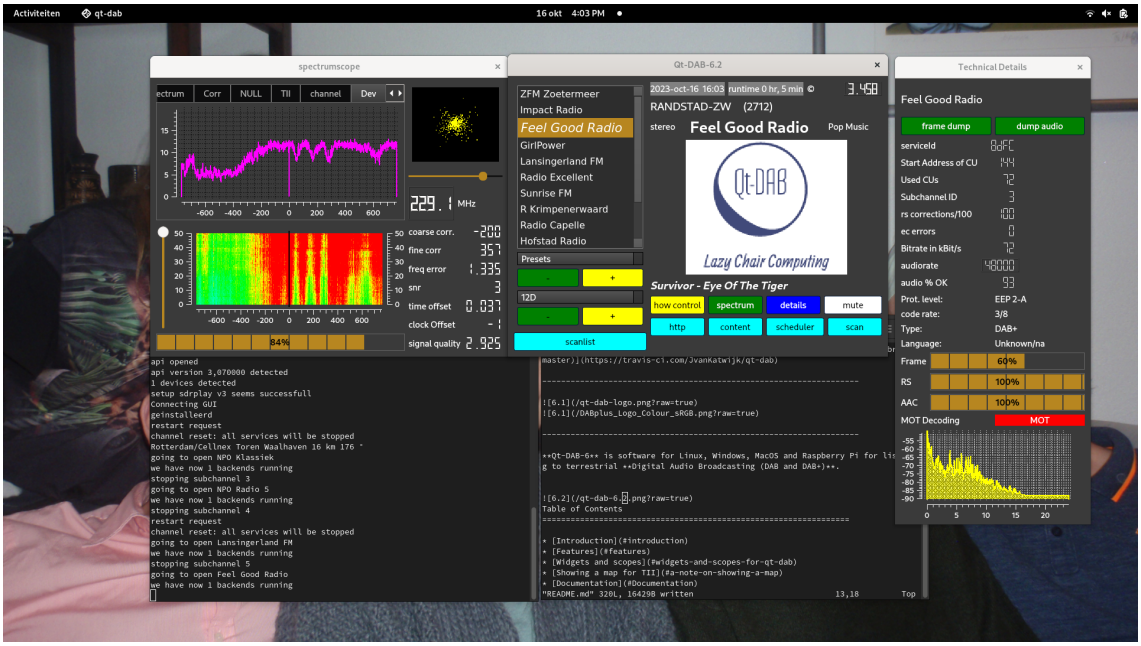
Qt-DAB 6*

User's guide for version 6 (updated for 6.2)

Jan van Katwijk, Lazy Chair Computing
The Netherlands

October 31, 2023



*© both the software and this document is with J.vanKatwijk, Lazy Chair Computing. While the software is available under a GNU GPL V2, the manual is not. No parts of this document may be reproduced without explicit written permission of the author. I can be reached at J.vanKatwijk at gmail dot com

Contents

1 Qt-DAB-6

1.1 Introduction

Qt-DAB is an advanced program for decoding terrestrial DAB and DAB+ transmissions. Qt-DAB, a program with a GUI, is designed to run on both Linux (x64) computers, on RPI 3 and up running some Linux variant, and is *cross compiled* for Windows (32 bits).

Qt-DAB decodes a DAB stream, input is by an *SDR* device or from a file. Qt-DAB is (or can be) configured with a number of popular SDR devices, such as SDRplay RSP's, AIRspy, Lime, Hackrf, Adalm pluto, and obviously, the cheap RTLSDR devices (including the recent V4 version), the dabsticks.

Qt-DAB is a GUI driven program, while different elements of the GUI show all kinds of information, all *interaction* to control the program is using a mouse. In normal use there is no need for passing on command line parameters.

The focus in the development always is - next to actual decoding the signal of course - visualisation of the signal in its raw and its processed form. This is reflected in the structure of the GUI.

The GUI of Qt-DAB consists of 4 parts, the picture on the fromt page shows three of them: the *main* widget, a *spectrum scope widget*, and a *technical details widget*. A fourth widget contain checkboxes and buttons with which details of the decoding process are set. The main widget is always visible, th visibility of the widgets is strictly under user control.

The structure of this guide is simple, section 2 deals with installing the program, in section 3 it is discussed what happens when the program is started for the first time, in section 4 the GUI and GUI widgets are described, In section 5 some functions, i.e. content, scanning, and showing the map, is discussed. in section 6 *Scheduling* of service selections is discussed, and in section 7 the supported devces for the Qt-DAB program are briefly discussed.

1.2 Hardware Requirements

Qt-DAB runs pretty well on modern PC's. One does need a reasonable amount of computing power, though. On an RPI 3 and up the program will run well, obviously on an Arduino it will not. The load on Windows is app 5 to 10 times higher than the load on Linux x64, on a 5 year old laptop running W10 (With a 2.5G I5 processor) the load is app 30 percent, on my newer x64 laptop running Linux, also running an I5 (10-th generation), takes between 2 and 3 percent.

1.3 Differences between Version 6 and previous versions

The main differences between V6 and previous versions are in the GUI. Note that Versions 6, 5 and 4 are built, using over 90 percent of the same sources, essentially only the GUI controlling parts differ. There are two - more or less - larger differences between V6 and V5. In V6 there are some new scopes, in order to avoid overloading the screen, the scopes are grouped together in a *tabbed* widget. selection of the signal display is by choosing a tab.

The second difference is that some buttons are moved back to the main widget, the rationale being that buttons that directly influence the visibility of the main components should always be visible.

2 A note on installing Qt-DAB-6

A windows installer for Qt-DAB-6 is available. The installer will install Qt-DAB by creating a folder in the system folder "Program Files (x86)" on the C drive and will create - if checked - a link to the desk.

While that folder contains ".dll's" for the configured devices, one might need to add support for these devices. As an example, the well-known RTLSDR stick needs software to be installed by the Zadig orogram, for Pluto one needs to install some more support libraries.

For Linux on the x64 PC, a so-called *AppImage* is available. Such an AppImage is itself a small closed filesystem with all programs and libraries for running Qt-DAB aboard. However, the AppImage does NOT contain support libraries or the various configured devices. On selecting a device, Qt-DAB will look for a device support library, and will load the relevant functions if found. If not found, Qt-DAB will report that and wait for another device selection.

Of course since the sources for Qt-DAB are available, one might compile an executable. Since building an executable from the sources is far from trivial, a separate document exists with a detailed description on what to do.

The precompiled versions (i.e. Windows installer and AppImage) can be found in the Releases section of the github repository ("https://github.com/JvanKatwijk/qt-dab").

