qdmr - User Manual

A universal codeplug programming tool for Linux and MacOS X.

Hannes Matuschek, OV Y07, DARC <dm3mat@darc.de>

qdmr - User Manual: A universal codeplug programming tool for Linux and MacOS X.

by Hannes Matuschek

This document covers qdmr version 0.12.0.

About This Document

This document contains a brief manual for the qdmr graphical user interface (GUI) application to program DMR radios. The aim of this application is to provide a simple to use, platform and vendor independent and well documented codeplug programming software (CPS) for popular DMR radios.

This document is available in several formats:

- HTML [https://dm3mat.darc.de/qdmr/manual/],
- PDF [https://dm3mat.darc.de/qdmr/manual.pdf] and
- EPUB 3 [https://dm3mat.darc.de/qdmr/manual.epub].

Table of Contents

Foreword

Before you start programming your DMR radio, get familiar [https://www.dmrfordummies.com/] with this digital mode. I have written a brief introduction into DMR too. You will find it in Chapter 1, *Introduction* (also available in german [https://dm3mat.darc.de/qdmr/manual/script_de.pdf]). If you are already familiar with DMR, skip this chapter and head directly to Chapter 2, *The Graphical User Interface*, where I describe how qdmr is used. If you are a fan of command line tools, have a look at Chapter 7, *The **dmrconf** command line tool*.

Digital mobile radio (DMR) was not invented for the use in amateur radio. It was rather designed to be a radio standard for commercial applications in large companies (e.g., airports etc.). Therefore many features of this standard are of no use for ham radio or are even illegal (e.g., encryption). This complexity of the standard makes the programming of the radios cumbersome.

Moreover, the resulting configuration (codeplug) is highly device-dependent. These codeplugs cannot be shared between different devices let alone between different vendors. For commercial applications, this is not a big problem, as a company will likely buy identical radios at once from one company. Thus codeplugs can be shared between all radios.

For ham radio applications, this incompatibility is a real issue. Since assembling a decent codeplug for one region is hard enough, doing the same work all over again for different models of different vendors is not manageable.

Finally, the typical code-plug programming software (CPS), particularly those for cheap Chinese DMR radios, is by no means user-friendly and seldom documented completely. Many options are named cryptic and it is not possible to identify which options are necessary for basic DMR operation. Moreover, the CPSs provided by the vendors usually only run under Windows.

The aim of the qdmr project is to overcome these shortcomings of typical CPSs. It has a reduced feature set only supporting those options necessary of amateur radio usage. It tries to be user-friendly by finding repeaters nearby and importing their input and output frequencies. Moreover, it stores the final codeplug not in a device-specific binary format but in a human-readable text format that is device independent and can therefore be shared across multiple device and even across vendors.

Finally, I try to keep the application well documented. This manual is part of this effort. It is a guide on how a codeplug is set up using qdmr.

Chapter 1. Introduction

Abstract

The chapter tries to provide an introduction to DMR (digital mobile radio) targeted at the unexperienced operator and anyone interested in this topic. I try to hide details until it gets absolutely necessary to explain them. The majority of DMR introductions I've found, are more or less extensive glossaries (if you are interested in that, see Glossary). They are hard to comprehend, unless one has at least some experience with DMR.

The perceived complexity of DMR comes from its origin as a radio standard for commercial applications at large events and companies (i.e., trunked radio [https://en.wikipedia.org/wiki/Trunked_radio_system]). Therefore, I will first describe an example how DMR is used commercially before I describe how it is used in amateur radio. I hope that this approach will make some of the weird terms and concepts of DMR clearer.

Basics: Repeater operations

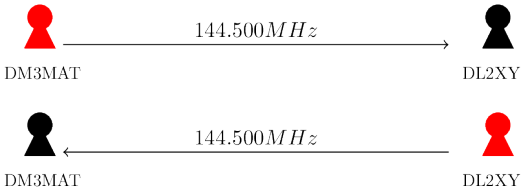
This section briefly describes the common amateur radio FM-repeater operation. The majority of all licensed operators will be familiar with this topic.

If you are not yet licensed and interested into amateur radio, consider reading this section. Otherwise, skip right to the section called “DMR Introduction & Origin”

The majority of connections between HAM operators are made in the so-called simplex mode. That is, the two operators transmit and receive alternately on the same frequency[1] and the connection is direct. This works very well on HF where world-wide direct connections can be made.

[1] This is actually called *semi-duplex*, however the term “simplex” stuck. The term simplex actually refers to the situation, where there is only one transmitter and possibly many receivers (e.g., broadcast).

Example 1.1. Typical FM simplex operation

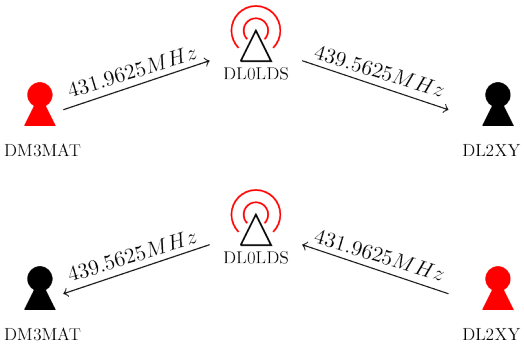


For this example, DM3MAT transmits directly to DL2XYZ on the frequency 144.500 MHz. The latter then also answers directly on the same frequency.

On higher frequencies, however, radio waves behave more like light and it gets increasingly difficult[2] to bridge significant distances beyond the horizon. This fact limits the operating range of simple hand-held radios. To still cover a large area, repeaters can be used.

Repeaters are autonomous amateur radio stations that are usually located on a mountain, hill or high tower. This allows them to easily cover a large area. They receive signals from HAM operators and retransmit them at the same time. To do that, they cannot send and receive on the same frequency (otherwise they would interfere with themselves). Therefore, repeaters operate in the so-called duplex mode. That is, the repeater receives on one frequency (the so-called input frequency) and transmits the received signal on another frequency (the output frequency) simultaneously.

Example 1.2. Simple FM repeater operation



For this example, DM3MAT sends on the input frequency 431.9625 MHz to the repeater DB0LDS. The repeater transmits the received signal on its output frequency 439.5625 MHz. On that frequency, DL2XYZ receives the original call.

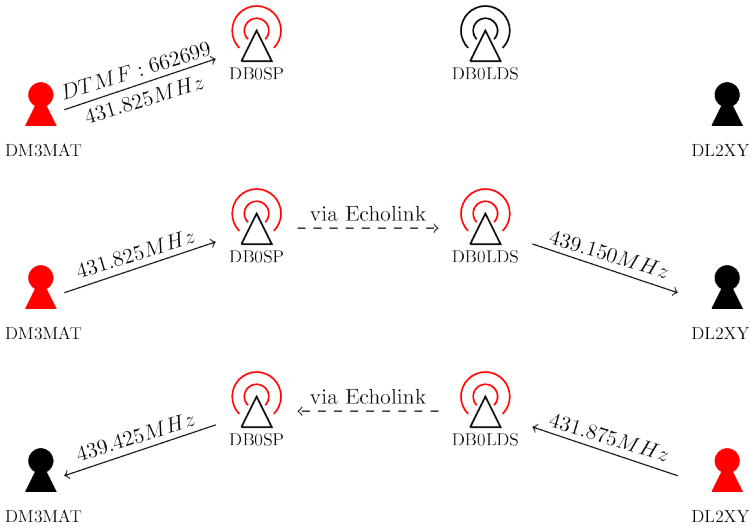
The Example 1.2, “Simple FM repeater operation” shows a common repeater operation on UHF. Here, the operator DM3MAT transmits its call to DL2XYZ not directly but on the input frequency of the repeater DL0LDS (431.9625 MHz). The repeater receives the call and transmits it simultaneously on its output frequency (439.5625 MHz). This signal is then received by DL2XYZ. Consequently, the call has reached its destination, although DM3MAT and DL2XYZ may not be able to communicate directly. The reply of DL2XYZ to DM3MAT follows the same path. Here DL2XYZ transmits on the repeater input frequency, and DM3MAT receives that call on the repeater output frequency. This way the two operators can communicate with each other even if they are not able to reach each other directly.

Echolink

However, there are situations, where two operators are far away and they cannot reach the same repeater. For these cases, it is possible to connect two repeaters via the EchoLink [http://www.e-cholink.org/] network.

[2] Also on VHF and UHF, larger distances can be bridged using an elevated location and larger antennas.

Example 1.3. Repeater operation with Echolink



DM3MAT connects repeaters DB0SP (near Berlin) and DB0LEI (near Leipzig) via EchoLink. Now, they are able to communicate with each other.

This network allows to link FM repeaters via internet™ or to connect directly to a remote repeater via internet. Many FM repeaters are connected to the EchoLink network, allowing for world-wide communication with simple hand-held radios.

Frequently, it is also possible to control a repeater over-the-air and to connect that repeater to some other repeater via EchoLink. Usually, this can be done by sending the EchoLink number of the destination repeater using DTMF to the local repeater. This is shown in Example 1.3, “Repeater operation with Echolink” above. There, DM3MAT sends the EchoLink number 662699 of the repeater DB0LEI near Leipzig to his local repeater DB0SP near Berlin. Then the local repeater (DB0SP) will link with the destination repeater (DB0LEI) via the EchoLink network. For some limited time, both repeaters will act like one logical repeater. That is, everything that is received by one repeater will also be transmitted by the other. This way, the two operators DM3MAT and DL2XYZ can communicate although they cannot reach a common repeater.

Note

Once two repeaters are linked via EchoLink, they behave like a single repeater.

All over the world, there are FM repeater that are part of the EchoLink network. Therefore, it is possible to communicate world-wide at any time using simple hand-held radios that are as cheap as 40€ or even less.

DMR Introduction & Origin

DMR (digital mobile radio) is a digital radio standard to transmit speech and data. That is, the speech is not directly modulated on the carrier but is first digitalized and compressed using a lossy compression codec (VOCODER). The compressed speech is then transmitted as data packets. The latter allows to attach meta-information to the data packet like source and destination of the packet.

DMR was designed to be the digital replacement for analog trunking networks in commercial applications. A classic example for such a commercial application of DMR would be a civil air port. With this, I do not mean the air-traffic radio but all the communication of the ground staff in and around the actual air port buildings.

At such an imaginary air port, there is a huge staff with a wide variety of tasks. For example (without any claim of completeness):

- The cleaning crew,
- technicians,
- security staff,
- apron staff for refueling, luggage and catering,
- the fire brigade and
- the headquarters.

All these people carry a radio and should be able to

- Directly call the headquarters. All staff should be able to call the headquarters directly.
- Direct communication between members of the same group, without interfering with other groups. For example, the cleaning staff should be able to communicate with each other, without interfering with the fire brigade.
- Each person should be able to call a complete group. For example, the headquarters may call the entire fire brigade or one member of the security staff may call the entire security for help.

