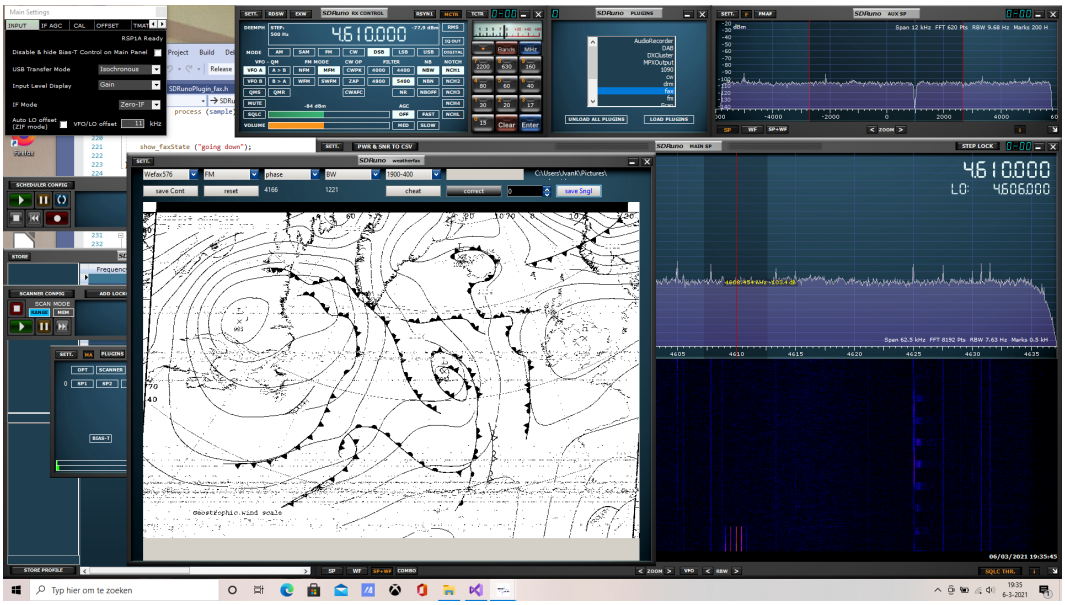
Plugins for the SDRuno environment

user manual

Jan van Katwijk
Lazy Chair Computing
The Netherlands
J.vanKatwijk@gmail.com

August 6, 2023



Contents

1 Introduction

SDRuno is a platform running under Windows, supporting Software Defined Radio applications with SDRplay devices. SDRuno provides a uniform interface to the different devices (RSP 1A, RSPII, RSPdx, RDSduo).

SDRuno also provides a *standardized* API, an Application Programmers Interface, for *plugins*, i.e. program fragments carrying out some specific tasks, embedded in the SDRuno environment. Plugins use the SDRuno as a "front end", the front end that will take care of device settings, tuning, displaying spectra, (some) filtering and - if needed - decimating the samplerate.

I am not a big supporter of Windows, personally I prefer Linux both as development and execution environment, and in the past years I have written quite some software for running under Linux. One of the software packages written for Linux was a so-called "swradio", i.e. an interface to devices together with a number of decoders, some simple, some more complex. The plugins, described here for use with the SDRuno platform, are (almost) copies of the decoders for my Linux (and Windows) swradio.

The plugins are

SDRunoPlugin_cw, a simple decoder for CW transmissions;

SDRunoPlugin_cwSkimmer, another decoder for CW transmissions;

SDRunoPlugin_psk, a decoder for transmissions in a variety of PSK modes;

SDRunoPlugin_rtty, a decoder for transmissions in a variety of RTTY modes;

SDRunoPlugin_navtex, a decoder for amtor-B, i.e. navtex, transmissions;

SDRunoPlugin_fax, a decoder for weatherfax transmissions;

SDRunoPlugin_drm, a decoder for DRM30 transmissions;

SDRunoPlugin_ft8, a decoder for FT8 transmissons;

SDRunoPlugin_acars, a decoder for acars transmissions;

SDRunoPlugin_acars_M, another plugin for decoding acars transmissions with slightly more functionality;

SDRunoPlugin_apt, a *very experimental* plugin for decoding apt transmissions.

Navtex transmissions are usually on 518 KHz, weatherfax transmissions in my region on 3588, 4610, 7880 and 8020 KHz. RTTY is still in use for some meteo transmissions, and on amateur bands. Of course cw, psk, rtty and ft8 can be heard/seen in the amateur bands. DRM30 - while not very popular in these regions - can be heard in the 15 and 7 MHz regions. The frequency ranges for both *acars* and *apt* are different, data in these modes is transmitted in the 130 MHz band.

2 Why these plugins

Long time ago building a radio was fun, with glowing tubes and the soft hissing on switching the radio on. After retiring I started again, but electronics in these days is completely different from the electronics in my youth, and I let it go soon.

In the early years of my professional life I did a lot of programming (of course computers in the 70-ties was different, a PDP-8, 9 or 11 are somewhat simpler compared to a modern PC) in assembler and languages like BCPL, and even C. So I picked up programming again and developed stuff - using Linux as development and application environment. Ever since 2014 I am using SDRplay devices, and when SDRuno matured it seemed interesting to transform a number of decoders - running as separate program or as decoder mode in Linux programs - to plugins for SDRuno.

3 Etiquette

The plugins are all open source and available for anyone who wants to use it, however, they are not freeware, they are available under a license. The license is the GPL V2 license and it roughly states that: *the software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.*

Since a plugin is software, there is always a chance that it contains an error. Of course not all errors show themselves all times, so plugins that run without problem on my W10 system, may cause a problem on another platform, there may be a problem with the installation, some incompatibility between systems, or some unforeseen combination of events, causing the plugin to malfunction.