3 Starting the program for the first time

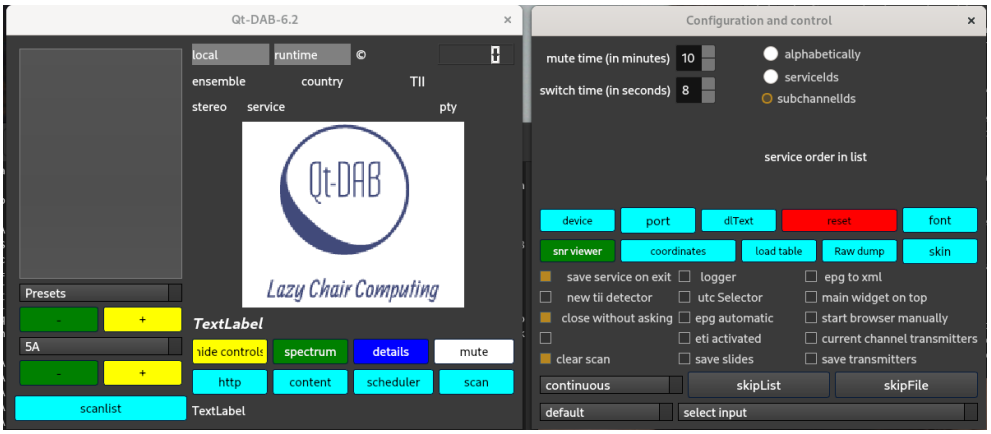


Figure 1: starting Qt-DAB

When Qt-DAB starts for the very first time (or any other time when the input device is not known or cannot be opened), next to the main widget, the *configuration and control widget* will show, that widget contains the *device selector*.

Touching this device selector, i.e. the combobox at the bottom of the configuration and control widget labeled *select input*, will show the list of configured devices. *Note again that showing up of a device name in the list means that the device is configured, it does not mean that the device support library on the system is installed.* In Qt-DAB, the required functions of the device support library are loaded whenever a device is selected.

On selecting a service, a widget shows where settings for the device, e.g. gain settings, can be done. If device selection fails, e.g. due to a non-connected device, a warning will show (as an example, see figure 2.

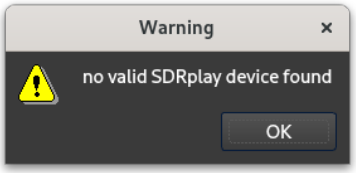


Figure 2: No device found

Selecting a channel with the channel selector, i.e. the combobox labeled "5A" on the main widget, will instruct the software to start decoding data from the selected channel.

Values for a large amount of settings will be stored in the so-called ".ini" file, a Latin1 encoded file (readable and writeable) with the name ".qt-dab.ini" which is stored in the user's home directory. If the file does not exist, it will be created. Settings like device selection, gain setting for the device, channel selection, service selection etc etc are typically the values that are maintaines between program invocations. A fragment

```
[General]
autoBrowser=0
channel=12C
clearScanResult=1
closeDirect=1
config-4-h=671
config-4-w=322
configVisible=1
configWidget-h=386
configWidget-w=553
configWidget-x=45
configWidget-y=581
correlationVisible=0
device=sdrplay-v2
deviceVisible=0
....
```

If device selection succeeds, the device starts reading input and decoding of the input starts. If a channel is selected that contains (recognizable) DAB data, a list of services from the payload will show up in the main widget.

4 The GUI: the widgets

4.1 The main widget

The main widget shows the relevant data for the user's selection of a channel and a service. It displays slide(s) carried in the selected service, and the dynamic label text. Furthermore, it provides buttons for controlling the visibility of the three before mentioned widgets, and for operations like scanning, scheduling etc.

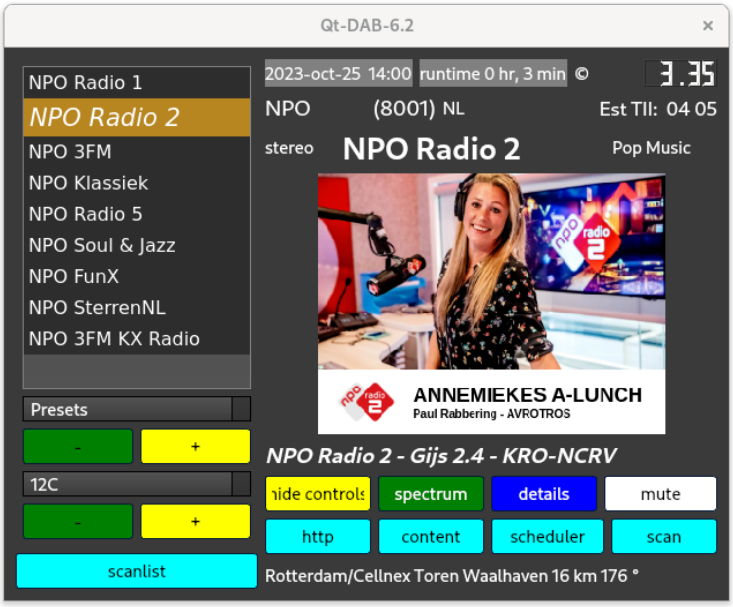


Figure 3: main widget

Selecting a *service* is - as can be expected - by clicking on the name of the service in the list, selecting the *next* or *previous* service (in the list) is by clicking on the "+" or "-" buttons below the combobox labeled *presets*.

Qt-DAB supports *presets*, i.e. a list of *channel, service* combinations that are put on a shortlist. The presets are maintained between program invocations.

Adding a service to the preset list is by clicking with the *right hand mouse button* on the service that is the selected one on the service list. Removing the service from the preset list is by clicking with the right hand mouse button on the service name in the list of presets.

Note that clicking with the right hand mouse button on another service in the service list, will start processing the service as *background service*. You will be asked to provide a filename for the AAC output.

Selecting a channel is with the aforementioned channel selector, labeled *12C* in this example. Skipping to the previous or next channel in the list is with the "+" or "-" buttons below the channel selector.

New is the *scanList* button. Touching it will show the *scanlist*, i.e. the list of all *channel, service* pairs that were seen during the last scan (if no scan was performed, the list is - obviously - empty). Of course, selecting a service in that list is possible (although in a mobile environment there is no garantee that channel contains any or this particular service data). Running a scan will will not clear the list, unless the configuration widget had the *clear scan* selector set (which is default). Note that adding an element from the scan list to the presets is possible.

The top half of the right hand side of the widget shows some info on the selected service. At the top right, a *mute* button allows muting the audio for a specified time (duration os specified in the configuration and control widget).

Audio services usually carry one or more slides that are displayed. In case no slide is (yet)

found, a default slide is displayed.

The bottom line of the widget shows - *if configured* - the name of the transmitter, together with an estimate of the distance and the azimuth.