An air port, however, is a rather large area. Consequently, not all staff members are able to reach all others. Therefore, some repeaters must be installed to cover the entire air port including all interiors.

If you compare this scenario with the classic FM repeater networks (see the section called “Basics: Repeater operations”), it gets clear that is hard to implement these concepts using analog FM repeaters. Especially, if several repeaters are connected though a network. In this case, a single call on one repeater may block the entire network[3].

Certainly, it would be much better if only those repeaters get activated that are actually required for the communication between two parties. Then, the remaining repeaters are still available for the rest of the staff. This routing, however, should happen automatically. An operator may not know, which repeater to use to reach a particular person.

DMR was designed to implements such complex communication networks without requiring from every participant to have detailed knowledge about the structure of the network. That is, the knowledge about where every repeater is installed and which participants are reachable on which repeater.

Note

DMR is more similar to a phone network that to classic FM repeater networks.

Speaking of phone networks: Each participant and thus his/her radio is uniquely identified by a number. The DMR-ID. This is a number between 1 and 16777215. And like for any other phone network, each participant may call any other directly using this number. This call is called *Private Call*.

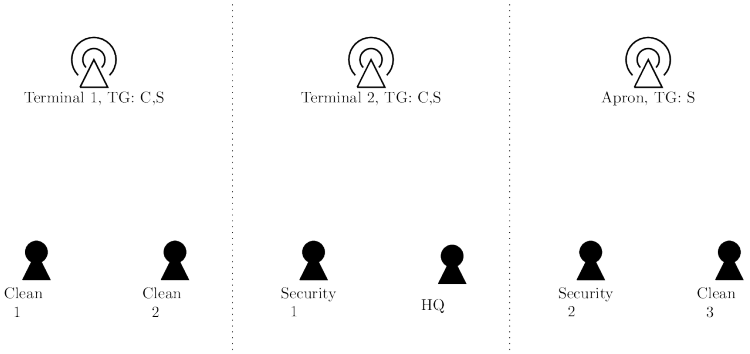
And there are groups. Each of these so-called *Talk Group*s is also assigned a unique number. A talk group can be used to group all staff with a specific task (e.g., the security, fire brigade, etc.). It is then possible to reach all members of this group at once, by performing a *Group Call* to that talk group. The network, however, does not know which participant is member of which group. Consequently, the radio of the participant needs to know which group calls to accept and which to ignore.

Note

This point is important to remember: The DMR network does not know which participant is member of which group. The radio needs to be configured to react on specific group calls.

[3] There are means to implement this concept on analog repeater networks using tone-signaling techniques (e.g., DTMF, five-tone etc.).

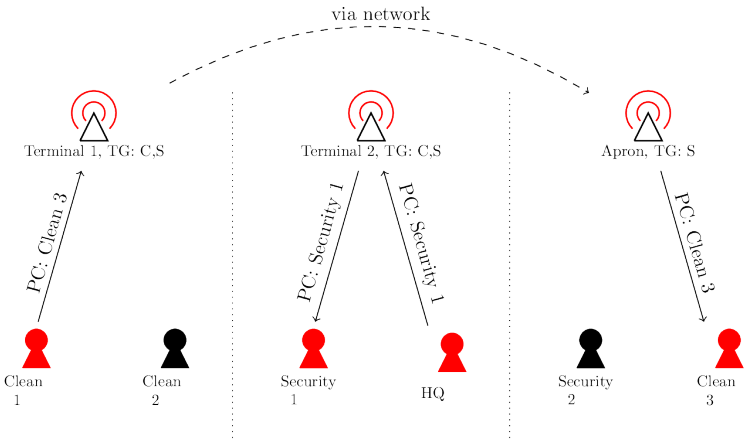
Example 1.4. Simplified air-port network



There are 3 cleaning staff, two security and one headquarters. To cover the entire area, three repeaters are required. One in terminal 1, one in Terminal 2 and one on the apron.

Example 1.4, “Simplified air-port network” is a simplified air port network (in reality, it is much larger and way more complex). Consider the situation, where a cleaning stuff 1 & 3 want to communicate. At the same time, the headquarters want to talk to security 1. In an simple analog network, the call between cleaning 1 & 3 would block the entire network and therefore the call between the headquarters and security 1.

Example 1.5. Simultaneous calls



Two Simultaneous private calls in the example network between cleaning 1 & 3 as well as between headquarters and security 1.

Private calls in DMR networks only use those repeaters, that are actually required to establish the communication. This is shown in Example 1.5, “Simultaneous calls”: Cleaning 1 starts a private call to cleaning 3. As the DMR network knows that cleaning 3 was last active on the apron repeater, this call gets routed only though repeaters terminal 1 and apron. The repeater in terminal 2, however, is not affected. Consequently, this repeater remains available for the call between the headquarters and security 1.

Note

The network only knows at which repeater each participant was last active. The network will therefore try to establish a connection though that repeater to the participant.

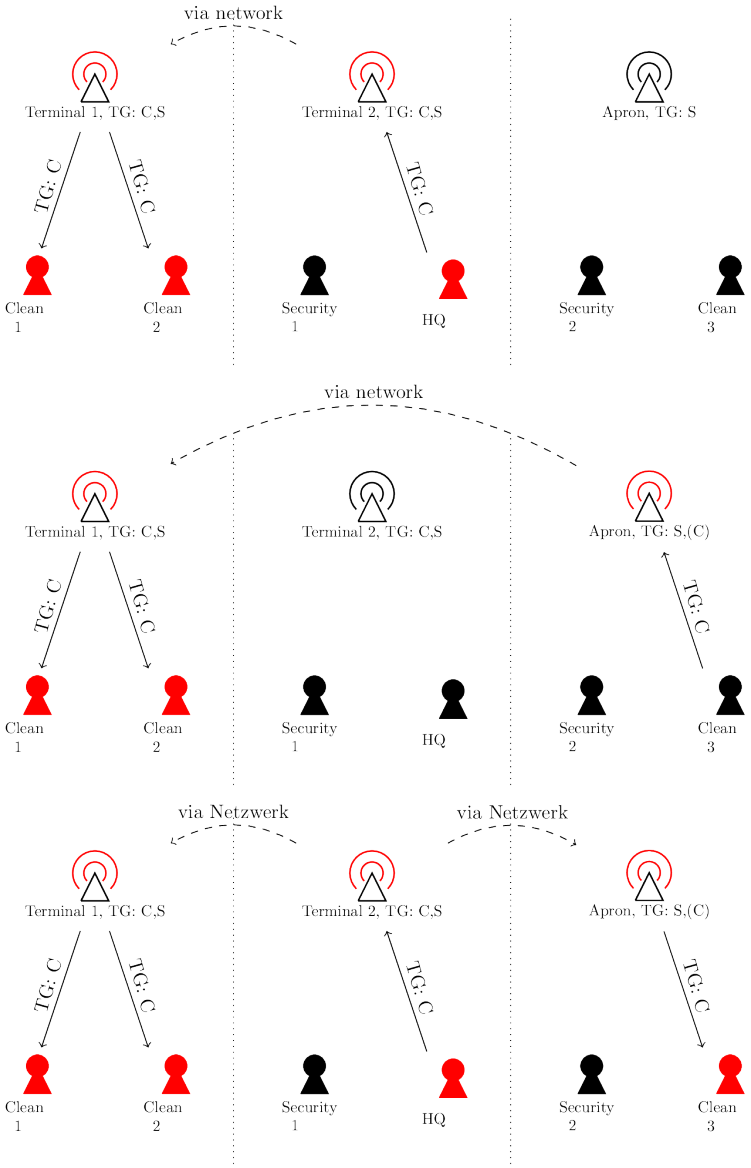
During the call between cleaning 1 & 3, the repeaters in terminal 1 and apron are blocked. This means, that the headquarters may not be able to reach cleaning 2 and security 2 immediately. This sounds worse than it actually is. In contrast to classic phone networks, a direct call is considered interrupted,

once the calling participant releases the PTT button. To this end, the headquarters may use the pauses between calls to reach the other participants.

In the next Example 1.6, “Temporary subscription of talk groups.”, the headquarters want to reach all cleaning staff. Therefore, they start a group call to the talk group “cleaning” (*C* for “cleaning” and *S* for “security”). With this call, it can reach cleaning 1 & 2 immediately. However, cleaning 3 does not receive that call.

This is due to the fact, that the DMR network does not know which participants are members of which groups. As the cleaning crew is usually not on the apron, the apron repeater has not subscribed the talk group “cleaning”. Therefore, it does not forward group calls to that talk group.

Example 1.6. Temporary subscription of talk groups.



To remain reachable for group calls, cleaning 3 needs to temporarily subscribe the apron repeater to the “cleaning” talk group. This can be done by starting a group call to that talk group on the apron repeater. Then the repeater will temporarily subscribe to that talk group for a limited amount of time (usually between 10-30min). During that time, the repeater will forward group calls to that talk group and cleaning 3 remains reachable via that repeater.

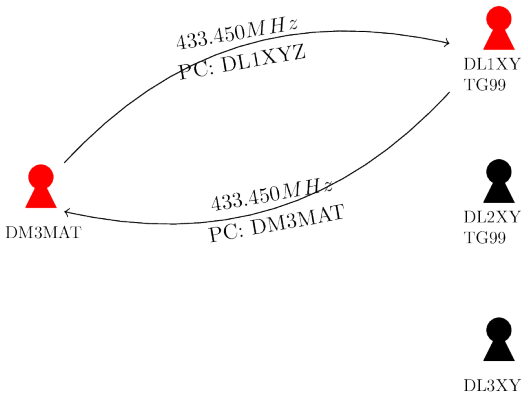
This temporary subscription will be renewed every time a participant starts a group call to that talk group on this repeater.

With these examples, the most basic terms of DMR (DMR-ID, talk group, private and group calls as well as talk group subscriptions where introduced and explained on an example network. The following sections will concern the use DMR in ham radio.

DMR Simplex Operation

The most simple form of a DMR QSO[4] is the simplex QSO. That is a direct connection between two two DMR radios. Like for the DMR repeater operations, this could be a private, group or so-called *All Call*.

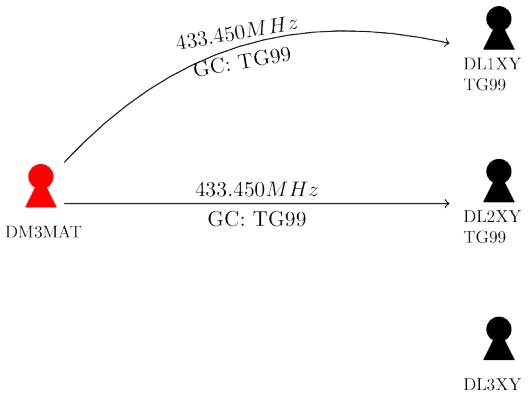
Example 1.7. Simplex private call.