Experience learns that there are two different styles in the reaction I get when such a thing happens:

> From time to time I get a mail with some (usually unreadable) output and just the text *it does not work*, sometimes even followed by a few "remarks" on the alledged quality of the software.
>
> most other times I get a mail, reporting issues with a plugin; sometimes the plugin fails to install, sometimes it does not show the expected behaviour, followed by a question for help.

My standard answer on mails of the first category is: *Read the license, stop using the software and ask your money back.*

On mails of the second category I usually respond, and - until now - manage in most cases to solve the issue.

The bottom line being that I am happy to make the software available, no doubt about it. However, if the software does not do what you want or what you expect it to do, read the license before contacting me.

4 Installing the plugins

Installing a plugin is simply by placing it in the folder for *community* plugins. While it may differ on different installations, on my W11 system the folder is in the folder *Documenten*. Location is determined by installing the SDRuno package, so if unsure, ask the SDRplay people about it.

The SDRuno panel contains a widget that contains the names of the plugins that can be loaded, and the control widget contains a label in the top with the text *plugins*. On the picture on the front page the widget showing (some) plugins in in the middle of the top row. /subsectionInstalling required dll's Plugins are implemented as *dll*, i.e. *Dynamic Load Libraries*, entities that dynamically can be added to (or removed from) a running program.

Some of the plugins need additional dll's to work, these dll's might or might not be installed on your system already. However, to avoid problems it is strongly suggested to install the dll's that are available in the folder *required dll* in the folder `C:\Program Files (x86)\SDRplay\SDRuno`.

Some of the dll's are required for DRM decoding, others for handling output for (".wav")files.

5 The simple plugins

Plugins for cw decoding, psk decoding, rtty decoding and navtex decoding a relatively simple. They all use the IQ decoder output, i.e. have an input rate of 192000. The drawback is that the main spectrum shows a spectrum with a width of 2MHz, and a signal with a width of, say 100 Hz, is not always easy to identify on such a spectrum.

5.1 CW decoding

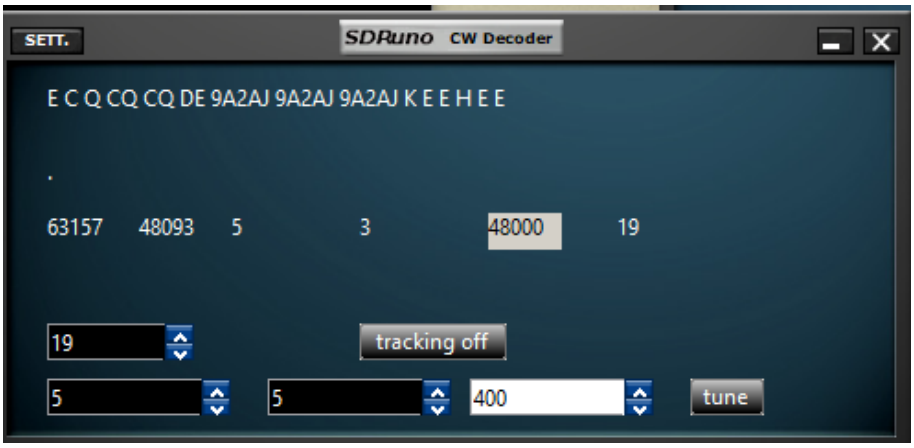
5.1.1 introduction

continuous wave is one of the most simple ways to encode data, just switch a carrier on or off. The morse alphabet is expressed in *dots*, *dashed* and *spaces*. The duration of a *dash* is (well, should be) three times that of a *dot*, space between the dashes and dots has the same duration of the dot, and the spacing between letters 3 times, so decoding seems trivial.

The obvious problem is to decide when there is a carrier and when there is only noise. Having done that, decoding is merely a mattter of counting samples and deciding what the current element is. Note that in practice the duration of the dash is not exactly 3 times the duration of the dot, so decoding morse code requires some educated guessing.

In spite of the age of the technology, CW decoding is still quite vivid. On 14020 to 14040 KHz it is usually overcrowded with CW signals.

5.1.2 The plugin



The plugin (see picture) shows - if possible - decoded text in the top lin, the numbers - here 63157 and 48093 - show the guesses for the duration of a *dot* and *space* element (in micro seconds). Ideally they are the same and it is clear that there is a difference in the measurement result.

The computed *number of words per minute* is - given these results - 19, a number shown on the right side of the second line.

The second line shows two more numbers, here 5 and 3, indicating the strength of the signal and the strength of the noise.

The spinbox showing *19* tells the software that the expected number of words per minute is 19, the software sets sets the limits for guessing the duration of dot and space based on this number.

The bottom line spinboxes set the filter depth and the noise strength values, both used in guessing the border in sigbnal strength between space and data.

