   INCLUDEPATH     += ./qt-devices/new-device
   HEADERS         += ./qt-devices/new-device/new-device.h \
                      .. add further includes to development files, if any
   SOURCES         += ./qt-devices/new-device/new-device.cpp \
                      .. add further implementation files, if any
   FORMS           += ./qt-devices/new-device/newdevice-widget.ui
   LIBS            += .. add here libraries to be included
}
```

Modifications to radio.cpp The file "radio.cpp" needs to be adapted in three places

- In the list of includes add

  ```
  #ifdef HAVE_NEWDEVICE
  #include new-device.h
  #endif
  ```

- The names of selectable devices are stored in a combobox. So, in the neighborhood of

  ```
  #ifdef HAVE_AIRSPY
  deviceSelector -> addItem ("airspy");
  #endif
  #ifdef HAVE_SDRPLAY
  deviceSelector -> addItem ("sdrplay-v3");
  #endif
  ```

  the text

```
#ifdef HAVE_NEWDEVICE
deviceSelector -> addItem ("newDevice");
#endif
```

is added.

- If selected, the class implementing the device handler should be instantiated, so, in the direct environment of

```
#ifdef HAVE_AIRSPY
if (s == "airspy") {
   try {
      inputDevice = new airspyHandler ....
....
#endif
```

the code for allocating a device handler is added

```
#ifdef  HAVE_NEWDEVICE__
        if (s == "newDevice") {
           try {
              inputDevice       = new newDevice (..parameters..);
              showButtons ();
           }
           catch (int e) {
              QMessageBox::warning (this, tr ("Warning"),
                                    tr ("newDevice not found\n"));
              return nullptr;
           }
        }
        else
#endif
```

Linking or loading of device libraries The approach taken in the implementation of the different device handlers is to *load* the required functions for the device library on instantiation of the class. This allows execution of Qt-DAB even on systems where some device libraries are not installed.

6 Configuring and building an executable

6.1 Introduction

While for both Windows and Linux-x64 there are ready-made executables for installing resp. executing the Qt-DAB program, there are situations where one wants (or needs) to create its own version. For e.g. use of the software on an RPI one has to create an executable, for e.g. using the software with other or non-standard configured devices one has to create an executable. This section will describe the configuration options and the building process.

6.2 What is there to configure?

The Qt-DAB software can be built using either qmake or cmake generating a *Makefile*. The current *configuration file* for qmake, *qt-dab.pro*, has more options for configuring than the configuration file for use with cmake, *CMakeLists.txt*.

QMake and CMake take a different approach, while the configuration options for use with qmake requires some editing in the *qt-dab.pro* file, selecting configuration options with cmake is ususally through command line parameters.

Note that the *qt-dab.pro* file contains a section *unix* and a section *win32* for Windows that contain settings specific to the OS used. The CMakeLists.txt file is only used for Linux-x64.

Both the AppImage and the Windows installer are built using qmake as makefile generator.

6.2.1 Console or not (qt-dab.pro only)

```
# CONFIG += console
  CONFIG -= console
```

While for tracing and debugging purposes it might be handy to see all the (text) output generated during execution, for normal use it is not. Including or excluding *console* in the configuration determines whether or not a console is present when executing.

6.2.2 Configurable common devices

Configuring devices is simple, for devices as mentioned above as well as for *rtl_tcp* the *qt-dab.pro* file and the *CMakeLists.txt* contain a description. File input (all versions, i.e. raw files, sdr files and xml files) is by default configured in Qt-DAB executables, changing this is possible, but implies changes to the sources.

Using the qt-dab.pro file For configuring devices in the *qt-dab.pro* file, comment out or uncomment the line with the devicename.

```
CONFIG += airspy
CONFIG += dabstick
CONFIG += sdrplay-v2
CONFIG += sdrplay-v3
CONFIG += lime
CONFIG += hackrf
CONFIG += pluto
CONFIG += soapy
CONFIG += rtl_tcp
```

Note that for *soapy*, and for *limeSDR* there is no support in generating a windows executable, due to the absence of a suitable dll. Furthermore, for Windows select "pluto-2" rather than pluto.

Using the CMakeLists.txt file The CMakeLists.txt file contains support for AIRspy, SDRplay, SDRplay_V3, RTLSDR, Hackrf, pluto and LimeSDR. Including a device in the configuration is by adding "-DXXX=ON" to the command line, where XXX stands for the device name.

6.2.3 Configuring the TII database

Using qmake/make Configuration for (not) using the database is by selecting one of