The controls on the widget are (from left to right, top to bottom):

1. controlling the visibility of the configuration and control widget;
2. controlling the visibility of the spectrum widget;
3. controlling scanning the band;
4. controlling the visibility of the technical details widget, that shows information on the selected (audio) service;
5. controlling the weg server for showing a map with the positions of the transmitters the signals of which can be decoder;
6. showing a specification of the services in the currently received ensemble;
7. adding data to the scheduler.

4.2 The spectrum widget

As mentioned, in this version the different scopes (displays) are put into a single widget. The "tab" defines which scope is shown. The widget further contains an IQ scope, a waterfall display and a list of quality indicators. Depending on the configuration, there are 5 or 6 scopes, the fifth scope tells about the channel, the 6-th scope about the frequency offsets of the carriers in decoding.

While the "channel" scope is part of the precompiled versions, its inclusion in the configuration is an option.

The waterfall display is directly coupled to the selected scope, it will show the progress in time of the data of the selected scope. Figure 4 shows the spectrum scope. The spectrum

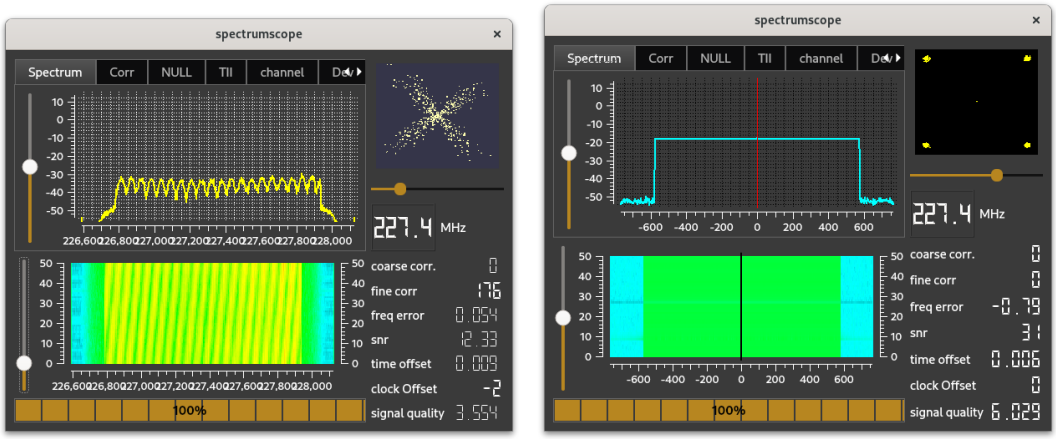


Figure 4: Spectrum scope and ideal signal

shows the signal, which has a width of just over 1.5 MHz, the number on the widget, 227.4, shows the frequency (in MHz) of the signal.

The "cross" in the box top right, is the so-called *constellation* of the *decoded signal*. The decoded signal elements are internally represented as complex numbers, and in the IQDisplay depicted in a two dimensional space. Ideally one sees 4 dots, one in the middle of each quadrant. In the examples here, the IQ display shows that while the phases differ app 90 degrees, the amplitudes of the different signal elements vary. (By clicking with the right hand mouse button on the IQ dsplay, a switch is made to the constellation of the *input* rather than decoded, signal).

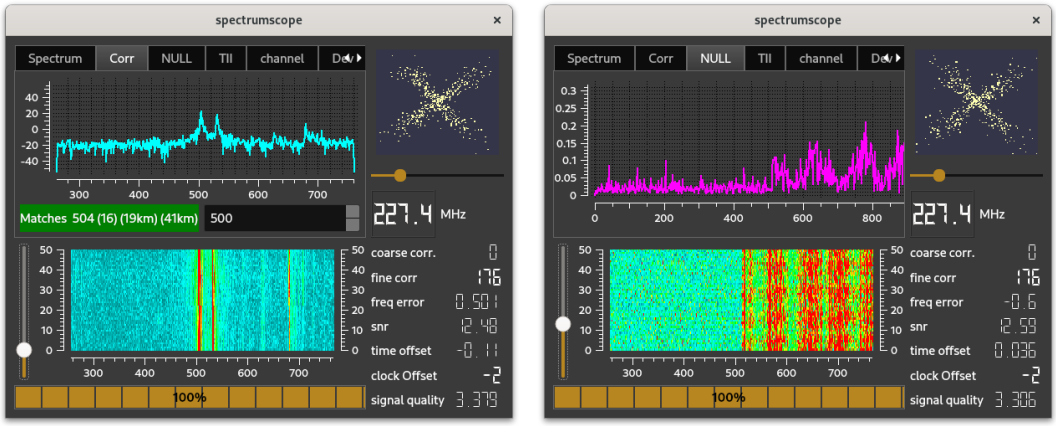


Figure 5: Correlation scope and NULL signal

The first step in decoding DAB signals is *synchronization*, i.e. finding where the actual data of a DAB frame starts in the incoming sample stream. The NULL scope (see figure 5, scope right) shows the samples in the transition phase from the NULL period to the first data block. To achieve such a synchronization a form of *correlation* is used (figure 5, scope left).

Ideally the maximum in the correlation is on (or about) sample 504 of the current segment of the incoming stream. The correlation scope here shows more than one peak, indicating that more than a single transmitter is received. The software detects here three "reasonable" peaks, it is up to the software to synchronize with the strongest signal.

If the current transmitter is "known", i.e. we know its name and distance, the distance and the estimated distances of other peaks to the receiver site are shown. In the picture here, the transmitter transmitting the strongest signal has a distance of 16 km, the others, 19 km resp. 41 km to the receiving location

As known, most transmitters send some identification data, encoded in the NULL period of the DAB frames. This data, Transmitter Identification Information (TII) consists of two numbers, a main Id and a subId, unique for each transmitter in a given country. The TII scope shows the spectrum of the NULL period, that is where the TII data is to be found. In this case, the software was able to detect that the TII data here can be decoded as (04 05) nd alternatively (04 21). The so-called *mainId* and *subId* (04 05) belong to a transmitter in Rotterdam.

Depending on the configuration there is (or not) a 5-th scope, a so-called channel scope

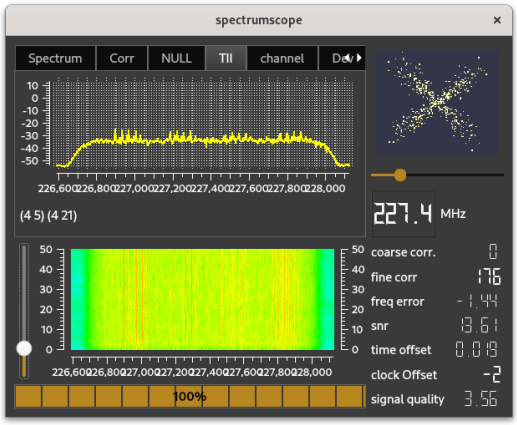


Figure 6: TII scope

(see figure 7[1]). The channel scope shows the difference between the signal as it is transmitted and how it is received. DAB signals may suffer from interference (more than one transmitter transmits the same signal) and reflections (signals arriving over different signals paths the receiver).