In Example 1.7, “Simplex private call.” a simple simplex private call from DM3MAT to DL1XYZ is shown as well its reply. Both operators transmit and receive on the same frequency (here the DMR calling channel at 433.450 MHz). Although other operators are in the area (DL2XYZ & DL3XYZ) which receive the signal, their radios remain silent. This is because this is a private call to a specific operator and only the radio of that operator will receive the call. All other radios will ignore the call. The channel, however, remains occupied during that call.

At that point it is worth mentioning, that if DL1XYZ answers directly to the initial call by pressing the PTT, he will answer with a private call to DM3MAT. He does not need to search for number of DM3MAT in his address book. The direct answering to calls is only possible for several seconds after the end of the initial call. After that period (called *Hang Time*) a press on the PTT will start a call to the default contact (see the section called “Creating channels”) associated with the simplex channel.

Example 1.8. Simplex group call



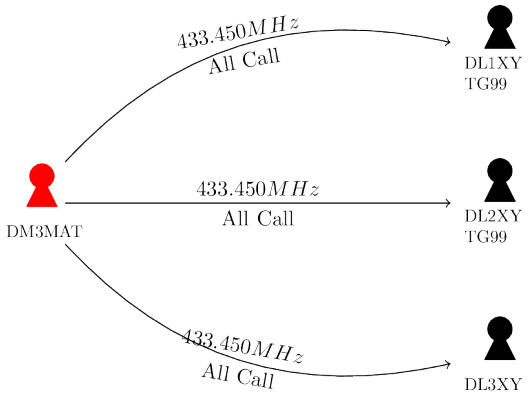
It is not only possible to call single operators in simplex mode. Also groups can be called using group calls. A common talk group for the simplex mode has the number 99, (TG99, for “talk group 99”).

[4]QSO is a code for call or connection between two amateur radio stations.

These group calls are then received by all radios that are configured accordingly. Like for the repeater operation, also in simplex mode, the radio needs to know which groups the operator belongs to and therefore, which talk groups to receive on which channels. This is done using so-called *Group Lists*, which are discussed later.

In Example 1.8, “Simplex group call” such a simplex group call is shown. There DM3MAT calls the talk group TG99. As DL1XYZ as well as DL2XYZ configured their radios to receive that call on simplex channels, they do so. DL3XZY did not, so he does not receive the call. DL1XYZ & DL2XYZ can now respond to that call by pressing on the PTT within the hang time irrespective of their default contact for the channel.

Example 1.9. Simplex all call



To be sure that a simplex call gets received by all operators in the area, a so-called *All Call* should be used. This is a special call type to the reserved number 16777215, that gets received by all radios irrespective of their configuration. For the Example 1.9, “Simplex all call”, the all call by DM3MAT gets received by all operators including DL3XYZ. By directly answering within the hang time, all participants are able to respond to that call with an all call as well.

Note

In short: A DMR channel consists of a transmit and receive frequency (identical on simplex channels), a default contact that gets called whenever the PTT button is pressed and a list of group calls the radio will receive on that channel.

Table 1.1. DMR simplex frequencies

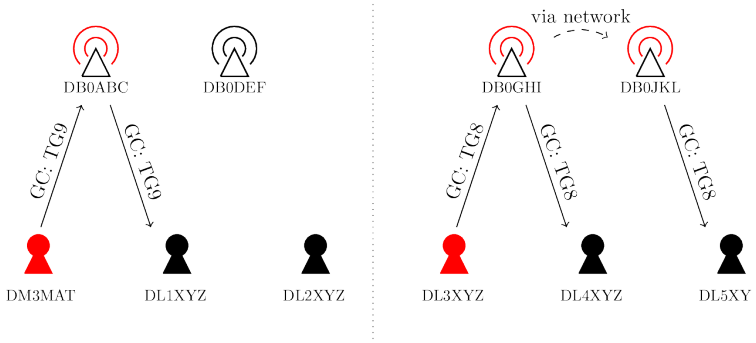
Name	Frequency	Name	Frequency
S0 (call)	433.4500 MHz	S4	433.6500 MHz
S1	433.6125 MHz	S5	433.6625 MHz
S2	433.6250 MHz	S6	433.6750 MHz
S3	433.6375 MHz	S7	433.6875 MHz

Table 1.1, “DMR simplex frequencies” lists the common simplex channel frequencies. The channel S0 is the calling channel. Especially in densely populated areas, you should switch to another channel for the actual QSO and use S0 only for the initial call.

Local Repeater Operation

One central objective of DMR is to be repeater transparent. That is, it does not matter which repeater you use. You will always be able to reach the same groups and be always reachable through the same means (private or group call). This concept is violated by the talk groups 8 & 9. The are the regional and local talk groups.

Example 1.10. Two regions with two repeaters each.



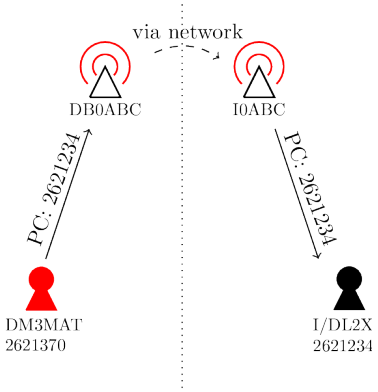
The talk group 9 (TG9) is the so-called local talk group. Group calls to that talk group are not forwarded though the network and only retransmitted locally. Usually “locally” means only the repeater. Sometimes, however, these calls are also forwarded to other repeaters nearby. This case is shown in Example 1.10, “Two regions with two repeaters each.” on the left side. Here DM3MAT sends a group call to TG9 via the repeater DB0ABC. This call is not forwarded to any other repeater and thus is only received in the local area around the repeater. DL1XYZ is in that local area and may receive that call if he configured his radio to receive calls from the TG9.

The talk group 8 (TG8) is the so-called regional talk group. A call to that talk group is usually forwarded to all repeaters within a specific region. Which repeaters are part of a “region”, is a decision of the repeater administrators. So it is hard to predict to which repeaters a regional call gets forwarded. In Example 1.10, “Two regions with two repeaters each.” on the right side, DL3XYZ sends a group call to TG8 to the repeater DB0GHI. This call gets forwarded to all repeaters in the “region”. In this case, also to the repeater DB0JKL within the same region. Therefore, all participants within that region are able to receive the group call if they configured their radios accordingly. In this example, not only DL4XYZ received the call but also DL5XYZ who is not close to the repeater DB0GHI and would have missed a group call to the local talk group.

Private calls

Private calls allow to call other participants directly without interfering with other calls (except for using the repeater). In the introduction into DMR above, the private call over several repeaters has been described. I consider this aspect of DMR particularly interesting. With the exception of TG 8 & 9, private and group calls are transparent with respect to the repeaters used. It simply does not matter which repeaters are actually used to establish a connection. Therefore, I do not need ot know where the other participants are located.

Example 1.11. Direct calls between countries.



Consider the typical vacation situation: An OM at his holiday location may want to participate in his local afternoon net. He can do that by simply starting a group call to the nets talk group over the repeater at his holiday location. Now the local holiday repeater has subscribed to the local talk group

at home and the OM can participate as usual in the net. The other participants in the net may not even recognize that the OM is at his holiday location.

In a similar fashion, private calls can be started and received at the holiday location. The DMR network knows the repeater a participant was last active on. By briefly pressing the PTT at the holiday repeater, the OM is registered and can now receive private calls at his holiday location. The friends at home may not even know where the OMs current location is nor which repeater to use to reach him. The network takes care of that.

Example 1.11, “Direct calls between countries.” shows such a private call between countries. DM3MAT starts a private call to DL2XYZ using his local repeater DB0ABC. DL2XYZ, however, is at his holiday location in Italy. As he had registered himself at that repeater, the private call gets forwarded to the holiday repeater I0ABC in Italy, where DL2XYZ can receive it. DM3MAT does not need to know the location of the callee nor which repeaters are near to him. This automatic routing of calls (group and private calls) is a major advantage over the analog FM repeater network and EchoLink. For the latter, the ID of the destination repeater needs to be known.

Data services

As DMR is a digital mode that transports digitalized speech, it is possible to transfer other data too. Consequently, there are some other digital services provided with DMR. First, there are text messages similar to the one provided by mobile phones. It is also possible to forward the own GPS position to the *APRS* network.

Textmessages (SMS)

With this service, you can transfer short text messages to other participants (like a private call) or to a talk group (like a group call). The latter is rather uncommon and should be avoided. In principle a text messages works like a private call. If the destination is reachable, the text message will be routed to it.

There are also service numbers (free of charge). If messages are sent to them, certain information can be retrieved to forwarded to other networks (e.g. to the *DAPNET*). In Germany (and other countries) there are:

1. 262993 -- GPS and weather
   - Send `help` and you will receive a list of commands.
   - Send `wx` and you will receive the weather at the location of the repeater you used.
   - Send `wx CITY` and you will receive the weather at the specified city.
   - Send `gps` and the last GPS position is returned that you have sent to the DMR network.
   - With `gps CALL` you can retrieve the last position sent by the specified call.
   - Send `rssi` and you will receive a signal report from the repeater.
2. 262994 -- Repeater information & pager messages
   - Send `rpt` to receive a list of static an dynamic subscribed talk groups at the repeater.
   - Send `CALL MESSAGE` to send the given message to the given destination call in the DAPNET.