```
#CONFIG += preCompiled
CONFIG += tiiLib
```

Configuring for *tiiLib* will select program items such that a *tiiLib*, *if available* of course, will be loaded and interpreted. *If no such library is available, Qt-DAB will function, however, loading a fresh copy of the database is not possible.*

Installing the tiiLib The tiiLibrary is - for Linux-x64 and for RPI - available in the source tree. The library should be placed in e.g. /usr/local/lib.

Using cmake In the CMakeLists.txt file, by default the "tiiLib" configuration is chosen.

6.2.4 Configuring SSE

In the deconvolution of data, use is made of code generated by the *spiral code generator*. If the code is to run on an x86-64 based PC, some speed up can be obtained by using the code generated for use with SSE instructions. If the code is to run on an RPI, it is - depending on the configuration - sometimes possible to speed up the process by using ARM specific instructions. Of course, the compiler used in the building process has to support generating the right instructions, as fas as known, the Mingw compiler, used for generating the windows executable, does not.

The qt-dab.pro file contains in the unix section

```
CONFIG += PC
#CONFIG += RPI
#CONFIG += NO_SSE
```

Selecting "CONFIG += PC" selects SSE instructions, and deselects threading of backends - after all, a standard PC has more than sufficient power to run the decoding in a single thread.

Selecting "CONFIG += RPI" selects options suitable for having the software run on an RPI. However, the precise content depends on the processor architecture and the compiler chain.

Selecting "CONFIG += NO_SSE" is for e.g. Mingw cross compiler for Windows.

When using *cmake*, pass "-DVITERBI_SSE=ON" as command line parameter for PC's.

6.2.5 Configuring audio

- When running the Qt-DAB program remotely, e.g. on an RPI near a decent antenna, one might want to have the audio output sent through an IP port (a simple listener is available).
- Maybe one wants to use the audio handler from Qt.
- The default setting is to use *portaudio* to send the PCM samples to a selected channel of the soundcard.

The *Linux* configuration for the Qt-DAB program offers in the qt-dab.pro file the possibility of configuring the audio output:

```
#if you want to listen remote, uncomment
#CONFIG         += tcp-streamer         # use for remote listening
#otherwise, if you want to use the default qt way of sound out
#CONFIG         += qt-audio
#comment both out if you just want to use the "normal" way
```

If cmake is used, pass "-DTCP_STREAMER=ON" as parameter for configuring the software for remote listening, use "-DQT_AUDIO=ON" for qt audio, or *do not specify anything* for using portaudio in the configuration.

Note that the configuration for Windows is only for "portaudio".

6.2.6 Configuring TPEG in the tdc

Handling TPEG in the tdc is only partially supported. Interpretation of the data is not part of the Qt-DAB software, however, the software can be configured to extract the TPEG frames and send these to an IP port.

In the qt-dab.pro file, we have

```
#very experimental, simple server for connecting to a tdc handler
CONFIG         += datastreamer
```

In cmake the parameter "-DDATA_STREAMER=ON" can be passed to include handling TPEG as described in Qt-DAB.

6.2.7 Configuring IP datastream (qt-dab.pro only)

IP data can be extracted from the DAB stream and send out through an IP port.

```
#to handle output of embedded an IP data stream, uncomment
CONFIG          += send_datagram
```

Note that - if not specified in the ini file - defaults are used for ip address and port.

6.2.8 Selecting an AAC decoder

By default the *faad* library is used to decode AAC and generate the resulting PCM samples.

The source tree contains - in the directory *specials*, the sources for the libfaad-2.8 version. It is quite simple to create and install an appropriate library if the Linux version supports a faad liubrary that is somehow incompatible.

An *alternative* is to use the *fdk-aac* library to decode AAC (contrary to the libfaad the fdk-aac library is able to handle newer versions of the AAC format, these newer versions are not used in DAB (DAB+)).

Selecting the library for the configuration is by commenting out or uncommenting the appropriate line in the file *qt-dab.pro* (of course, precisely one of the two should be uncommented).