Both the amplitude and the phase are affected. The first data block of a DAB frame has predefined data, so an estimate of the "channel" is made by looking at the transformation of a selected set of carriers between transmitter and receiver. The cyan coloured line in the picture shows the differences in amplitude, the red line the phase offset[2].

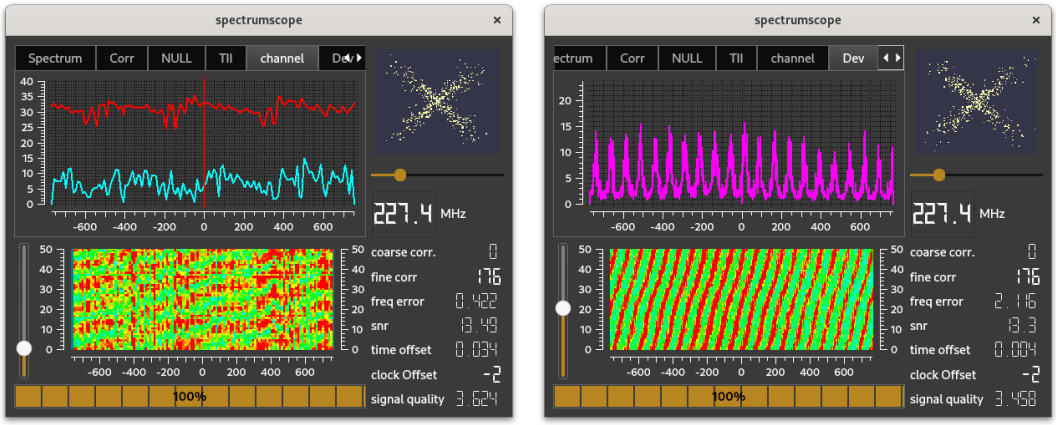


Figure 7: channel scope

The *deviation scope* shows the offsets, expressed in Hz, computed from the phase offsets in the decoded carriers. Ideally the offset in the decoded carrier is zero. Higher offsets means higher chances on errors in the decoding.

[1]While this is in the precompiled version, it requires the availability of some more libraries. By default the configuration does not include this scope

[2]A DAB block counts 1536 carriers, so the picture only shows a fraction

The widget shows further some numbers and a progress bar. The numbers, from top to bottom are

1. The *frequency correction*, computed from a DAB frame and applied to the data for the next DAB frame. Qt-DAB is able to correct frequency errors to up to 25 Khz, here the error is pretty small;

2. the measured *signal/noise ratio* (measured by looking at the signal strength in the NULL period and the period that data is transmitted);

3. the *remaining frequency offset*, which - obviously - should be small, but shows the effect of correcting incoming samples with an offset found earlier;

4. the *time offset*, i.e. the error in the sample clock;

5. the *clock offset*, here we measure how many samples we are short (in this case) or are too many per second.

6. the *quality* of the decoded signal, compared to the "ideal" signal of 4 dots, one in each quadrant. Higher values indicate a gigher signal quality. The plot on figure 4 showed an (almost) maximum value of 6.

Finally, the *progressbar* tells the success rate of decoding the FIC (Fast Information Channel) data. If the percentage is less than 100 percent, then apparently something is wrong with the signal and successfull decoding the payload is highly unlikely.

4.3 Technical details

The technical details widget shows data of the currently selected (audio)service, see figure 8.

On top of the widget, the name of the current service is repeated. Below this line, there are two buttons, both for saving audio data of the current service. 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 button *dump audio* - when touched - instructs the software to dump the audio output into a ".wav" file, i.e. a PCM file, with a samplerate of 48000 samples/second.

The bottom of the widget shows the spectrum of the audio output, it shows that the audio frequency goed to app 15 to 16 KHz.

The three progress indicators above the spectrum display (for MP2 output there will be only one) show the successrate of the different steps in the transformation from decoded DAB data to audio. The steps are:

- Frame recognition and extraction, DAB+ audio is organized in frames, recognizing frames in the input stream requires some testing and verification;

- *Reed Solomon* decoding. The indicator tells the successrate of the Reed Solomon decoder over the last 25 frames;

- the AAC decoder does the actual transformation from input bits and bytes to PCM samples.

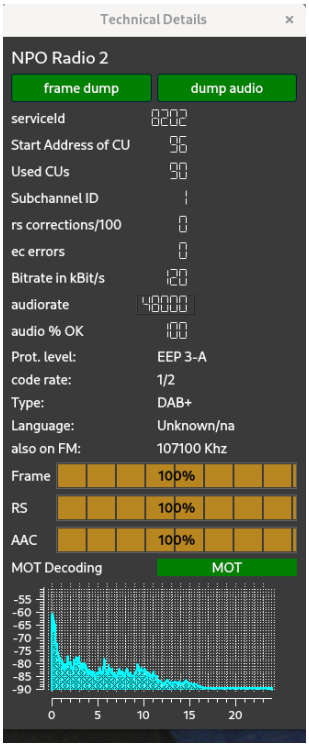


Figure 8: Technical details

Most of the other data that is displayed speaks for itself, a few less obvious numbers:

- *rs corrections* displays the number of corrections, performed by the Reed-Solomon decoder in the last 100 frames. The Reed Solomon decoder operates with frames of 120 bytes, the last 10 of them being parity bytes, and is able to correct up to 5 byte errors in a frame;
- *ec errors* displays the number of errors detected *after* the Reed Solomon decoder has repaired the errors (of course, if the parity bytes in the data contain errors, it is hard to see how an error free audio frame can be constructed);
- *audio % OK* displays the percentage of audio frames that reaches the audio output. If the AAC decoding fails, no data is sent to the audio output (usually the soundcard).
- *also on FM* is shown - with a frequency - if the audio is also transmitted on FM and this transmission information is encoded in the DAB signal;
- *audiorate* indicates the audiorate of the decoded AAC (or MP2) data. The audiorate for the output to the sound device or file is always converted to 48000;
- The *green label* with text "MOT" shows that the service carries MOT data, and usually the default slide on the main widget will be replaced by the slide(s) provided by the service. If no MOT is carried, the label will be "red".