Position reporting to the APRS network

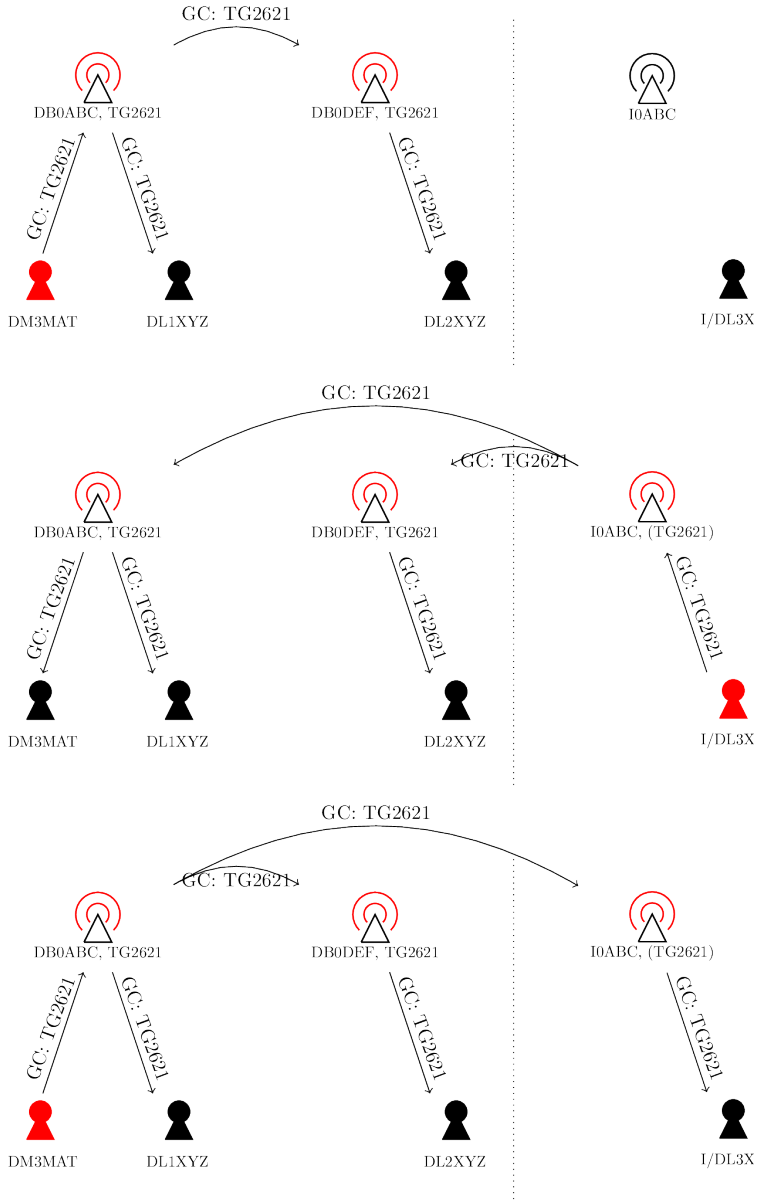
As mentioned in the previous section, it is possible to report the own position via the DMR network to the APRS network. This position can then be tracked at for example aprs.fi [https://aprs.fi]. To do that, a radio with a built-in GPS receiver is required. But not even these devices are expensive anymore. Simple DMR hand-helds with built-in GPS receivers are available for about 120€.

Beside the text message services, it is also possible to transmit the position to the DMR network using a special service number 262999. How the radio is configured to do that, depends heavily on the device being used. qdmr helps with the configuration by providing common means for these settings across all supported radios.

Talk Group Operation

As mentioned in the section called “Private calls”, DMR aims at being repeater transparent. That is, it does not matter which repeater is used for the operation. This is also true for talk groups. In this section, I'll continue with the example of an OM at his holiday location, who wants to participate in his afternoon net. For example, the afternoon net is happening on the talk group 2621 (Berlin/ Brandenbug, BB). This talk group is usually statically subscribed on all repeaters in the states Berlin and Brandenbug. That means that the local afternoon net can be performed in this talk group without any additional action in this area. (see Example 1.12, “Talk group operation between countries” top image).

Example 1.12. Talk group operation between countries



For the OM at his holiday location, this it not true. An Italian repeater will certainly not subscribe the BB talk group statically. Therefore, the OM at the holiday location will not hear his afternoon net. He knows, however, when this net starts. So he can perform a group call to the that TG shortly before the net starts (see Example 1.12, “Talk group operation between countries” second image). This will subscribe the BB talk group temporarily at the holiday repeater (I0ABC). Now, the OM can hear and participate in the afternoon net. The subscription will be renewed whenever he starts a group call to that TG.

Once the OM subscribed the TG to the holiday repeater, he can participate normally at his afternoon net. All other participants will not even notice that he is not in the area and is participating from an italien repeater.

Table 1.2. Some talk groups

Name	Talk group number
Global	91
Europe	92
Germany	262
Mecklenburg-Vorpommern & Sachsen-Anhalt	2620
Berlin & Brandenburg	2621
Hamburg & Schleswig-Holstein	2622
Niedersachsen & Bremen	2623
Nordrhein-Westfalen	2624
Rheinland-Pfalz & Saarland	2625
Hessen	2626
Baden-Württemberg	2627
Bayern	2628
Sachsen & Thüringen	2629

Cluster

In contrast to the regional talk group TG8, all other talk groups are reachable from everywhere in the DMR network. This means, that the OM at his holiday location can easily participate in his afternoon net from everywhere as described above.

If the net, however, is happening in the regional TG8, the OM at his holiday location cannot participate. This talk group is only reachable from within the region. If the OM starts a call to TG8, he would only reach the region in italy and not the region at home.

For this reason, some regional *clusters* of repeaters are linked to a so-called “Cluster”. These clusters provide “normal” talk groups for the repeater within a region. These clusters are then also reachable from the outside. A list of regional clusters and their associated talk group numbers can be obtained under bm262.de/cluster/ [http://bm262.de/cluster/].

Roaming

Usually all repeater within a region will subscribe the same talk groups. This allows to operate in these talk groups in a repeater transparent way. Therefore, it does not matter which repeater is being used within the region, the same talk group remains reachable. In the region Berlin & Brandenburg, this it the TG2621.

It therefore makes sense to enter all repeaters into one list that have the same talk groups subscribed. If the radio now would automatically select a reachable repeater, one could drive around in the region

and would stay connected to these talk groups irrespective of the own position in the region. This feature exists in many radios and is called *Roaming*. Many of the slightly more expensive devices support this feature (e.g., AnyTone). The cheapest ones usually do not[5].

To use this feature, all channels with a certain talk group should be added to a list. The so-called *Roaming Zone*. This could actually be done automatically, but the programming software for these devices is usually not very user friendly.

If the signal strength of the currently selected repeater falls below a certain threshold (usually -105dBm), the radio will start to search the roaming zone for a repeater which is stronger than this threshold. This only happens if the radio is in standby. That is, if neither something is received nor transmitted.

If the radio finds a stronger repeater in the roaming zone, it automatically changes to that repeater. The new repeater does not necessarily needs to be the strongest in the zone. It only needs to be stronger than the threshold. If no stronger repeater is found, the radio remains on the currently selected one.

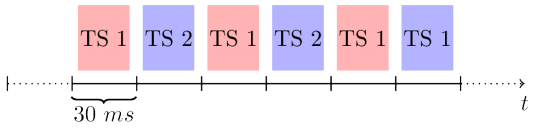
This roaming can also be set to “manual”. That is, the roaming search will only start if the signal strength is lower than the threshold and the PTT is pressed or the search is started from the menu.

Technical background

In the previous sections, I tried to explain the basic concepts of the DMR operation. That is, the repeater independent private and group calls. This section concerns the more technical details of the DMR mode. In particular the *Time Slot* and *Color Code*.

Time Slots

As mentioned before, DMR is a digital mode. The speech signal is first digitized and compressed with a lossy compression codec. The latter is also called *VOCODER*. Modern codecs are very efficient and allow to transfer two independent speech signals within a single 12.5kHz wide channel. This is exploited in DMR using a technique called *TDMA*.



TDMA means “time-division media access” and describes that two independent calls can happen simultaneously on one physical channel. To achieve this, each call is assigned a *Time Slot* (time slot 1 & 2) and both are transmitting and receiving only within their assigned time slot. These time slots are very short. For DMR they are only 30ms long. This short time, however, is sufficient to transfer audio for at least 60ms. Therefore, DMR allows for two independent and simultaneous calls on a single physical channel.

When time slot 1 or 2 “happens”, is determined by the repeater. The repeater defines the beat. This also implies that time slots are irrelevant for the simplex operation. Thus you can ignore the time slot settings when programming simplex channels. (see the section called “Codeplug Assembly”).

Note

What happens on each time slot, is a convention defied by the repeater. A general suggestion is that regional communication happens on the time slot 2 while trans-regional communication should happen on time slot 1.

[5]This is somewhat weird, as this feature is a pure firmware feature and does not need any additional hardware.

Color Codes

*Color Code*s are a technical tool to avoid conflicts between repeaters operating on the same frequency. This usually happens in commercial applications of DMR. One company usually gets only a small number of frequencies assigned. To cover the entire campus, much more repeaters are needed than frequencies are available. Consequently, overlapping repeater ranges on the same frequency will occur and two repeaters may receive the call of a participant. Then, the color code allows the repeater to detect whether a call was intended for it. Only if the color code of a call matches the color code of the repeater, the repeater will react on that call.

This issue usually does not arise in amateur radio applications. We just have enough channels. Hence the color code is usually set to 1. In very densely populated areas, however, overlapping repeater ranges may still occur and different color codes might be used there.

Note

To use a repeater, you not only need to know the input and output frequencies but also the color code of the repeater.

Codeplug Assembly

After the basic concepts and technical details of the DMR mode has been discussed, it is time to consider the actual configuration of the DMR radios. This usually not done via the keypad of the radio but with the help of a separate software. The so-called *CPS* or codeplug programming software.

Before we can start, we need like any other participant in the DMR network a unique number, the DMR ID.

Note

You can get your personal and unique DMR ID from radioid.net [https://www.radioid.net/]. There you need to verify that you are a licensed ham operator.

You will receive your personal DMR ID usually within 24h per Mail. Once you've got your ID, you can start.

As this script is intended for the beginners, it is very likely that you do not own a top-shelf Motorola device but rather one of the cheap devices of the common manufacturers.

Warning

If you do not own a DMR device yet but consider to by one, you should explicitly check whether it supports DMR Tier I and II[6]. Ignore any marketing BS and check the technical description of the product for Tier I & II. If it is not mentioned there, simply skip that product. This is particularly true for the Baofeng MD-5R but not the RD-5R.

The manufacturer of the device of your choice will provide the CPS for download you need to program your radio. Usually you will also find there firmware updates for your device. The manufacturers usually provide a separate CPS version for every device and even firmware revision. So please check whether you've got the correct CPS version. The configuration of the device differs from device to device and even more from manufacturer to manufacturer. The basic setup, however, remains the same.

When you start the CPS for the first time, you will likely note two things. First, that the user experience stems from the last millennium (about Windows 3.11). And second, that there are a tone of obscure

[6] As usual, DMR is not a single standard but a family of standards. Tier I describes the simplex operation while Tier II considers the repeater operation and time-slots. You will therefore need a device that also implements Tier II to be able to work with repeaters.

and badly translated options for your device. These options are usually named cryptic and are not documented.

The configuration of your device usually happens in five to six steps:

1. General settings,
2. creating contacts,
3. assembling group lists,
4. creating channels,
5. assembling zones and
6. optionally assembling scan lists.

Within the following sections, I want to guide you through these steps.

General radio-wide settings.

The single most important options within the general settings is your DMR ID and your call sign. These options are usually located under the label Radio Settings or General Settings[7]. Your DMR ID is entered in the field name Radio ID. Many radios support to enter several DMR IDs. This feature is usually not used in ham radio. In fact you will only always need a single DMR ID even with several radios.

Your call can be entered in the Radio Name field.

Creating Contacts

Once you have made these basic settings, you may create some contacts in your contact list. This list should contain all talk groups you are interested in, some private contacts to OM you know as well as some “service numbers” for the “echo”-service, SMS service etc. A sample is shown in Table 1.3, “Example contacts for germany”.