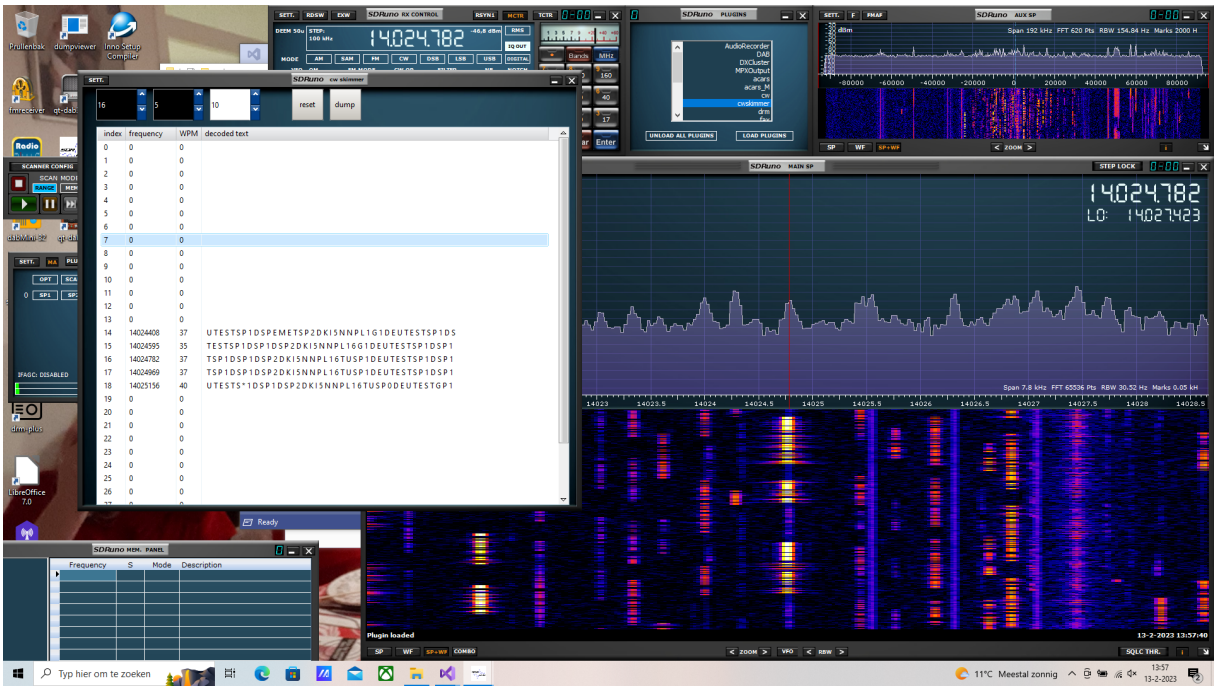
Since tuning to the signal is not always easy, the cw decoder provides a tuning assistent, touching the button *tune* tells the software to compute the spectrum and switch the frequency to the strongest signal in a range of 2 * 400 (the number in the spinbox) Hz.

Of course, the main spectrum shown by SDRuno has an excellent zooming facility, making it much easier to tune to the right signal.

5.1.3 The cwSkimmer

While the cwDecoder looks at samples in the time domain, the approach in the cwSkimmer is to look in the frequency domain. The samplerate of the incoming samples is 192000, the cwSkimmer computes per two milliseconds, i.e. 500 times a second, a spectrum with 1024 bins, The width per bin is then slightly less than 200 Hz.

In a continuous stream of computed spectra, the signal energy in the bins are measured and duration of space or data is determined. Based on these figures a decoder tries to detect dots, dashes and spaces and tries to map these into text.



The plugin shows the result of a (small) row of bins, the number and place of these bins within a small central part of the spectrum can be selected by the user, using the two spinboxes at the topline. The spinbox top left defines the central bin, the spinbox to the right the number of bins,

The spinbox, here showing the number "10" defines a threshold value for distinguishing between noise and data in the incoming signal.

The buttons *reset* and *dump* have their obvious meaning.

5.2 The psk plugin

5.2.1 Introduction

psk, i.e. *phase shift keying* is a technique where the data is encoded as shifts in the phase of the signal, with - at least theoretically - a signal with a constant amplitude. For amateur modes, there are two variants, *bpsk* and *qpsk*, standing for *binary* and *quadrature* phase shift keying.

Binary is - obviously - the simplest one, the phase shifts either 0 or 180 degrees, In the quadrature mode phase shift can be 0, 90, 180, 270 degrees. In the binary mode there is one bit per sumbol, in quadrature mode 2.

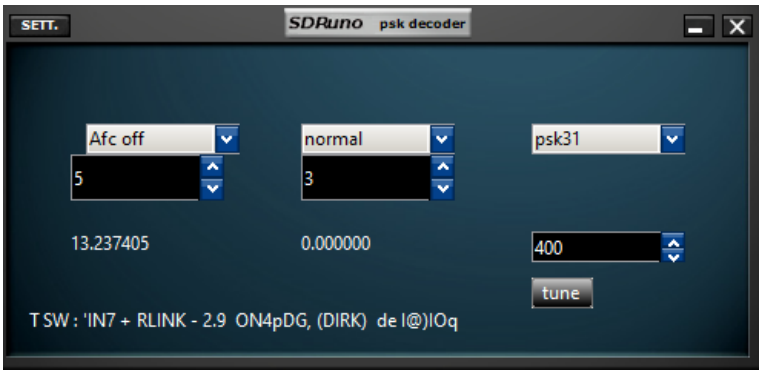
The rate for bpsk is 31.25, variants use the double or 4 times that rate. Qpsk has the same rate.

Decoding is not very difficult, decimate the incoming samplestream to e.g. 500 samples per second, then the duration for a single bit in bpsk (two bits in qpsk) is 16 samples. So, collect 16 samples, such that the first 8 are part of one symbol, and the last 8 are part of the next symbol. Then compute the phase difference between the two

parts and send the bit to a procedure that adds it to a sequence of bits for decoding.

The main problem is obviously the tuning, the width of the signal is a few tens of Hz.

5.2.2 The plugin



Here the decoded text is shown on the bottom line. The widget has two rows with selectors, the top row shows 3 comboboxes,

> selecting the *Afc*. If tuning is in range, the afc can be used to correct the offset, but do not expect miracles here;
>
> selecting *normal* or *reverse*, sometimes the encoding of bits wrt to phase and phase change is reversed;
>
> selecting the *mode*, i.e. bpsk31, qpsk31, bpsk63, etc.

Below these comboboxes there are two spinboxes, the left one for setting the degree of the filtering (note that a high degree of filtering causes averaging in the signal), the next one is a setting for a threshold value (i.e. when is the signal to be considered noise).

As with the cw decoder, a simple form of automated tuning is included, touching the *tune* button instructs the software to tune to the strongest signal within a range, indicated by the spinbox above the button.

5.3 The rtty plugin

5.3.1 Introduction

RTTY data is transmitted using a frequency shift between a *mark* and a *space* frequency. I.e. as with psk, a continuous carrier is transmitted, with - depending on the baudrate - frequency switches over a given range.