4.4 Configuration and control

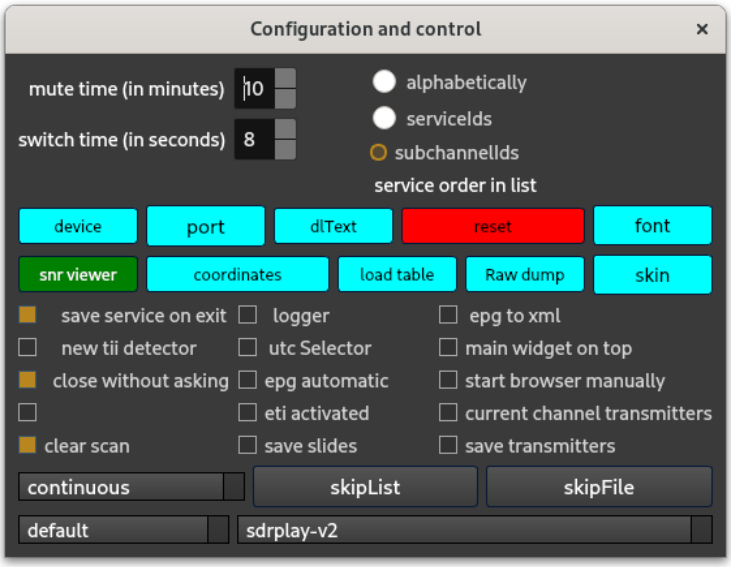


Figure 9: Configuration and control

The configuration and control widget is the widget with the buttons and checkboxes for influencing the configuration of the decoding process. The settings - together with other settings generated by the system - are stored in the ".ini" file.

- *mute time* tells the system what the duration should be in muting the audio;
- *switch time* tells the system what the waiting time should be after switching a channel before the software is convinced that the channel does *NOT* contains DAB data;
- *service order* tells the system in what order the services should be displayed on the main widget;

There are two rows with push buttons

- *device* is a button controlling the visibility of the device widget; A device widget usually shows controls for setting e.g. gain, and once these are set, there is in general no need to have the widget visible;
- *port* allows setting *another port number* than the default one used otherwise for the http handler, i.e. for communicating with the webbrowser when displaying maps and data on maps;
- *dlText* controls an option to save the data from the *dynamic label* into a file. Touching it will show a file selection menu, touching it again will close the file. If a file is selected, the texts in the dynamic label are stored in the file;
- *reset* will stop all activities and do a restart;

- *font* allows selection of a font that is used to display the names of the services. Changing the font will be effective the next program invocation;
- *snr viewer* controls the visibility of a small display, showing graphically the progress in time of the SNR;
- *coordinates* allows specifying the local coordinates, used to compute distance and azimuth to transmitters.
- *load table*, if correctly configured, this allows (re)loading a new instance of the TII database (see section 5);
- *Raw dump* (Obsolete) allows selecting a file to which the input data will be saved in a PCM file. Use xml files instead.
- *skin* allows selection of a skin for the displayed widgets. The selection will be effective the next program invocation;

Below these buttons there is a list of 14 check boxe (in the column left one entry is unused)

- *save service on exit* tells the system to save the channel and service name on program termination, and to use these on program start up to initialize the system;
- *logger* obsolete;
- *epg to xml* tells the system to map EPG data using an imported library to xml. Note that it is known that the imported library has a bug that might lead to a crash of the program;
- *new tii detector*. TII detection may use a second algorithm that detects TII data sometimes earlier, but generates more erroneous TII values;
- *utc Selector*, when enabled shows time as UTC rather than local time;
- *main widget on top*, as it suggests, enabling this ensures that the main widget *always* will be on top. Note that it might cause problems on Windows, since newly generated widgets will be on top and therefore be covered by the main widget. Not ideal if the new, covered, widget asks for confirmation;
- *close without asking* does what it suggests;
- *epg automatic*. Some ensembles carry - next to regular services - an EPG (Electronic Program Guide) service. If enabled the system will execute that service in the background;
- *start browser manually*. By default, on enabling the http service, the local default browser will be started. If this option is enabled, the user has to start his/hers favorite browser;
- *eti activated*. Since V5 there is an option of generating an ETI file from the current channel input. If this option is emabled, the *scan* button on the main widget is changed in an "eti" button, with which the eti generator can be started and stopped;

- *current channel transmitters* if enabled, the transmitters shown on a map will only be the ones belonging to the current channel;
- *clear scan*. On scanning (single scan, see below) the service names envountered will be added to the scanlist that can be made visible on the main widget. If this option is activated, that list will be cleared on a new scan;
- *save slides* does what it suggests;
- *save transmitters*. If checked, transmitter names will be saved (see section 5).

Scanning, scan modes and skip files The second to the bottom line shows three selectors all related to scanning. The first one defines the *Mode*, there are three scan modes

- *single scan*, a scan is performed from the first up to the last channel. Data is shown and can be saved in a ".csv" file.
- *scan to data*, a scan is performed until (a) a channel with DAB data is detected, or until the end of the list;
- *continuous*, the scan will start and continue until stopped. In this Mode only a single line of data is generated per channel with detected DAB data;

Wgen scanning, it is usually known on which channels there is no DAB data. As an example, it is quite useless here to scan the channels 13A .. 13F. A *skipList* can be filled in where it is indicated which channels is to be taken into accounf, and what channels can be skipped.

Of course directing the antenna to a different direction brings in new opportunities, so Qt-DAB provides the possibilities to apply a so-called *skipFile*.

bottom line The bottom line of the configuration and control widget contains two selectors, that are usually not modified often.

- the *audio out selector*, the selector is filled by the underlying sound system;
- the *device selector*, the configured devices are in the list here.

The settings of these selectors is always stored in the ".ini" file and used in initialization.

4.5 Coloring buttons and scopes

Selecting a color for buttons and the scopes is personal. While not essential for functioning, the ability to choose one's on color scheme seems important to me. So, a color for the different buttons on the *main widget* and on the *configuration and control widget* can be set (and subsequently changed). Touch the button with the *right mouse button* and a small widget appears in which first the background color, and second the text color can be selected (see figure 10.

The same applies to the scopes in the spectrum widget, clicking with the *right mouse button* on the field will show a small selection widget with which a *background color*, a *grid color* and a *curve color* can be set. The color settings are immediately effective and maintained between successive program invocations.

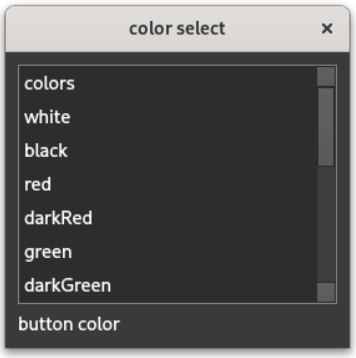


Figure 10: Button color selection

5 Contents, scanning and maps

5.1 Introduction

Of course, Qt-DAB can display a description of the content of the currently selected channel. Next to that, Qt-DAB offers extensive options for scanning. As mentioned above, there are three modes, *scanning until DAB data is found*, just *scanning a single round over all channels in the band*, and *continuous scanning*.