Table 1.3. Example contacts for germany

Name	Typ	Nummer	Name	Typ	Nummer
Local	group call	9	Ham/SlHo	group call	2622
Regional	group call	8	NiSa/Bre	group call	2623
TG99	group call	99	NRW	group call	2624
All call	all call	16777215	RhPf/Saar	group call	2625
World wide	group call	91	Hessen	group call	2626
Europe	group call	92	BaWü	group call	2627
D-A-CH	group call	920	Bay	group call	2628
Germany	group call	262	Sa/Th	group call	629
Austria	group call	232	Echo Test	private call	262997
Switzerland	group call	228	SMS Serv.	private call	262993
EMCOM EU	group call	9112	DAPNET	private call	262994
EMCOM WW	group call	9911	APRS GW	private call	262999
MeVo/SaAn	group call	2620	DM3MAT	private call	2621370
Ber/Bra	group call	2621	...	...	...

[7]The actual name may vary from manufacturer to manufacturer.

Of cause there are much more talk groups. There are also talk groups for specific topics which are not necessarily targeted at a specific region. A rather complete list can be found in the Brandmeister Wiki [https://wiki.brandmeister.network/index.php/TalkGroups].

Assemble group lists

The next step is to assemble so-called *Group List*s. These are simple lists of talk groups that you want to receive on a particular channel. As mentioned in the introduction, the network does not know which talk groups you are interested in. This must be programmed into the radio. Group lists do exactly that: The specify which talk groups you want to receive. All others are ignored.

You should at least create two group lists. One for the simplex operation, one for regional communication and optionally one for the trans-regional communication. You should also create one for each region you frequently visit.

The simplex group list is theoretically not necessary as simplex calls should always use the so-called *All Call*. Frequently, however, also the talkgroups TG99, TG9 and TG8 are used in simplex operation. Hence a group list with these talk groups is needed for simplex operation.

Your trans-regional talk group should include the talk groups for the entire world TG91, your continent (e.g., Europe TG92), your country (e.g., Germany TG262) and also the emergency talk group (e.g., 9112 in Europe).

Finally the talk group for the local/regional communication should contain the local TG9, regional TG8 and the talk group for your region (e.g., TG 2621 for my region Berlin/Brandenbug). As I am also frequently in saxony, I also created a group list for that region. My group list settings are shown in Table 1.4, “Example group lists”.

Table 1.4. Example group lists

Name	Group calls
Simplex	Local, Regional, TG99
WW/EU/DL	World wide, Europe, D-A-CH, Germany, EMCOM EU
Ber/Bra	Local, Regional, Ber/Bra
Sa/Th	Local, Regional, Sa/Th

Creating channels

Before we start assembling any channels, I should mention that DMR radios are also able to transmit and receive analog FM. Thus, you can also use them for classic FM simplex and repeater operation. In this section, I describe the configuration of digital DMR channels usually called “digital channels”. The configuration of analog FM channels is not described. To create a DMR channel, you have to select digital for the channel type, for FM channels analog.

When you already have some experience with the analog FM repeater operation, the configuration of DMR channels may appear quiet weird. For analog FM repeaters, you usually configure exactly one channel. For DMR repeaters you will configure at least two (one for each time slot), but usually many more. To cut a long story short, let me explain it with a concrete example.

Creating Simplex Channels

Table 1.5. Example simplex channel configuration

Name	RX Freq.	TX Freq.	TS	CC	TX Contact	Grp.List
DMR S0	433.4500 MHz	433.4500 MHz	1	1	all call	simplex

Name	RX Freq.	TX Freq.	TS	CC	TX Contact	Grp.List
DMR S1	433.6125 MHz	433.6125 MHz	1	1	all call	simplex
...	...	...	...	...	...	...

In Table 1.5, “Example simplex channel configuration” an example for a simplex channel configuration is shown. Of cause, you should extend it to all 8 simplex channels. The first column simply specifies the name of the channel.

The second and third columns specify the transmit and receive frequencies for these channels. For simplex channels, RX and TX frequencies are the same.

In simplex operation, there is no repeater. That is, no instance that dictates a beat. To this end the choice of the time slot (TS) is irrelevant and usually TS1 is chosen.

The color code, however, matters. Repeater as well as your radio will ignore calls with a mismatching color code. For simplex channels, the color code 1 has been established.

The sixth column specifies the default transmit contact. For simplex channels, the so-called *All Call* should be chosen, to ensure that really everyone can receive the call irrespective of the receivers group list settings. The default transmit contact specifies the contact (private, group or all call) that is called whenever the PTT is pressed. As mentioned earlier, there is an exception to that rule. Whenever you directly answer a call within in the so-called *Hang Time*, you will answer with the same call you received.

The last column specifies the to so-called *Group List*. This list specifies which talk groups are received on that channel. As mentioned earlier, no entry should be needed here, as the all-call should be used for the transmit contact on simplex channels. Unfortunately, it is not uncommon to find several talk groups being used as transmit contacts on simplex channels like TG9, TG8, TG99. For these cases, a group list “simplex” was created earlier.

Within your CPS, you will find many more options for channels. The majority can be left untouched. At the end of this section, I will describe some of these settings briefly. Many of these settings are quite uncommon in amateur radio or even straight illegal.

The *Admit Criterion* specifies under which conditions your radio is allowed to transmit. For simplex channels, the option channel free should be chosen. This configures the radio to only transmit if the channel is currently free.

Creating repeater channels

Creating repeater channels is slightly more complex than creating simplex channels, as we need to create several channels per repeater. Before you can create any channels, you need to know which DMR repeaters are near to you. A good overview provides the repater book [https://repeaterbook.com]. There you can also filter for DMR repeaters and you get all information you need to configure the DMR repater channels. That is input and output frequencies and the color code of the repeater.

Table 1.6. Example channels for a single repeater DB0LDS

Name	RX Freq. (output)	TX Freq. (input)	TS	CC	TX Contact	Grp.List
DB0LDS TS1	439.5625 MHz	431.9625 MHz	1	1	-	WW/EU/DL
DB0LDS DL TS1	439.5625 MHz	431.9625 MHz	1	1	Germany	WW/EU/DL
DB0LDS Sa/Th TS1	439.5625 MHz	431.9625 MHz	1	1	Sa/Th	Sa/Th
DB0LDS TG9 TS2	439.5625 MHz	431.9625 MHz	2	1	TG9	Ber/Bra
DB0LDS TG8 TS2	439.5625 MHz	431.9625 MHz	2	1	TG8	Ber/Bra
DB0LDS BB TS2	439.5625 MHz	431.9625 MHz	2	1	Ber/Bra	Ber/Bra

I think, it is the best to explain the creation of repeater channels using a concrete example for a repeater near to me shown in Table 1.6, “Example channels for a single repeater DB0LDS”. This repeater has the call DB0LDS and has the input frequency 431.9625 MHz and output frequency 439.5625 MHz. According to repeater book, this repeater expects the color code 1. These are the elementary information you need to set for all channels using this repeater. Many CPSs allow to copy or clone channels. This way you only need to enter this basic information once.

At the end of the section called “Time Slots”, I mentioned that trans-regional communication is happening on time slot 1 while regional communication is happening on time slot 2. This is visible in this example. The repeater is located in the region Berlin/Brandenbug (Ber/Bra), consequently all channels with within-region talk-groups have the time slot 2, all others have the time slot 1.

The first channel “DB0LDS TS1” is a generic channel for the time slot 1. There is no default transmit contact defined for this channel. This channel can be used to perform arbitrary direct and group calls by selecting a contact or talk group from the contact list. This means, that a call cannot be started by simply pressing PTT on that channel. First, a contact must be selected that should get called.

The second channel “DB0LDA DL TS1” is almost identical to the first except for the default transmit contact. Here “Germany” (TG262) is selected. This means, if this channel is selected and the PTT is pressed, the talk group 262 is called. By configuring a separate channel for this talk group allows to start a call to it without having to search for it in the contact list. This also allows to temporarily subscribe this talk group on a repeater easily, by simply pressing PTT briefly.

Note

Irrespective of the default transmit contact, you can always answer to a call within the hang time.

The third channel “DB0LDS Sa/Th TS1” is also similar to the first two. Here the default transmit contact is the talk group for Saxony/Thuringia (TG2629) to be able to subscribe that talk group at my local repeater and call it easily. Please not that for this channel the time slot 1 is used. The repeater is located in Brandenburg and therefore any communication with Saxony is inter-regional and should happen on the time slot 1. The group list contains only the talk group for Saxony/Thuringia and thus other inter-regional talk groups are not received on that channel.

Channels four, five and six are for repeater-local (TG9), regional (TG8) and the talk group Berlin/Brandenburg (TG2621) calls. As this is all regional communication, it happens on the time slot 2. Also they all have the group list “Ber/Bra” set (see Table 1.4, “Example group lists”). Therefore, all regional talk groups (TG8, TG9, TG2621) are received on that channel. As the default transmit contact, the corresponding talk group is set. If the channel “DB0LDS TG9 TS2” is selected and the PTT is pressed, a call to TG9 is repeated only by the repeater DB0LDS. If the channel “DB0LDS BB TS2” is selected and the PTT is pressed, a call to TG2621 is repeater by almost all repeaters in the region Berlin/Brandenbug. Therefore, chose a talk group that is sufficient for you intended communication.

Note

On any channel, you can start an arbitrary call (group, privat, all) by either selecting the contact from the contact list of even simply entering the DMR number into the keypad of the radio. This is independent from the default contact on the current channel. In the end, the default transmit contact is a convenience feature. With the default contact, channels for frequently used contacts can be created.

The so-called *Admit Criterion* should be set to Color Code for DMR repeater channel. This means, that the radio will only transmit if the channel is free and the color code of the repeater matches the color code of the channel.

Other channel options

The user interface of the manufacturer CPS where you configure the channels, is usually very extensiv. There is a huge amount of options that control the behavior of the channel. The majority of these options are not used in ham radio applications. Some of these, however, I want to describe here briefly.

The *Admit Criterion* was mentioned before. It controls under which conditions the radios can transmit. There are usually three options. Always does exactly what it says: it allows to transmit always. This option should be chosen for analog FM repeater channels. Channel free means that the radio will only transmit if the current channel is free. This option should be choses for simplex channels. When Color code is selected, the radio will only transmit if the channel is free and the color code of the repeater matches the color code of the channel. This option should be chosen for DMR repeater channels.

The *TOT* setting or “transmit timeout” secifies the maximum duration of continuous transmission. After that period of continuous transmission, the radio will stop the transmission automatically. The feature is used in commercial applications to avoid the blocking of a channel or talk group by a participant. This option has little sense in amateur radio and can be set to infinity.