For amateur modes - btw it seems that rtty is less used than e.g. a decade ago - the baudrate is 45 baud, and the distance between mark and space is 170 Hz. The amateurband I am often listening to is around 14080 KHz.

From time to time one finds data transmissions, usually weather related - on non amateur bands on short waves.

Decoding rtty signals is not very difficult, tune in to the frequency in the middle between mark and space and count the durations of the mark and space signals. Since the difference between mark and sspace frequency is - for amateur modes - 170 Hz, tuning is easier than tuning a psk signal.

5.3.2 The plugin

The plugin has 8 control switches, and 4 indicators. The decoded text is shown on the bottom line of the plugin. The controls are

the *shift*, default set to 170 Hz, however, with a variety of options up to 1200 Hz;

the *rate*, default set to the amateur rate of 45 buad, but, here as well, showing a variety of options;

the *bits* per element, default set to 5 as used for common baudot encodings;

the *stopbits*, default set to 1, as used for common encodings;

the *parity*, which is usually *none*, but the combobox offers other options;

the *msb*, i.e. a selection between *msb most significant bit* or *lsb*, *least significant bit* first;

the *afc on* or off, the software measures the distance between the tuned frequency and the mark and space frequencies and is able to correct itself (a little);

reversed or not, tells whether or not mark frequency is the higher one or the space frequency.

The small number displays to the left give an indication of the measured frequency offset and the computed baudrate.

5.4 The navtex plugin

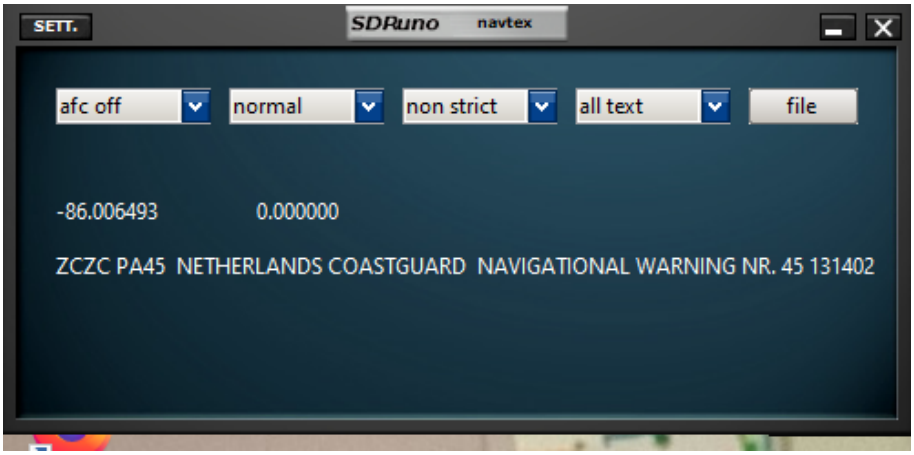
5.4.1 Introduction

NAVTEX (NAVigational TEleX) is a service for delivery of navigational and meteorological warnings and forecasts, as well as urgent maritime safety information (MSI) to ships.

The transmissions are layered on top of SITOR collective B-mode. SITOR-B is a forward error correcting (FEC) broadcast that uses the CCIR 476 character set. NAVTEX messages are transmitted at 100 baud using FSK modulation with a frequency shift of 170 Hz.

Other than with common RTTY messages, the text of the message is protected on two levels, a FEC protection on bit level and a (simple) protection on text level by preceding the message with a "ZCZC" prefix.

5.4.2 The plugin



While not shown in the widget, on starting the plugin it will set the tuned frequency in the SDRuno platform to 518 KHz (of course you can change that to whatever frequency you want.]par The controls on the widget are with 5 comboboxes

afc switch, as with the rtty decoder, a limited form of automatic frequency correction is possible, of course tuning should already be between the mark and space frequencies;

normal or reverse, see the rtty decoder;

non strict or strict, i.e. with the FEC switched on off;

all text or only text validated as message;

file touching this will show a file selection menu and save the text of the messages into the file. Touching the button when the file is open will close the file.

The two numbers give aan indication of the signal quality and the computed frequency offset for optimal decoding.