```
CONFIG    += faad
#CONFIG   += fdk-aac
```

(see the subsection for installing the libraries).

6.2.9 Configuring for platforms

Processing DAB (DAB+) requires quite some processing power. On small computers like an RPI2, performing all processing on a single CPU core overloads the core.

In order to allow smooth processing on multi core CPU's, an option is implemented to partition the workload. In order to partition processing, uncomment

```
DEFINES += __THREADED_BACKEND
DEFINES += __MSC_THREAD__
```

in the *qt-dab.pro* file.

In case cmake is used, edit the file CMakeLists.txt and comment out or uncomment the line

```
#add_definitions (-D__THREADED_BACKEND)  # uncomment for use for an RPI
#add_definitions (-D__MSC_THREAD__)  # uncomment for use for an RPI
```

It is recommended to use

```
CONFIG += PC
```

in the qt-dab.pro file, when targeting towards a standard x64 based PC running Linux, using this will set the SSE and the threading.

It is recommended to use

```
CONFIG += RPI
```

in the qt-dab.pro file when targeting for an RPI, the threading will be set and the NO_SSE option is set.

6.3 Preparing the build: installing libraries

6.3.1 Installing the libraries

Prior to compiling, some libraries have to be available. For Debian based systems (e.g. Ubuntu for PC and Buster for the RPI) one can load all required libraries with the script given below.

```
sudo apt-get update
sudo apt-get install git cmake
sudo apt-get install qt5-qmake build-essential g++
sudo apt-get install pkg-config
sudo apt-get install libsndfile1-dev qt5-default
sudo apt-get install libfftw3-dev portaudio19-dev
sudo apt-get install zlib1g-dev rtl-sdr
sudo apt-get install libusb-1.0-0-dev mesa-common-dev
sudo apt-get install libgl1-mesa-dev libqt5opengl5-dev
sudo apt-get install libsamplerate0-dev libqwt-qt5-dev
sudo apt-get install qtbase5-dev
```

Note that on newer versions of Ubuntu, this list might change, especially w.r.t. Qt5 libraries. It seems that "qt5-default" is not available as separate library anymore, it just can be removed then.

If *libfaad* is the selected aac decoder, install

```
sudo apt-get install libfaad-dev
```

If *fdk-aac* is the selected aac decoder, install

```
sudo apt-get install libfdk-aac-dev
```

6.3.2 Downloading of the sourcetree

Since the script also loads *git*, the sourcetree for Qt-DAB (including the sources for dab-mini) can be downloaded from the repository by

```
git clone https://github.com/JvanKatwijk/qt-dab.git
```

The command will create a directory *qt-dab* with subdirectories for dab-mini, dab-maxi and the unsupported dab-2.

6.3.3 Installing support for the Adalm Pluto

The Pluto device uses the *iio* protocol. Support for *Pluto* is by including

```
sudo apt-get install libiio-dev
```

and - to allow access for orinary users over the USB - ensure that the user name is member of the pugdev group, and create a file "53-adi-plutosdr-usb.rules" is in the "/etc/udev/rules" directory.

```
#allow "plugdev" group read/write access to ADI PlutoSDR devices
# DFU Device
SUBSYSTEM=="usb", ATTRS{idVendor}=="0456", ATTRS{idProduct}=="b674",
MODE="0664", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="2fa2", ATTRS{idProduct}=="5a32",
MODE="0664", GROUP="plugdev"
# SDR Device
SUBSYSTEM=="usb", ATTRS{idVendor}=="0456", ATTRS{idProduct}=="b673",
MODE="0664", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="2fa2", ATTRS{idProduct}=="5a02",
MODE="0664", GROUP="plugdev"
# tell the ModemManager (part of the NetworkManager suite) that
# the device is not a modem,
# and don't send AT commands to it
SUBSYSTEM=="usb", ATTRS{idVendor}=="0456", ATTRS{idProduct}=="b673",
ENV{ID_MM_DEVICE_IGNORE}="1"
SUBSYSTEM=="usb", ATTRS{idVendor}=="2fa2", ATTRS{idProduct}=="5a02",
ENV{ID_MM_DEVICE_IGNORE}="1"
```

6.3.4 Installing support for the RTLSDR stick

It is advised - when using an RT2832 based "dab" stick - to create the library for supporting the device

```
git clone git://git.osmocom.org/rtl-sdr.git
cd rtl-sdr/
mkdir build
cd build
cmake ../ -DINSTALL_UDEV_RULES=ON -DDETACH_KERNEL_DRIVER=ON
make
sudo make install
sudo ldconfig
cd ..
rm -rf build
cd ..
```

6.3.5 Installing support for the AIRspy

If one wants to use an AIRspy, a library can be created and installed by