The *Emergency System* is a method to signal an alarm or an emergency situation. Also this feature is not used in amateur radio.

The option *Privacy Group* or *Encryption Key* refers to a built-in method of encrypting the traffic. This is actually forbidden in amateur radio.

The flags “Emergency Alarm Confirmed”, “Private Call Confirmed” and “Data Call Confirmed” specify how the radio starts these calls. The radio will first establish a call to the destination and will signal once the call is “confirmed”. Once the confirmation is received, the actual call starts. These options are not used in amateur radio and should be disabled as they may interfere with the normal operation.

The option *Talkaround* allows to operate simplex on a repeater channel. That is the radio transmits and receives on the repeater input frequency. This allows to bypass the repeater and to communicate directly with other participants on the same repeater channel. Also this option makes little sense in amateur radio.

When the *RX Only* flag is enabled, the radio cannot transmit on that channel. This may be useful for out-of-band monitoring channels where you are not allowed to transmit.

The *VOX* feature is actually used in ham radio. It stands for “voice operated switch” and allows to start a call using the voice without the need to press PTT. Some radios allow to enable this option on a per-channel bases others only radio-wide.

The “Power” option allows to specify the transmit power level. This can usually be set in predefined steps like Low, Middle, High. Some radios may also allow a fine grained setting of the power level.

The *Scan List* specifies a list of channel that are scanned if a scan is started on that channel. This feature might be used as an alternative to a missing roaming feature (see the section called “Roaming”).

Assembling zones

Once you have assembled all channels of interest, you may notice that the list is quite long. Hence all DMR radios organize the channels in so-called *Zone*s. Zones are simple lists of channels that group them into relevant sets, usually based on the location. You may therefore collect all channels for “Home”, “Work” and “Holidays” into one zone each. How you organize your zones is up to you.

You may also organize these channels by talk groups. This way you may implement some kind of a manual roaming. Once you left the range of a repeater you may search for another one in the same zone. This way you stay connected to a particular talk group. In contrast to the automatic roaming, you have to select the repeater by hand.

Note

Channels that are not assigned to any zone are usually not selectable by the radio. It is, however, perfectly fine to assign a channel to several zones.

Assembling scan lists

*Scan List*s are simple lists of channels. When the scan is started on a particular channel, the channels scan list is used. The radio will then step though that list and may stop on a channel that shows activity. It is then possible to answer the received call. This function allows for observing several channels. Additionally, it is usually possible to specify one or more “priority channels” for a scan list. This channel is then visited more frequently and thus monitored more intensively.

DMR Networks

Within the previous sections, I tried to outline the concepts and some technical details of a DMR network and how a codeplug might be assembled. These concepts, however, apply only to the so-called *Brandmeister* network. This is the network in the background that routes your private and group calls, connects repeaters etc. In Germany, this is the dominant network. But also world wide, it connects the majority of repeaters (about 5000). However, there are also other networks. There is the *DMR-MARC* network and the *DMR+* network. Which network you are likely to encounter, depends on your location. In countries like France, Spain, BeNeLux, Poland, Czech Republic and Slovakia, almost all repeaters are connected to the Brandmeister network, while in Denmark the DMR+ network dominates. In the USA and Austria, DMR-MARC repeaters aren't rare. All these networks do not differ on the technical level. That is, you DMR-ID is valid in all of these networks and you can use any DMR Tier II radio.

The concepts however, in particular how group calls are performed, depends heavily on the network. This means, that you need to configure the repeater channels for a DMR+ repeater in a different way compared to a Brandmeister repeater.

Reflectors (DMR+)

*Reflector*s play an important role in the DMR+ network. They represent a talk group within the DMR + network.

The major difference between a reflector and a talk group is, that they cannot be simply called using a group call. They are subscribed to a local repeater by a private call to the reflector. Then all repeaters subscribed to that reflector behave like a single repeater. You will then participate on that reflector by performing a group call to TG9, the local talk group. Your call will then be sent to the reflector as well as to all other reflectors currently subscribed on the repeater and consequently to all repeaters also subscribed to that reflector.

This has the advantage of a much simpler codeplug assembly, as only two channels are configured for each repeater. One for each time slot. The default transmit contact will always be the local talk group TG9. To subscribe a reflector, a private call is started to the reflector from the contact list. This implies that the contact list should contain all reflectors you are interested in. This concept is also much closer to the semi-analog concepts of EchoLink. However, advanced features like roaming are not possible this way. Also the repeater transparency gets lost. Instead of simply starting a group call to the destination talk group, the local repeater needs to be “configured”. Once that configuration is done, the communication will happen on the local talk group TG9, even if the communication is not local anymore.

Chapter 2. The Graphical User Interface

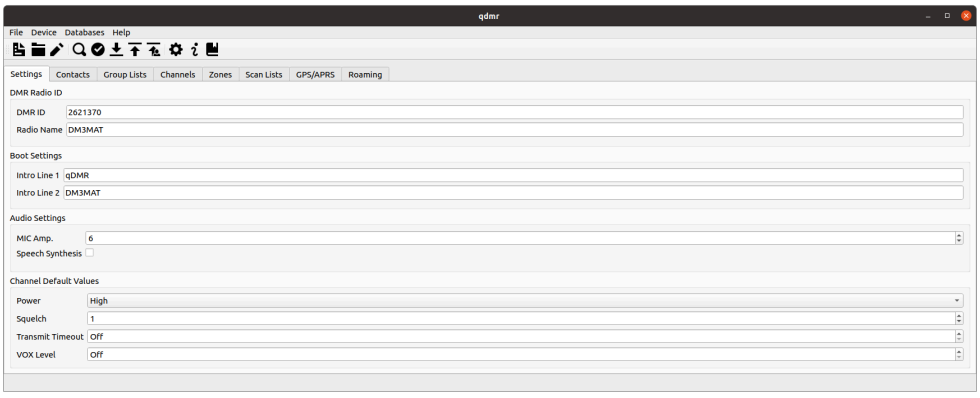
This chapter describes the graphical user interface (GUI) in some detail.

Note

qdmr aims at being a device and manufacturer independent CPS. To this end, the GUI reflects a device independent code plug. This means, that not all of the shown features are present in all radios. For example, not all supported devices implement APRS or roaming. These settings are then ignored when generating the device specific binary code plug.

General configuration

The figure below shows the General Configuration tab of qdmr. This tab is divided into 4 sections: DMR Radio ID, Boot Settings, Audio Settings and Channel Default Values.



General settings tab.

The general configuration of your radio is the simplest step: You only need to enter your DMR ID and your radio name. The latter is usually just your call sign. If you do not have a DMR ID yet, you can request one at ham-digital.org [https://register.ham-digital.org/].

Note

You will always need only *one* DMR ID, even if you have several radios. The DMR network is able to handle multiple endpoints with the same ID. Never request more than one ID, they are a limited resource.

In the rare case, where you actually need more than one DMR ID, for example if you the same radio from HAM as well as commercial applications, you may add your additional DMR IDs using the Radio IDs tab. This tab is usually hidden and can be accessed by enabling Show commercial features in the application settings (see the section called “Application Settings Dialog” below).

Within the Boot Settings, the Intro Line 1 & 2 specify the text that some radios show on startup. You may enter any text here. Some radios show an image during boot. For those radios, these settings have no effect.

Within the Audio Settings, the MIC amp. option specifies the microphone amplification (if supported by the radio). This must be a value between 1 and 10, where 1 is the smallest amplification and 10 the loudest.

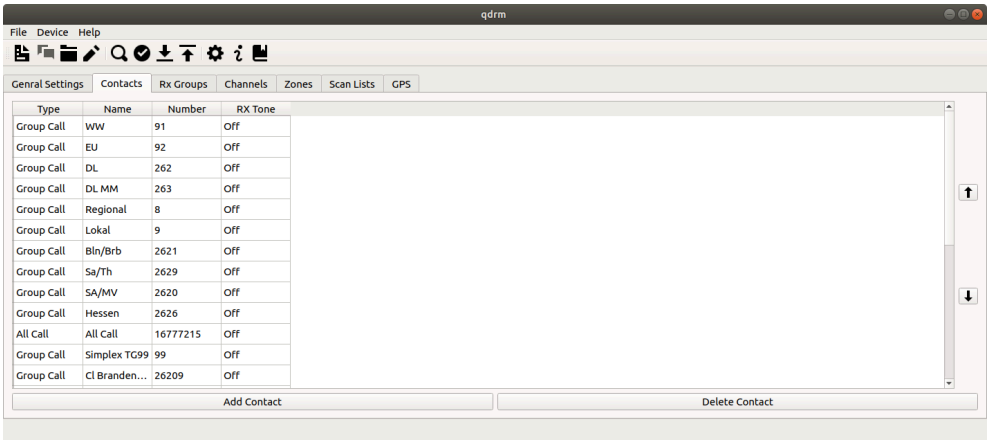
The Speech Synthesis option allows for enabling speech synthesis by the radio. Some radios support speech synthesis to help visually impaired operators to handle the radio and navigate its menus. If this option is checked, the speech-synthesis will be enabled if the radio supports this feature.

The Channel Default Values block allows to specify some default values that can be referenced by channels (see the section called “Creating Channels” below). This serves two use cases. First, it allows to set some channel properties for all channels that reference these values at once. More importantly, however, some radios do not allow to set these options on a per-channel basis. For those radios, these values are used for all channels.

If Show Commercial Features is enabled in the settings dialog (see the section called “Application Settings Dialog”), an additional block labeled Extensions is shown at the right side. This view allows to add, remove and edit device specific radio settings. All device specific settings are handled and displayed in the same way. See the section called “Edit Device Specific Extensions” for some description of this process.

Creating Contacts

The second tab is the Contact List. Here all DMR contacts are defined, irrespective of their type. It is not only possible to define digital DMR contacts (i.e., private, group and all calls) but also DTMF contacts (and in future two-tone, five-tone contacts too). This eases the control of the *EchoLink* feature of repeaters.

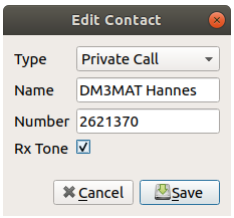


The contact list.

You may add a contact by clicking on the Add Contact button at the bottom. You can also delete a contact by selecting the contact in the list and clicking on the Delete Contact button at the bottom. You may also reorder the contacts by selecting a contact in the list and use the arrow-up and arrow-down buttons on the right to move the contact up and down the list, respectively.

Editing DMR contacts

When you create a code-plug, the contact list should contain all talk groups and reflectors you are interested in, as well as a so-called *All Call* contact to the number *16777215*. Additionally you may add private calls to several operators you know, as well as some *service numbers*.



The new/edit DMR contact dialog.