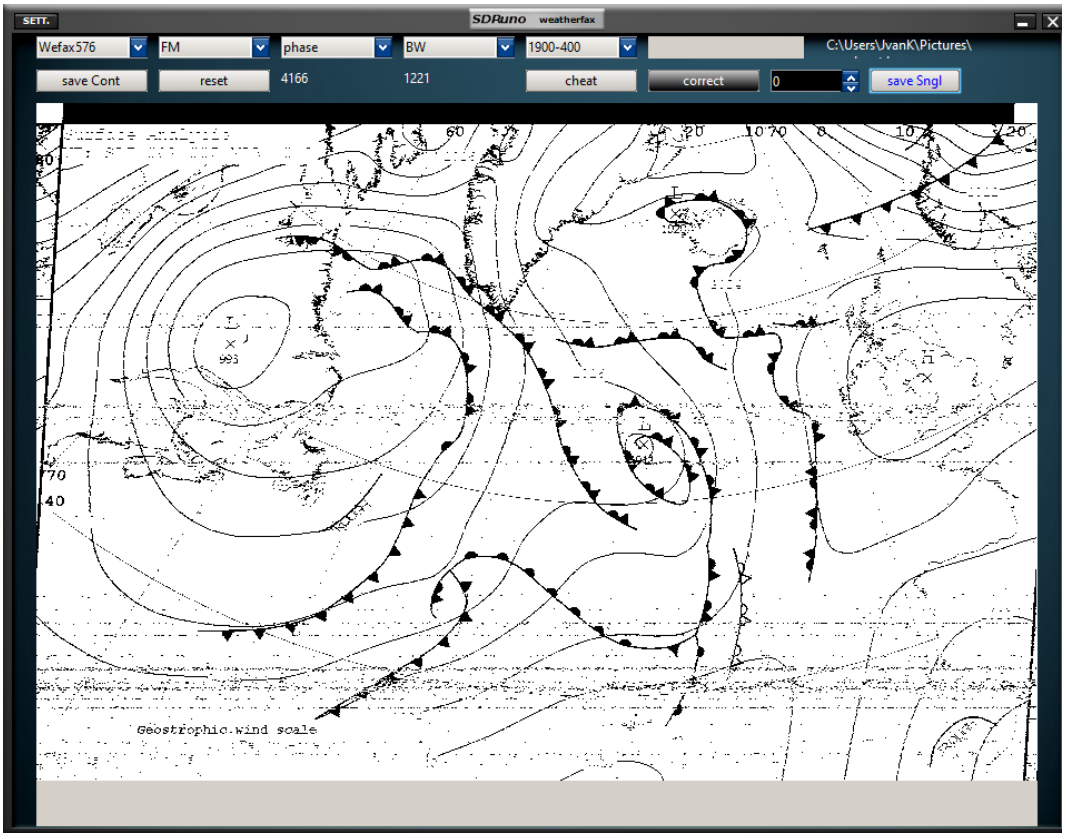
6 The weatherfax plugin

6.1 introduction

Weatherfax data is still being transmitted on shortwaves, typical frequencies that I receive are 3855 Khz, 4610 Khz, 7880 Khz and 8040 Khz. The encoding of a picture starts with a single tone (for the commom Wefax576 mode 300 Hz), followed by a number of lines with white margins left and right, and a black middle. These *synchronization* lines are followed by the lines of the actual picture. At the end of the picture another tone is transmitted, usually 450 Hz.

The picture can be quite lengthy, and since the transmission speed is 2 lines a second, a full transmission of a 1200 line weatherchart takes about 10 minutes.

6.2 The widget



The plugin operates in one of a few *states*, on startup the state will be *APTSTART*, after recognizing a start tone the state will be *PHASING*, on successful recognizing the

phases and synchronizing, the state will be *ON_SYNC* and the picture will be decoded. When recognizing an end signal (or after a predefined number of recognized lines) the state will be *FAX_DONE*.

The widget shows two lines with displays and controls and about 600 lines for the transmitted picture. The size of the picture displayed is *half* the real size, when saving the picture (see the selectors below), the picture in its original format is saved.

The controls and displays on the top line are

transport mode, by default *Wefax576*. Alternatively *Wefax288* can be selected;

modulation, by default FM, alternatively AM can be selected;

phase, the software assumes that the encoding of black and white in the signal is reversed;

black and white or color;

1900-400, with as alternative *1900-450*, tells the deviation of the signal wrt the central frequency. The *1900* number is here meaningless, one should tune to the exact frequency of the transmission rather than tuning with an offset of 1900 Hz.

The top line furthermore contains an indicator for the *state* (not shown here) and - if a file is selected - the (path)name of the file.

The second row contains

the *save continuous* selector, having enabled this, the software will continuously try to detect pictures and save results into a file;

the *reset* button, touching this will reset the mode of operation;

the *cheat* button, which will make the software think that synchronization is achieved, and will set the state to *ON_SYNC*.

the *correct* button, which will allow a user to correct the number of samples per line, for which

the *correction* value is used.

the *save single* selector, instructing the software to save the received picture into a file.

The number on the left tells the aberage frequency of the encoding of the current line, the other number tells the number of the current line.

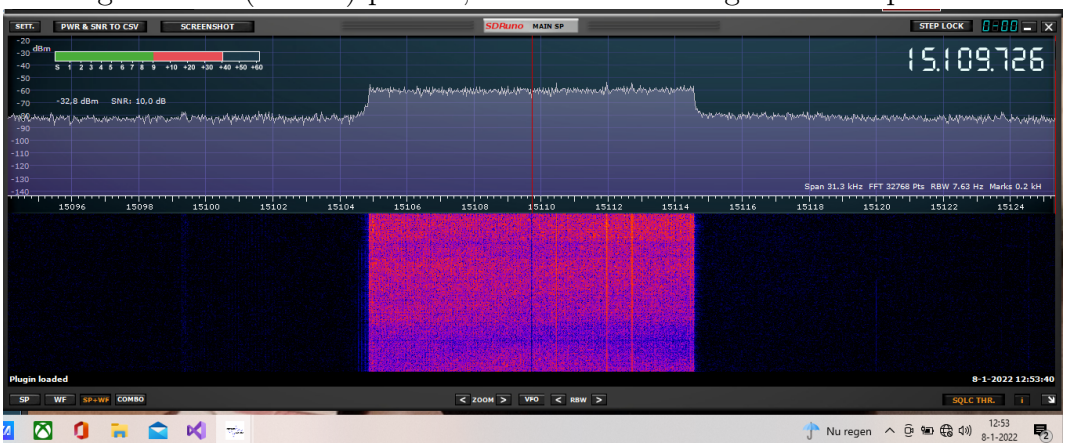
7 The DRM plugin

7.1 Introduction

DRM, Digital Radio Mondiale, is a type of digital radio that is transmitted on medium and shortwaves. Here in western Europe there are not many transmissions anymore, but some people believe that it has a future.

A DRM transmission fits within the transmission scheme of AM transmissions in short and medium waves. While there are some variations in bandwidth and mode, most transmissions (at least the ones I can receive) are just 10 KHz wide.

A DRM transmission is recognizable on the main spectrum of the SDRuno, rather than a single carrier with some sidebands, one sees about 300 carriers, close to each other, as in the picture. Distance between the carriers is app 45 Hz, and each carrier "carrier" part of the transpoted data. The major drawback of this type of technique is that tuning must be (almost) perfect, otherwise decoding will be impossible.