5.2 Looking at the content of the current channel

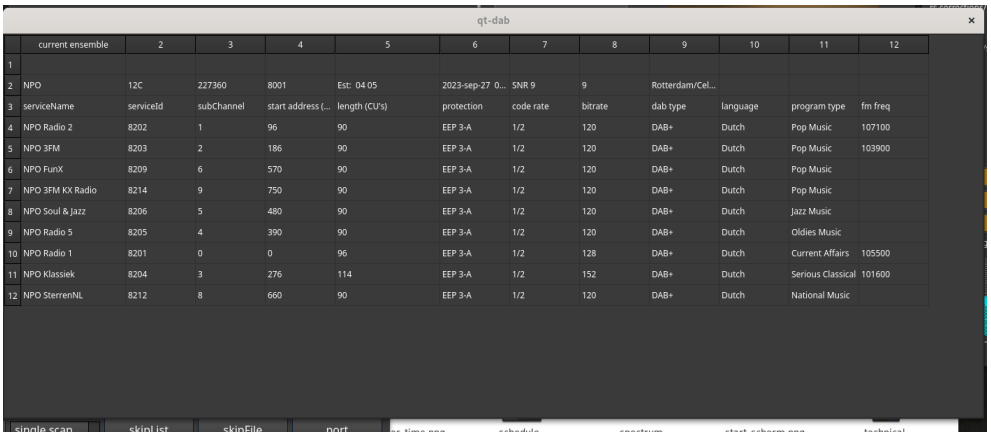


Figure 11: Content description

Of course, even without scanning, it is possible to get a view on what services are available in the currently selected channel. Next to the obvious list of service names as given on the main widget, one might have a look at the "content", i.e. details of the ensemble and its constituents.

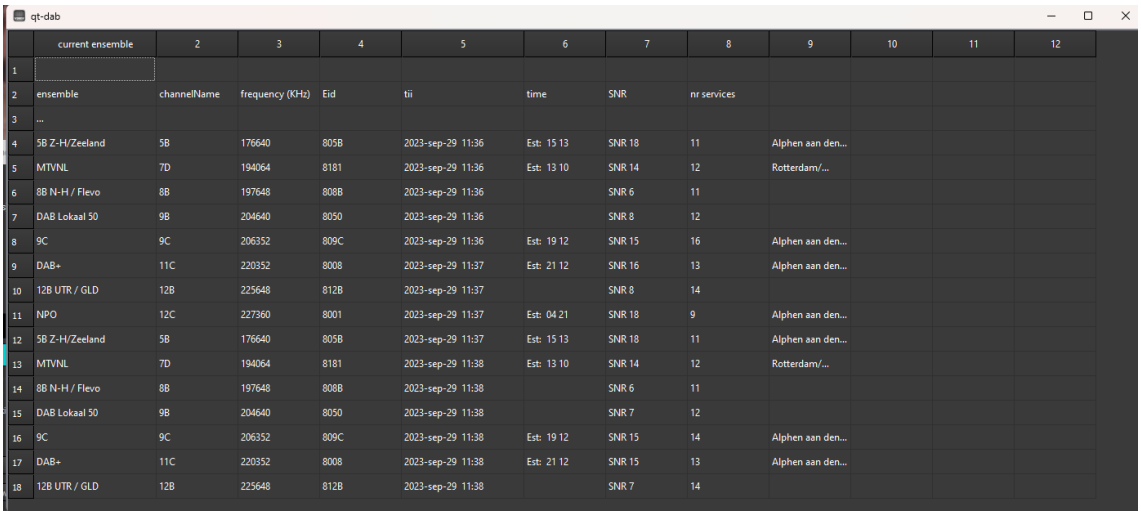
Picture 11 shows the content of the NPO ensemble, it shows that the TII identifiers at the time of reception were 04 05, the transmitter is located in Rotterdam, that 9 services were

detected and the SNR was 9.

For each of the services (NPO does not transmit a data service anymore) the known data is printed. Note that it might take some time before all data items are known. Whether or not the service description shows an FM alternative FM frequency might not be known before a few seconds have passed.

When scanning in mode "single scan", the output consists of a content description for each channel where DAB data is found. Note that the time spent on collecting data for each channel that contains data is limited, and elements such as program type or fm frequency might not yet be recovered when switching to the next channel.

When scanning in mode *continuous*, the data per channel is restricted to a single line, as shown in figure 12.



	current ensemble	2	3	4	5	6	7	8	9	10	11	12
1												
2	ensemble	channelName	frequency (KHz)	Eid	tii	time	SNR	nr services				
3	...											
4	5B Z-H/Zeeland	5B	176640	805B	2023-sep-29 11:36	Est: 15 13	SNR 18	11	Alphen aan den...			
5	MTVNL	7D	194064	8181	2023-sep-29 11:36	Est: 13 10	SNR 14	12	Rotterdam/...			
6	8B N-H / Flevo	8B	197648	808B	2023-sep-29 11:36		SNR 6	11				
7	DAB Lokaal 50	9B	204640	8050	2023-sep-29 11:36		SNR 8	12				
8	9C	9C	206352	809C	2023-sep-29 11:36	Est: 19 12	SNR 15	16	Alphen aan den...			
9	DAB+	11C	220352	8008	2023-sep-29 11:37	Est: 21 12	SNR 16	13	Alphen aan den...			
10	12B UTR / GLD	12B	225648	812B	2023-sep-29 11:37		SNR 8	14				
11	NPO	12C	227360	8001	2023-sep-29 11:37	Est: 04 21	SNR 18	9	Alphen aan den...			
12	5B Z-H/Zeeland	5B	176640	805B	2023-sep-29 11:37	Est: 15 13	SNR 18	11	Alphen aan den...			
13	MTVNL	7D	194064	8181	2023-sep-29 11:38	Est: 13 10	SNR 14	12	Rotterdam/...			
14	8B N-H / Flevo	8B	197648	808B	2023-sep-29 11:38		SNR 6	11				
15	DAB Lokaal 50	9B	204640	8050	2023-sep-29 11:38		SNR 7	12				
16	9C	9C	206352	809C	2023-sep-29 11:38	Est: 19 12	SNR 15	14	Alphen aan den...			
17	DAB+	11C	220352	8008	2023-sep-29 11:38	Est: 21 12	SNR 15	13	Alphen aan den...			
18	12B UTR / GLD	12B	225648	812B	2023-sep-29 11:38		SNR 7	14				

Figure 12: Result from continuous mode

5.3 The TII database, transmitter names and distances

As mentioned earlier, the NULL period preceding the data in a DAB frame usually contains the encoding of the transmitter. Since DAB is transmitted with a large number of lower power transmissions, all using the same frequency, an encoding of the TII (Transmitter Identification Information) is added to the NULL period.

A database exists (and is continually updated) with program and location information of DAB transmitters (see www.FMList.org). The owner of the database kindly provided an URL to extract the relevant data for DAB transmitters for use with Qt-DAB only. Due to the latter requirement, the code to access the database can not be open source, and is therefore not part of the source tree.