When you click on the Add Contact button or when you double-click a contact entry in the list, the Edit Contact dialog will appear. The first drop-down box allows to choose the type of the call. The possible options are *Private Call*, *Group Call* and *All Call*. The second entry is the name of the contact. Here any text can be entered. The third entry is the number of the contact. This entry gets disabled when *All Call* is selected as the call-type. Finally, if the last option Rx Tone is enabled, you will hear a ring-tone whenever this contact calls you.

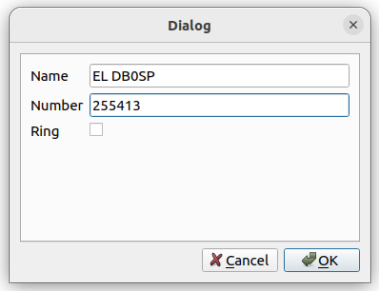
Tip

qdmr tries to download the current list of all registered user DMR IDs. The contact dialog will use this information (once downloaded) to resolve call-signs to DMR IDs. Just start entering the call-sign into the name field and matching call-signs are shown.

The same holds true for Brandmeister *Talkgroup* names. That is, start typing the name of a talk group and a drop-down list of talk group names will appear for auto-completion. Select the talk group and its DMR ID gets set.

If Show device extensions is enabled in the settings dialog (see the section called “Application Settings Dialog”), a tab bar is shown at the top. There you can also access the device specific settings for contacts.

Editing DTMF contacts



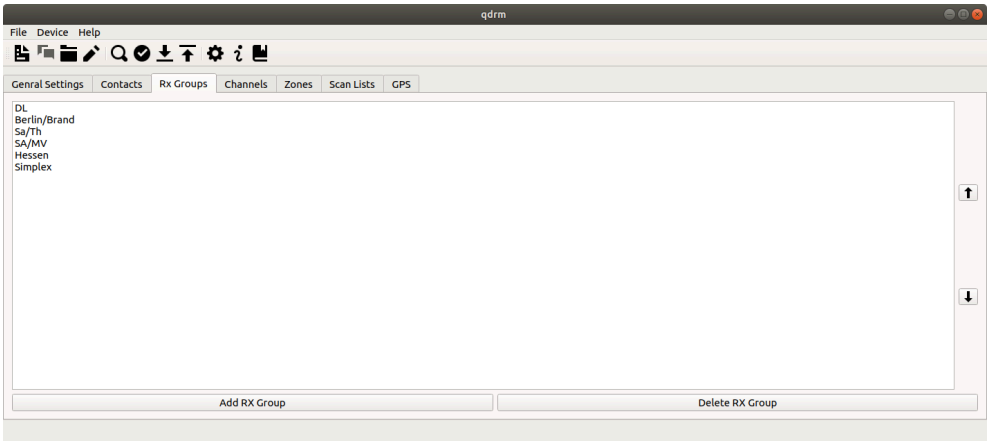
The new/edit DTMF contact dialog.

Beside DMR contacts, the contact list may also contain analog DTMF contacts. You can add a new DTMF contact by clicking on the Add DTMF contact button. This will open the Edit DTMF contact dialog (see above). This is a rather simple dialog.

Name specifies the name of the DTMF contact, while Number specifies the DTMF number. There, any DTMF number can be entered, this includes symbols like *, #, A-D. The Ring option specifies whether the radio should ring if a call by this contact is received.

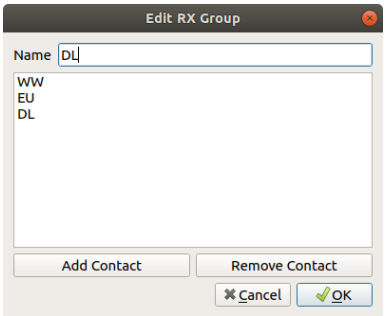
Assembling Group Lists

RX group lists or simply *Group Listss* are just lists of *Group Call*s you may wish to receive on a channel. Of cause, you may want to receive calls to multiple talk groups on one channel, hence you have to create these lists of talk groups beforehand.



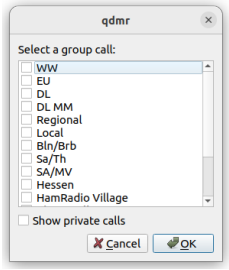
The list of group lists.

The Group Lists tab is just a simple list of all group lists you created. You may add a group list by clicking on the Add RX Group button at the bottom. You can delete a group list by seleting it in the list and clicking on the Delete RX Group button there. You can also edit a RX group list by double-clicking on that group in the list.



The group list dialog.

When creating a new group list or when editing one, the Edit Group List dialog will open. Within this dialog, you can change the name of the group at the top. In the center of the dialog, you will find the list of group calls of this group list. You can add group call to the list by clicking on the Add Contact button on the bottom. You can also remove contacts from the list by selecting the contact and clicking on the Remove Contact button. When you are done editing the group list, click on the Ok button. The Cancel button will discard all changes and closes the dialog.



Selecting the group calls to add.

When adding contacts to the group list, the a dialog appears, that allows for selecting the group calls to add to the group list. Some radios (in particular the OpenGD77 firmware) also allow to add private calls to group lists. To this end, the Show private calls option allows to add private calls too. All other devices do not allow for private calls in group lists. They will simply ignored by qdmr when the codeplug is generated for these devices.

If Show Commercial Features is enabled in the settings dialog (see the section called “Application Settings Dialog”), a tab bar is shown at the top. There you can also access the device specific settings for the group list.

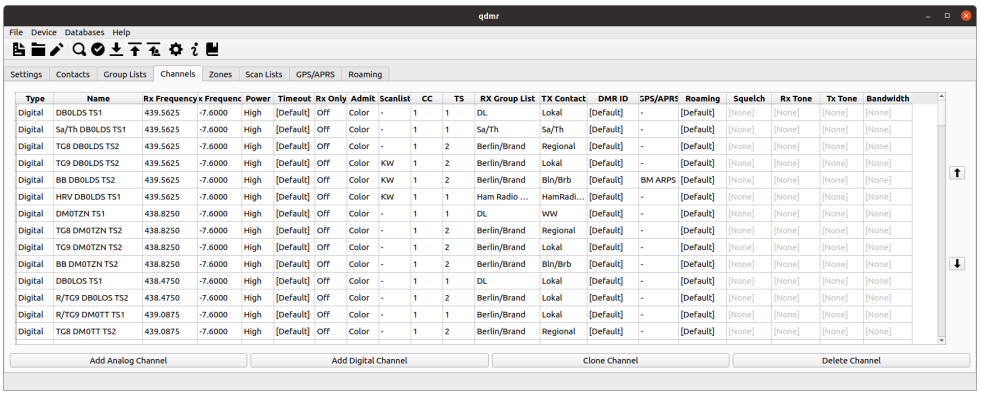
Creating Channels

Creating the list of channels for the DMR radio is the most cumbersome task. Remember, each repeater has two time-slots with possibly multiple talk-groups assigned to each time slot. For the sake of convenience, it is reasonable to define a channel for each talk-group you are interested in on every repeater. Thus, instead of a single channel per FM repeater, you will likely define at least 3-4 channels per DMR repeater.

To ease the burden of creating a lot of channels, qdmr implements some features that should help you in creating these channel. One feature is the automatic retrieval of repeater input and output frequencies from repeaterbook.com [https://repeaterbook.com]. This is a world-wide database of ham-radio repeaters. Both, FM and DMR.

Tip

When you enter your locator into the settings dialog (see the section called “Application Settings Dialog”), qdmr will provide you with a list of nearby repeater and fill in the input and output frequencies. This feature works for both, FM and DMR repeaters.



List of channels.

The Channels tab shows the list of all defined channels, irrespective of whether they are FM or DMR channels. You may add an analog or digital channel by clicking on the Add FM Channel or Add DMR Channel button on the bottom, respectively.

The Clone Channel button allows for cloning of a selected channel. This enables one to create a set of channels that differ only in the default transmit contact but share the same remaining settings.

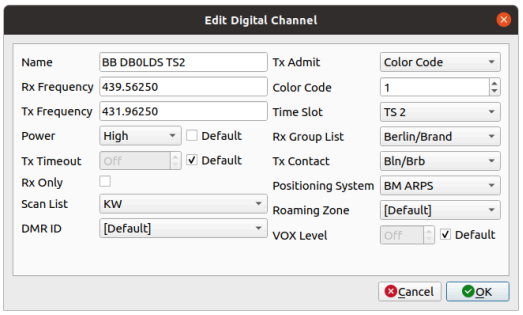
You can also delete a channel, by selecting that channel in the list and clicking on the Delete Channel button at the bottom. You may move a channel up or down the list by selecting that channel and clicking on the arrow-up or -down button to the right, respectively. Finally you can edit a channel by double-clicking it in the list.

Tip

The number of channels usually grow fast and it becomes hard to find channels within the list. To search the list for any channel name or frequency, just hit Ctrl+F to open a search box. This search feature is present in all lists. The channel list, however, is likely the largest.

Edit DMR channels

When you double-click on a DMR channel or click on the Add DMR Channel button, the digital channel editor dialog will be shown. This dialog allows you to edit or create digital channels.



The DMR channel editor.

The dialog is limited to the DMR-channel settings that are relevant for amateur radio. Thus, it is much smaller that the typical dialogs to edit DMR channels in commercial CPSs. However, if you enable the Show device extensions option in the settings dialog (see the section called “Application Settings Dialog”), an additional tab will appear called Extensions. There all implemented device-specific settings are hidden. They can be used to set all options you also find in the manufacturer CPS. See also the section called “Edit Device Specific Extensions”.

The Name, Rx and Tx Frequency fields contain the chosen name of the channel as well as the transmit and receive frequencies. The latter can be set automatically by using the repeater auto-completion feature: Start to enter the call-sign of a repeater. A list of matching nearby repeaters is shown. Select a repeater from this list and the RX/TX frequencies will be set using the information from repeaterbook.com [https://repeaterbook.com]. For simplex-channels RX and TX frequencies must be identical.

Note

The retrieval of the repeater information may time some moments. Hence, the auto-completion list of repeaters near by may not appear immediately. The found repeaters are cached locally for faster retrieval later on.

The Power setting specifies the power used on that channel. For a nearby repeater, you may reduce the power. If you check the Default box, the global power setting will be used. See the section called “General configuration” above.

Tx Timeout (*TOT*) specifies the transmit timeout in seconds. This limits the continuous transmission time to this period. A value of Off disables the timeout. If you check the Default box, the global TOT setting will be used. See the section called “General configuration” above.

VOX Level specifies the sensitivity of the *VOX* for this channel. If OFF is selected, the VOX is disabled for this channel. If you check the Default box, the global VOX setting will be used. See the section called “General configuration” above.

Checking Rx Only will disable transmission on this channel.

Scan List allows to specify the scan list associated with this channel. If a scan is started on this channel, this scan list will be used. Each channel may have a different scan list.