7.2 The plugin

The widget does not have a single control button, it only shows information it gets

from interpreting the incoming data stream/ Important in decoding DRM is *synchronization*, there are four steps in synchronization:

time sync tells whether or not the software is able to detect a bitstream looking more or less as a DRM stream;

FAC sync tells whether or not the software is able to decode some data in the input stream, giving information about the structure in the input stream;

SDC sync tells whether or not the software is able to extract information about the encoding of the service(s) in the input stream;

AAC sync tells whether or not the software is able to decode the data of the current service and create audio from it.

The four blocks on the right side of the widget give these indications. As can be expected, green is good, red is not. Successful decoding requires all 4 indicators to be green.

The widget shows that at the time of receiving this data the transmission was from *DRM_radio_kuwait*, the time (in UTC) of the transmission was 11:53.

The two numbers top left give an indication of the detected frequency offset (the frequency itself is not shown, it was 150110 KHz), the overall ofset is obtained by adding these two numbers.

The bottom line tells that the *spectrum* type was 3, i.e. just a 10 Khz wide spectrum, the *mode*, telling how the data is organized, is mode *B*, the data is encoded as QAM64, the audio encoding is plain AAC and the baudrate of the audio output is 24000 baud.

The space between the bottom line on the picture and the bottom of the picture may contain a text message, a message encoded within the audio stream.

8 The FT8 plugin

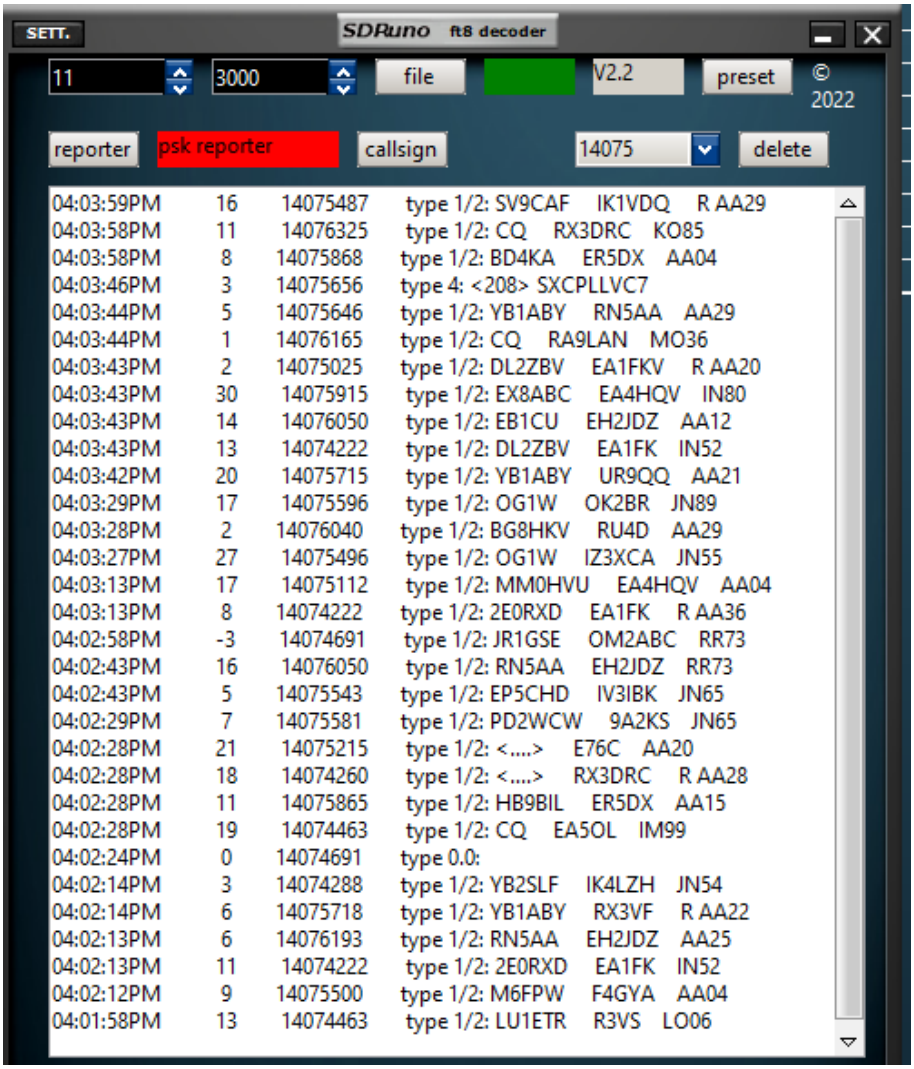
8.1 Introduction

FT8 seems to be a quite popular mode, messages are very short, less than 15 seconds and highly structured. In the 20 meter band - my favourite - is the spectrum around frequency 14074 Khz usually overcrowded with the short FT8 messages.

FT8 decoding is essentially based on a - more or less - "intelligent" brute force approach. In a first step an attempt is made to preselect the potential messages using detection of a Costas array that is used for locating the message. In the second step for each of these potentials an LDPC based error detection/recovery mechanism is applied, in a third step the resulting message is subjected to a CRC check. There is no garantee that all messages in such a message stream are detected and decoded.

The code in the plugin uses parts of the code of the FT4FT8 decoding software of Karlis Goba. Especially the LDPC decoder is a copy of his code, and the copyright to the parts taken or derived from his code are gratefully acknowledged.

8.2 The plugin



The picture shows the output, apparently from a transmission around 4.02PM. The messages have a (more or less) standard FT8 format, preceded by 3 columns

the time of decoding the message;

a strength indicator (derived from the strength of the costas array);

the frequency of the message (the accuracy of the frequency obviously depends on the accuracy of the reading from the SDRuno platform, and is expressed in steps of 3 Hz.