However, the code is included in the precompiled versions. The "load table" button on the configuration and control widget will ensure that a copy of the freshly axquired database is installed in the user's home directory.

Qt-DAB can be configured to use a library with which accessing a local copy of the database is possible, that is why a local copy of the database is provided for in the source tree.

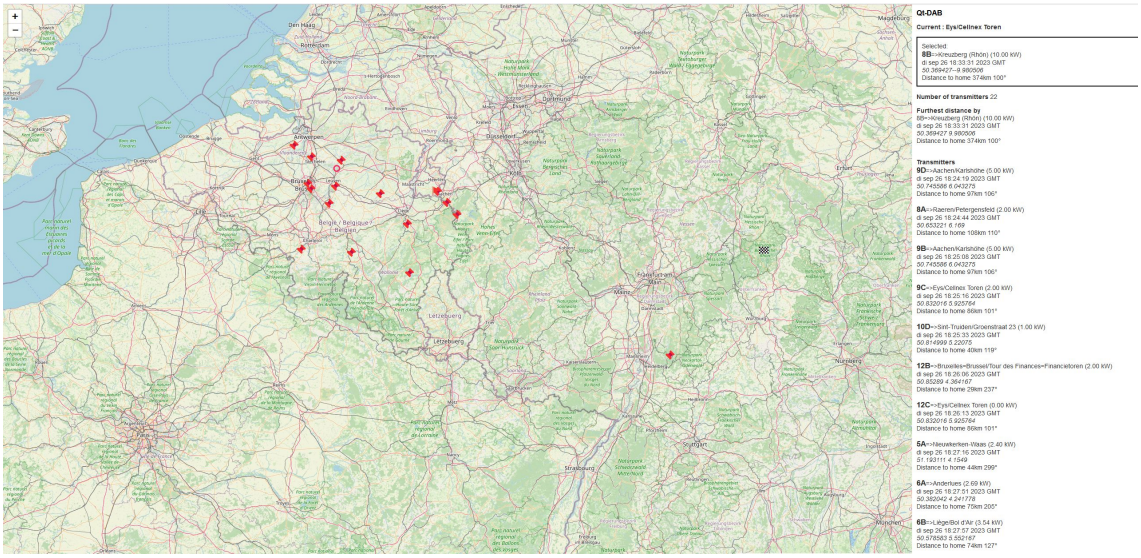


Figure 13: Transmitters on the map

Given that access is obtained to a local copy of the database, and given that the home position of the system where Qt-DAB is running is known, Qt-DAB displays the name of the transmitter received (by looking up the TII code in the database), and will compute an estimate of the distance and the azimuth.

5.4 Transmitter names and a map

Qt-DAB offers the possibility of displaying a map on a webbrowser with the home location and the names and locations of the transmitters seen

The button "http" on the main widget will - whenever the database is accessible and the home location is known - issue a command to start a web browser, and will send data about the home location and the transmitters received to the browser program as shown in figure 13[3]. Note that while by default the "default" browser, but the *configuration and control* widget contains a checkbox with which the user has to choose (and start) a webbrowser. The right hand side of the map contains a list of transmitters, with for each transmitter the name, the precise location and if known the transmission power. The description is augmented with the distance and the azimuth from the specified home to the transmitter. In figure 14 part of the list is shown in more detail.

The picture shows that the transmitter with the furthest distance was found on 374 Km, that the number of transmitters - collected for different channels - is 22.

The transmitter descriptions can be saved in a file. If the checkbox *save transmitters* is checked, the http handler will - on start up - show a file selection menu, for selecting a file in which the descriptions are stored.

[3]The map is made available by Herman Wijnants

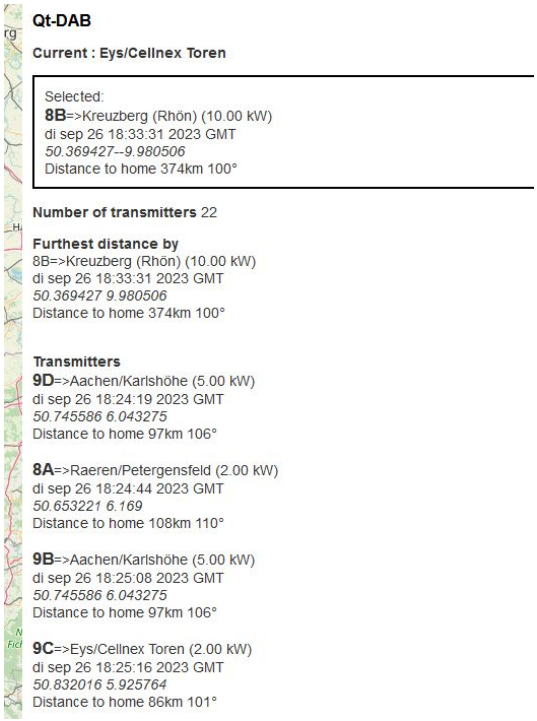


Figure 14: list of transmitters

6 Scheduling in Qt-DAB-6

When working, I am usually listening to a service with continuous music, but I want to hear the news bulletins on the hour on another service. Of course in 9 out of 10 cases I am late. That is why Qt-DAB has a mechanism to switch over to a specified activity on soecified times. The scheduling option allows scheduling an operation to be performed within the next 7 days on a specified time. *Since the scheduling mechanism is part of the program, Qt-DAB should run for the scheduling to be effective.* Scheduling settings are kept between program invocations, on program startup, the software will remove schedule instructions that refer to the past.

Touching the schedule button shows a selection widget, as shown in figure 15

The first step is selecting a service name or operation. The service names are those in the currently selected channel, and those in the preset list. Furthermore, some operations can be selected

- *nothing*, with the obvious semantics;
- *exit*, which, when executed will terminate the execution of Qt-DAB;
- *framedump* for scheduling the recording of the AAC frames for the service *that is active at the moment the scheduling time is reached*;
- *audiodump*, the same for the PCM output;
- *dltext*, for scheduling the start of saving the dynamic label text in a file;

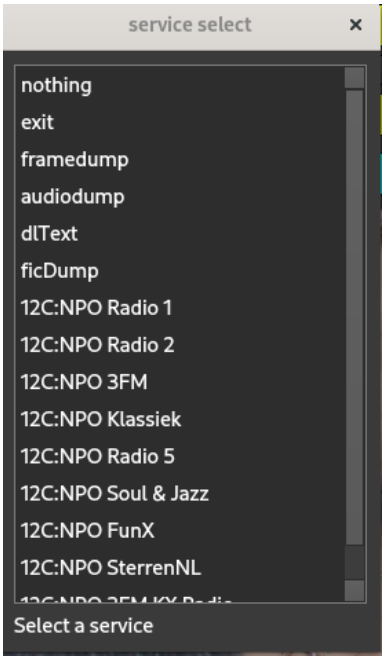


Figure 15: Schedule selection

- *ficDump*, for scheduling the start of the dump of the FIC data into a file.