```
wget https://github.com/airspy/host/archive/master.zip
unzip master.zip
cd airspyone_host-master
mkdir build
cd build
cmake ../ -DINSTALL_UDEV_RULES=ON
make
sudo make install
sudo ldconfig
cd ..
rm -rf build
cd ..
```

6.3.6 Installing support for SDRplay RSP

If one wants to use an RSP from SDRplay, one has to load and install the library from "www.SDRplay.com".

6.3.7 Making the installed libraries visible

The installation of these device handlers will install libraries in the

`/usr/local/lib`

directory. Note that the path to this directory is NOT standard included in the search paths for the Linux loader. To add this path to the searchpaths for the Linux loader, create a file

`/etc/ld.so.conf.d/local.conf`

with as content

`/usr/local/lib`

The change will be effective after executing a "sudo ldconfig" command.

The installation of these device handlers will furthermore install some files in the

`/etc/udev/rules.d`

directory. These files will ensure that a non-root user has access to the connected device(s).

Note that in order for the change to be effective, the *udev* subsystem has to be restarted. The easiest way is just to reboot the system.

6.4 Finally: building an executable

6.4.1 Using cmake to build the executable

After installing the required libraries, and after editing the configuration (if required), compiling the sources and generating an executable is simple.

Using cmake, creating an executable with as devices the SDRplay, the AIRspy, and the RTLSDR based dabsticks, the following script can be used:

```
cd qt-dab/dab-maxi
mkdir build
cd build
cmake .. -DSDRPLAY=ON -DPLUTO=ON -DAIRSPY=ON -DRTLSDR=ON ... -DRTL_TCP=ON
make
```

The CMakeLists.txt file contains instructions to install the executable in "/usr/local/bin".

```
sudo make install
cd ..
cd ..
```

6.4.2 Using qmake to build the executable

Assuming the file qt-dab.pro is edited, the same result can be obtained by

```
cd qt-dab/dab-maxi
qmake
make
```

In some Linux distributions replace qmake by qmake-qt5!

The *qt-dab.pro* file contains in both the section for unix as for windows a line telling where to put the executable

```
DESTDIR         = ./linux-bin
```

By default in Linux the executable is placed in the *./linux-bin* director in the *qt-dab* directory.

6.5 Configuring for pluto-rxtx

Qt-DAB can be configured to use the Adalm Pluto as both receiving device and as transmitting device, but in Linux only/ Configuring is easy,

```
CONFIG         += pluto-rxtx
#CONFIG         += pluto
```

is all that is needed, i.e. unselect regular pluto and select pluto-rxtx.

When configured, and when the device *pluto-rxtx* is selected, the audio of the selected service, augmented with the text of the dynamic label encoded as RDS signal.

Transmission frequency is set default to 110 MHz, but can be set as command line parameter using the "-F XXX" flag, where XXX stands for the frequency **expressed in KHz**.

7 Acknowledgements

Qt-DAB and derived programs are written and maintained by me. The software is provided *as is*, and made available under the Gnu GPL V2 license.

Many people contributed (and contribute) by providing feedback, suggestions and code fragments, in particular:

- Andreas Mikula, for continuous feedback, testing and suggestions;

- Herman Wijnants, for his continuous enthousiastic feedback on and suggestions for the Windows version of Qt-DAB;
- Stefan Pöschel, for providing code for and giving suggestions to handling the AAC code;
- Stuart Langland, for its comments, suggestions and code contributions;
- probonopd, for its contribution with creating appImages;
- Przemyslaw Wegrzyn, for contributing code for handling charsets;
- Paul Howard-Beard, for his enthousiastic experiments with new features, comments and his suggestion to add features like a mute button, the per channel gain settings, and alarm; and
- Michael Lass, for showing me the use of the Gcc address sanitizer, pointing out some (actually too many) address violations discovered by the sanitizer and giving suggestions and advice for the repair.

Furthermore I am grateful

- to SDRplay ltd (Andy Carpenter), for providing me the possibility to use the Ia and II versions of the SDRplay RSP devices, all wonderful devices;
- to Benjamin Vernoux, for making an AIRSPY device available;
- to Great Scott Gadgets, for making an HACKRF device available;
- to Jan Willem Michels, for making a LimeSDR device available;
- to Olaf Czogalla, for donating an RT2832 based stick after having lively discussions on TPEG; and
- to Robin Getz (Analog Devices), for making an Adalm Pluto available, a device with lots of possibilities, still to discover.

Qt-DAB is developed as hobby program in spare time. Being retired I do have (some) spare time and programming Qt-DAB (and my other programs) is just hobby. Contributions are always welcome, especially contributions in the form of feedback and additions and corrections to the code, but obviously also in the form of equipment that can be used.

If you consider a financial contribution, my suggestion is to support the red cross or your local radio amateur club instead.