The two row on top of the widget show some controls.

the spinbox, here set at value 11, sets the number of iterations to be applied in the LDPC decoding. A higher value *may* lead to better decoding results, although values too high will lead to false positives.

the spinbox with value 3000 sets the searchwidth in Hz, the range arund the currently selected frequency where the software will search for FT8 messages;

the button labeled *file*, when touched will instruct the software to show a file selection menu for storing the results (in plain ASCII), touching the button when a file is open will close the file. The label to the right, here colored green, will change its color to red when a file is open;

The label wwith text *V2.2* shows the version number of the software;

the button labeled *preset* will add the currently selected frequency to the list of preset frequencies.

The second row

the button *reporter*, when touched will activate a handler to transmit the received data to the so-called psk reporter. If successfull the label to the right will color red. Sending data to the reporter will only work is a *callsign* is filled in, the button with that name - when touched - shows a small widget where such a callsign can be entered.

The combobox showing *14075* tells the currently selected frequency. Elements in this combobox are *preset frequencies*. As mentioned earlier, frequencies can be added to the list of presets. (Obviously, the list will be maintained between invocations of the plugin).

the button labeled *delete* will delete the last element of the preset list.

9 The acars plugins

9.1 Introduction

Acars stands for *Aircraft Communications Addressing and Reporting System*, a digital datalink for transmitting short messages between aircraft and ground stations. The plugins decode Acars transmissions in the VHF band (around 130 Mhz).

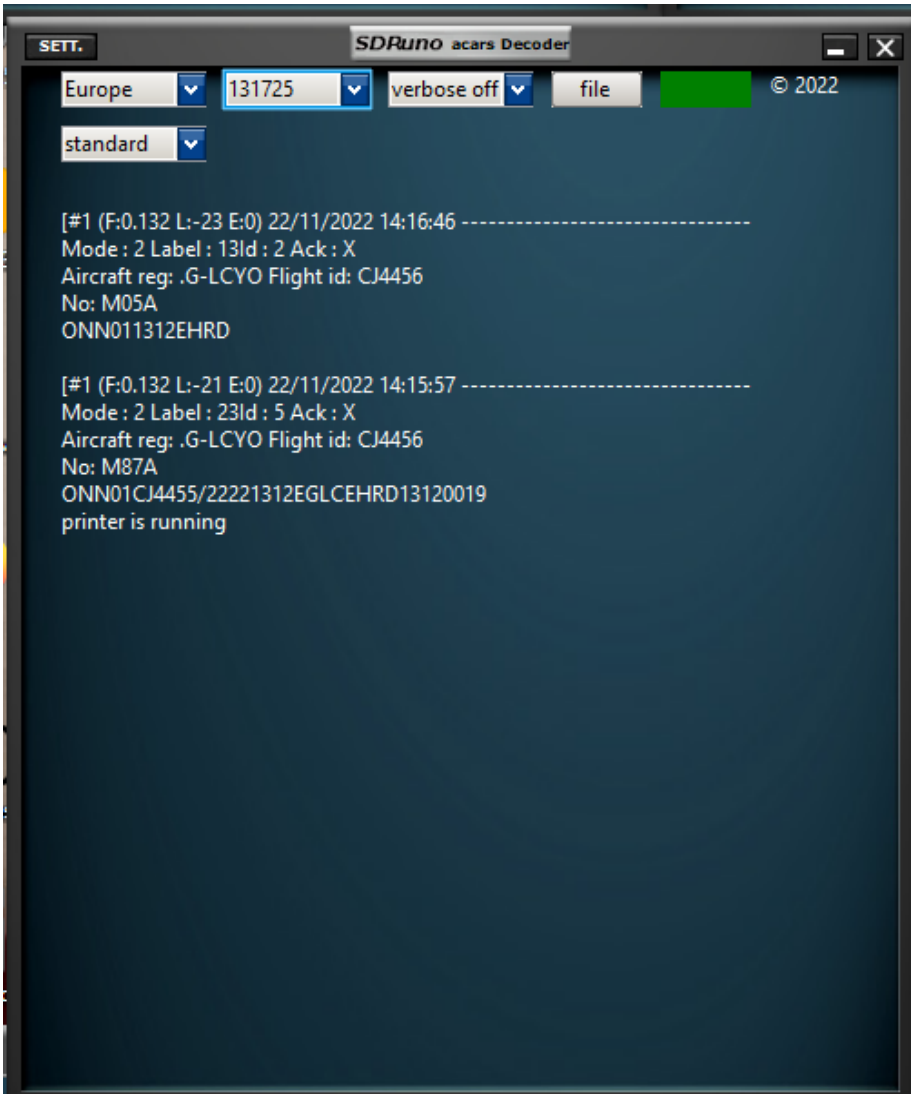
It should be emphasized that the SDRuno plugin for acars decoding uses parts - rewritten though - from the implementation of the Msk decoding and the printer functions from the original acars decoder that was written by Thierry Leconte and his copyrights are acknowledged.

Transmission frequencies for Acars differ from continent to continent, Uusually there is a main frequency and some alternative frequencies per continent (or part of it).

There are two versions of the acars plugin, the first one is for decoding data on a single frequency. The second one uses the fact that a frequency range of 2 MHz can be covered by the SDRplay devices and is able to decode data from different frequencies.

Both plugins have selectors for selecting from common acars frequencies.

9.2 The acars plugin



Controls for the regular acars plugin are:

the *continent* selector. Frequencies differ per continent, so the continent selector can be set, selecting a continent will ensure that the frequencies, listed in the the combobox next to the continent selector, are the ones valid for the selected continent.

the *frequency* selector. This combobox shows the frequencies common for the selected continent. Of course tuning to a different frequency using the SDRuno platform is always possible;

the *verbose* selector. Setting this selector will create some more details in the output;

the *file* selector. Touching this selector will instruct the software to show a file selection menu. When a file is selected, all output will also be written to the file. Touching the button when the file is open will close the file. If a file is selected, the label will color red;

the *standard* selector gives a choice to different forms of output. Standard is displays in the picture, a one line output per message or a monitoring message.