Note that executing the command when a previous invocation of that command is still active, will terminate the execution of the operation.

Having selected an operation of service, the next step is specifying the time and day of the operation to be executed. A small widget shows, that allows setting day and time. see figure 16

If the list of scheduled events is not empty, it will be shown in a separate (small) widget, see figure 16.

7 Supported input devices

Qt-DAB supports a variety of input devices, the Adalm Pluto, the AIRspy, the hackrf, the

Figure 16: Schedule time selection and the schedule list

limeSDR, RT2832 based sticks and SDRplay RSP devices. Furthermore, there is support for

the rtl_tcp server, 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.

7.1 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 if API 3.10 is installed on Windows, the 2.13 library is not accessible.

Figure 17: Qt-DAB: The two control widgets for the SDRplay

As figure 17 shows, the control widgets for the two different versions resemble each other, their implementation differs considerably though. Both have spinboxes for setting the *if gain reduction*, the *lna state* and a *ppm offset*.

Since Qt-DAB is capable of correcting th frequency, there is no actual need for setting a value here.

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*.

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.

Note that when running Windows and version 3.09 or higher of the SDRplay support library, the version 2 library is disabled. Note further that the 2.13 library does not provide support for the SDRplay RSPDx.

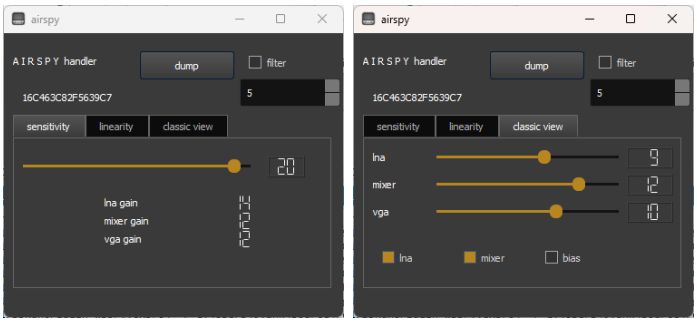


Figure 18: Qt-DAB: Widgets for AIRspy control

7.2 The AIRSpy

The control widget for the AIRspy (figure 18, right) 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 18 left 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.

7.3 The hackrf

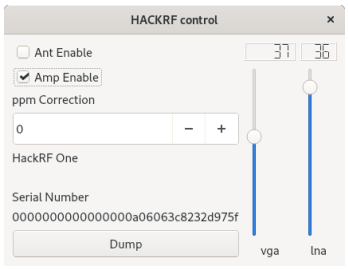


Figure 19: Qt-DAB: Widget for hackrf control

The control widget for hackrf (figure 19) shows, next to the Serial Number of the device, a few sliders, a few checkboxes, a spinbox and a push button.

- 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.

7.4 The LimeSDR

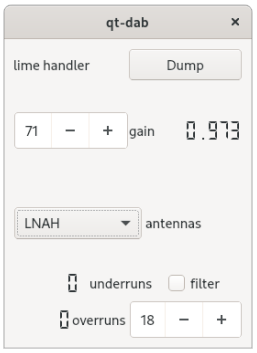


Figure 20: Qt-DAB: Widget for Lime control

On selecting the LimeSDR (if configured), a control widget for the LimeSDR is shown (figure 20). 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 3.

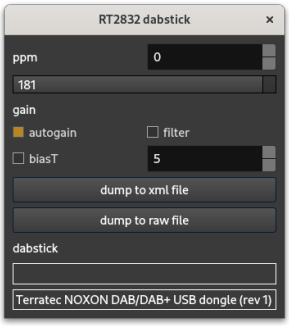


Figure 21: Qt-DAB: Widget for rtlsdr device

7.5 The RTLSDR stick

On selecting the dabstick (i.e. RT2832 based devices) (if configured), a control widget for the device appears (figure 21).

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 for 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 3.

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

7.6 The Pluto device

When selecting *pluto*, a widget (figure 22) 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 furthermore three buttons:

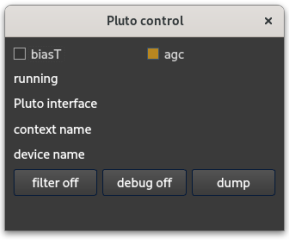


Figure 22: Qt-DAB: Widget for Adalm Pluto device

- 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[4] - 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.

7.7 Support for Soapy

Figure 23: Qt-DAB: Widget for soapy

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

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 24) when applied to the Soapy interface for the SDRplay contains the obvious controls, similar to that of the regular control for the SDRplay.

7.8 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

```
rtl_tcp_port=XXX
```

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

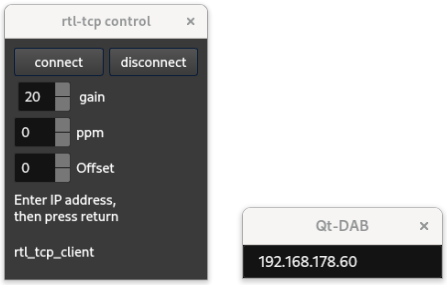


Figure 24: Qt-DAB: Widget for rtl_tcp

7.9 File input

Qt-DAB supports both *writing* raw input files and *reading* them back. Writing a file as PCM file is initiated by the *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.

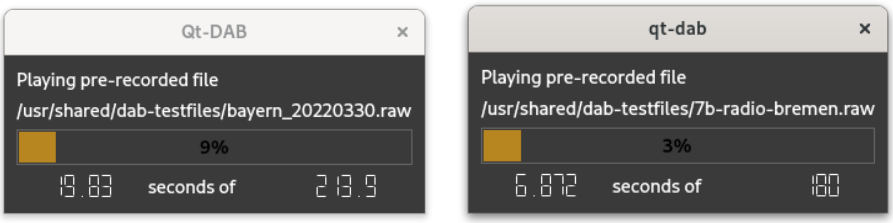


Figure 25: Qt-DAB: Widgets for file input

When selecting file input ".raw" or ".wav", a simple widget is shown (figure 25), with as indication the number of seconds the file is being played.

Since processing an xml file implies some interpretation, the widget (figure 26) for control when reading an xml file is slightly more complex. It contains - next to the progress in reading the data - a 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.

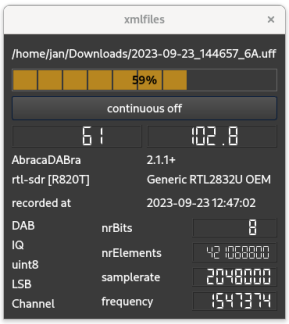


Figure 26: Qt-DAB: Widget for xml file input

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

8 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;
- Przemysław 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.