9.3 The acars-M plugin

The difference between the regular acars plugin and the acars-M plugin is that the acars-M plugin handles messages from more than a single frequency at the same time. The picture shows that *Europe* is selected, and that the plugin listens to 3 frequencies, 131725, 131525 and 131825 (as a matter of fact, the picture is old, and for Europe one

chooses either Europe_1 or Europe_2, Europe_1 with 5 frequencies within a 2 MHz range, Europe_2 with 2 frequencies).

So rather than selecting a continent first and a frequency for transmissions within that continent next, one chooses here a label, and each label represents a continent and a few frequencies used on that continent. For Europe one may choose between Europe-1 and Europe-2, for the US between USA_1 and USA_2, furtermore for Australia and Japan.

10 The EXPERIMENTAL apt137 plugin

10.1 Introduction

As known, there are hundreds of satelites flying over our head, some with somewhat undefined purpose, quite a number watching us or our environment. Three satelites, flying polar circles, transmit pictures, photographs of the earth, on a frequency that is known and with a format that is known. These satelites, the NOAA-15, NOAA-18 and NOAA-19 transmit data in the 137 Mhz band and - with some effort - the pictures (photographs) they send downwards can be received.

The photographs are transmitted in an analog form, basically a number of values, one value per color grey, organized in lines, lines with a defined structure, 2 lines per minute.

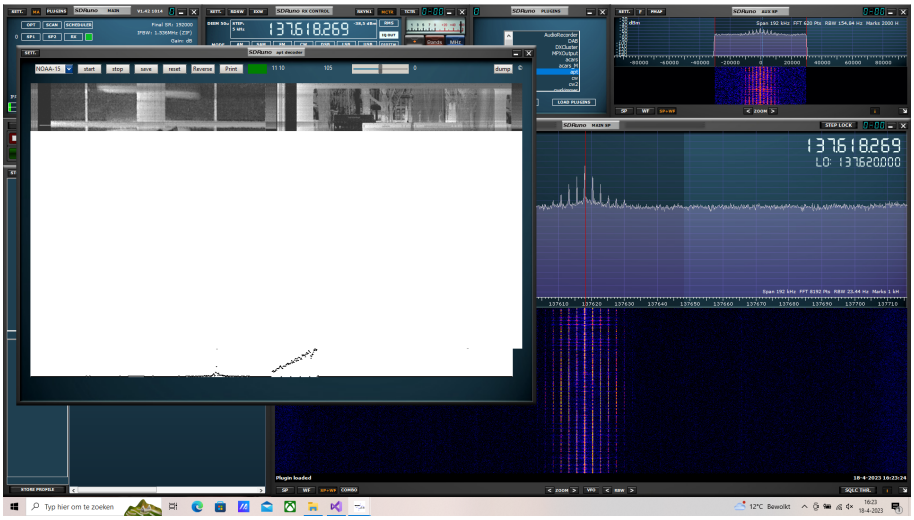
Modulation of the values is first as AM signal on a 2400 Hz carrier, and then FM modulated and transmitted on a frequency specific to the satelite.

Of course there are issues. Satelites fly fast, and their appearance over the horizon is usually limited to at most about 15 minutes. Furthermore, transmission power is very, very low, so one needs a decent antenna for the reception of the signal. Since I am a person with two left hands, I only made a very simple antenna, the one described in

"http://rfelektronik.se/manuals/Datasheets/DIY%20137MHz%20WX-sat%20V-dipole%20antenna.pdf

Although I get a signal, it is pretty weak and the quality of the pictures is not very high. Rather than sitting from time to time in the rain outside aiming my antenna to an assumed azimuth, I developed - based on a decoder made by "Gokberk Yaltirakli" and available under a GNU GPL and using an Adalm Pluto - a simple "transmitter" for fake APT 137 pictures to test the decoder. In that sense the decoder is experimental, I got some data from a satelite, but most of the testing was done with my own generated transmissions. Of course, I'm interested to learn about experiences.

10.2 The plugin



A line in an APT picture contains 2080 pixels, since the screen on my laptop has a width of around 2000 pixels, I have the picture reduced on the screen. In "core" the picture has its full size, and saving it (as a bitmap) preserves the full size.

The controls om the top of the widget are:

the *satelite* selector, when selecting a satelite (either NOAA-15, NOAA-18, or NOAA-19), the plugin will ask the SDRuno platform to switch over to the frequency specific to that satelite;

the *start* button, touching it will start the actual attemtps to decode;

the *stop* button, touching it will stop de actual decoding attempts, some other operation will only function with decoding swtiched off;

the *save* button, when touched will instruct the software to present a file selection menu. After selecting a file, the current picture is saved, as said in the full format as bitmap; Touching the button is a void operation if decoding is "on";

the *reset* button, when touched will - as the name suggests - reset a number of variables;

the *reverse* button, when touched will - as the name suggests - reverse the picture. As can be imagined, sometimes the satelite passes the region from north to south, other times from south to north, in which case the picture will be upside doen. If decoding is off, the picture will be reversed;

the *print* button, when touched, will reprint the picture on the screen. This is useful when changing the "grey setting", with the slider.

the *grey slider* alters the mapping from incoming values to grey values (a little);

the *dump* button will , when touched, instruct the software to present a file selector, with which a ".wav" file can be selected. If a file is selected, the incoming data, i.e. a samplestream, will be saved after the FM decoding and after converting it to an 11025 Ss single channel stream. The reason is that I found a few decoders on the internet that could decode input after the FM decoding and as single channel streams with lower rates.

The green label on the top line indicates that the software seems to be synchronized, the two numbers to the right tell how many samples from the start of the current buffer the first pixel of the line is found. Since that is computed in two different ways, there are two numbers that should not differ too much.

The number to the right of these 2 numbers, here 105, is the number of the line being written.