The DMR ID field allows to select the Radio DMR ID for this channel. Some radios allow to program several DMR IDs to be used with one radio. This option makes not sense for HAM-radio usage. However, if you use the same radio for HAM as well as commercial applications, you may need to set your HAM DMR ID for HAM-radio channels and your commercial ID on commercial channels.

The Tx Admit field specifies the *Admit Criterion*, under which you are allowed to transmit on the channel. For DMR repeater channels this should be set to Color Code. This means that you may only transmit if the radio received the correct color code of the repeater before. On simplex channels Channel Free should be chosen and thus the radio will only transmit on the channel if the channel is free.

The *Color Code* specifies the color code of the repeater. For simplex channels, this should be set to 1. For repeater channels, this must match the color code of the repeater. If the auto-completion feature is used, the color code is set automatically.

The *Time Slot* specifies the time-slot of the repeater for this channel. All repeaters have two time slots but different talk groups might be associated with each time slot. You may need to consult the webpage of the repeater. In the *Brandmeister network*, usually the time-slot 2 is for local/regional communication while time-slot 1 is for “DX”.

The *Group List* specifies the list of group-calls you want to receive on this channel. See the section called “Assembling Group Lists” above for details.

The Tx Contact specifies the default *Transmit Contact* you want to call on this channel when pressing the PTT button.

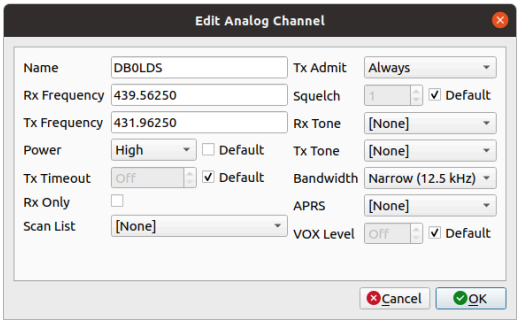
The Positioning System (DMR ARPS) specifies how you location information is send over this channel (selecting None disables GPS for this channel). Please note, that this setting is ignored for radios without GPS.

The Roaming allows to associate a roaming zone (see the section called “Roaming” below) with this channel. If the radios has the roaming feature, this zone is then used to search for a another channel of another repeater that is reachable once this repeater get out-of-range.

If Show Commercial Features is enabled in the settings dialog (see the section called “Application Settings Dialog”), a tab bar is shown at the top. There you can also access the device specific settings for the channel.

Edit FM channels

When you double-click on an analog channel or click on the Add FM Channel button, the analog-channel-editor dialog will be shown. This dialog allows for editing or creating FM channels.



The FM channel editor.

The left column of the editor are identical to the DMR channel dialog, that is Name, Rx and Tx Frequency, Power, Tx Timeout, VOX Level Rx Only and Scan List.

Like for DMR channels, analog channels may also have an Tx Admit criterion. Possible options are Always, Channel Free and Tone. For FM repeaters the Always option should be chosen to allow for a quick turn-around in a QSO. For simplex channels Channel Free should be chosen, as it only allows to transmit when the simplex channel is free. Selecting Channel Free on repeater channels, would prevent transmission while the repeater is active although the last transmission already ended.

The Squelch field specifies the squelch threshold. If Open is selected, the squelch is disabled. If Default is enabled, the global squelch level is used. See the section called “General configuration” above.

RX and TX Tone specify the *CTCSS*/*DCS* sub-tones for this channel/repeater. The RX Tone specifies the sub-tone that is needed to open the squelch. The TX Tone specifies the sub-tone that gets transmitted (e.g., to open the repeater). If the auto-completion feature is used, these tones are set automatically.

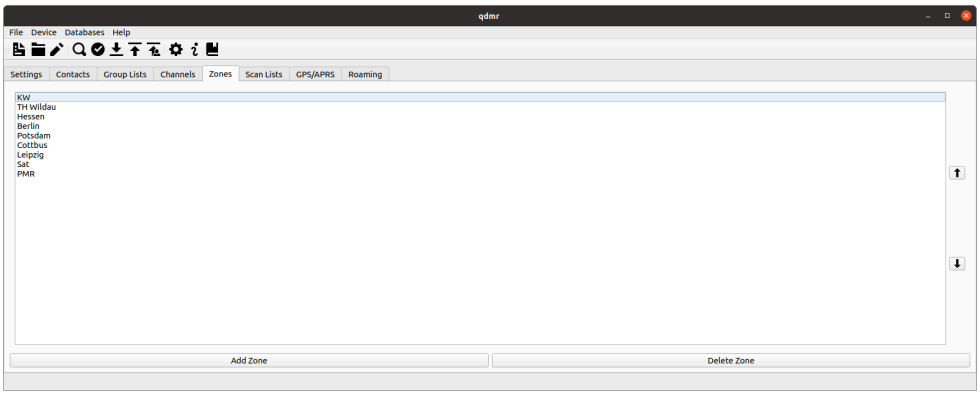
Bandwidth specifies the band-width of the transmission. Possible values are 12.5kHz (Narrow) or 25kHz (Wide).

APRS allows to specify an APRS system to be used on this channel. If the radio supports analog APRS, your position will be sent using the frequency, destination and path specified in this APRS system. If None is selected, APRS is disabled for this channel. If your radio does not support APRS, this setting is ignored.

If Show Commercial Features is enabled in the settings dialog (see the section called “Application Settings Dialog”), a tab bar is shown at the top. There you can also access the device specific settings for the channel.

Assembling Zones

You may program a myriad of different channels for you radio. To organize them in handy chunks, zones are used. That is, a zone is just a named list of channels that are relevant for a particular area or a particular situation.

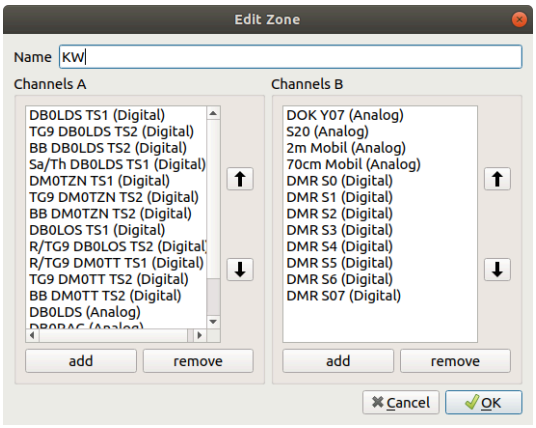


The list of zones.

The Zones tab just lists all defined zones. You may add a *Zone* using the Add Zone button or you may delete one by selecting the zone in the list and clicking on Delete Zone. You may also alter the ordering of the zones by selecting one from the list and using the up and down buttons on the right.

How zones are implemented differs from radio to radio. For example, some radios allow to set a different zone for each VFO (A or B), consequently these zones are simple lists of channels. Other radios allow to select a single zone for both VFOs. For these radios, a zone consists of two lists of channels. One for each VFO. qdmr zones follow the second approach. That is, a zone consists of two lists. One of each VFO. When programming radios that support only one channel list per zone, the zone is split into two (unless the second list is empty). One for each VFO and the A/B label is added to the name.

Double-clicking a zone or clicking on the Add Zone button will open the zone editor dialog.



The zone editor.

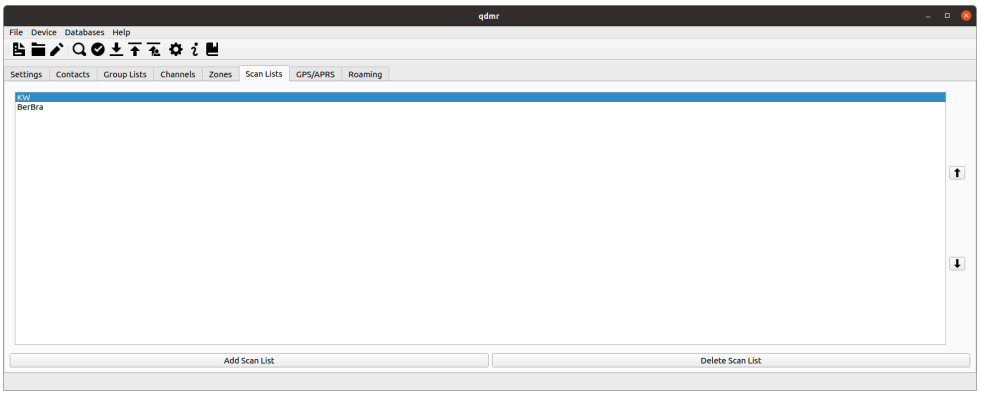
This dialog allows for adding or removing channels, and changing the order of the channels within the zone using the up- and down-buttons on the right.

Assembling Scan Lists

Scan lists are simple lists of channels that are scanned sequentially, when scanning is started. A *Scan-List* may be associated with an analog or digital channel (see the section called “Creating Channels”).

Note

For many radios, you need to associate a scan list with a channel (see the section called “Creating Channels” above) in the analog or digital channel edit dialog. This determines which scan list is used, when a scan is started on a particular channel.

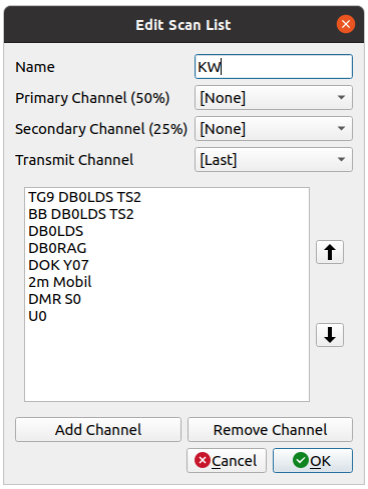


The list of scan lists.

A new scan list can be created by clicking on the Add Scan List button. A scan list can be deleted by selecting the scan list and clicking on the Delete Scan List button. The order of the scan list can be changed by selecting a list and moving it up/down using the up and down buttons on the right.

Double-clicking on a scan list or clicking the Add Scan List button will open the scan-list edit-dialog. This dialog allows to alter/assemble the scan list by adding, removing or reordering the channels in the scan list.

Edit Scan Lists



The scan-list editor.

By double-clicking an existing scan list or clicking on the Add Scan List button, the scan-list dialog opens. The Name field allows to specify the name of the scan list.

The optional Primary and Secondary Channel fields allow to specify channels that are visited more frequently. The Primary Channel will be visited 50% of the time. That is, after a channel of the scan list was visited, the primary channel is visited again, and after that, the next channel from the scan list is visited. In the end, the primary channel is scanned half of the time. The secondary channel is similar but gets visited only 25% of the time. The drop down list allows to select none, any channel or the Selected channel. The latter refers to the channel, the scan started on.

The optional Transmit Channel or *Revert Channel* specifies the channel to transmit on during a scan. Here none, any channel, the selected channel and also the Last channel can be chosen. The Last channel refers to the last active channel on the scan list. This allows to answer a heard call during a scan.

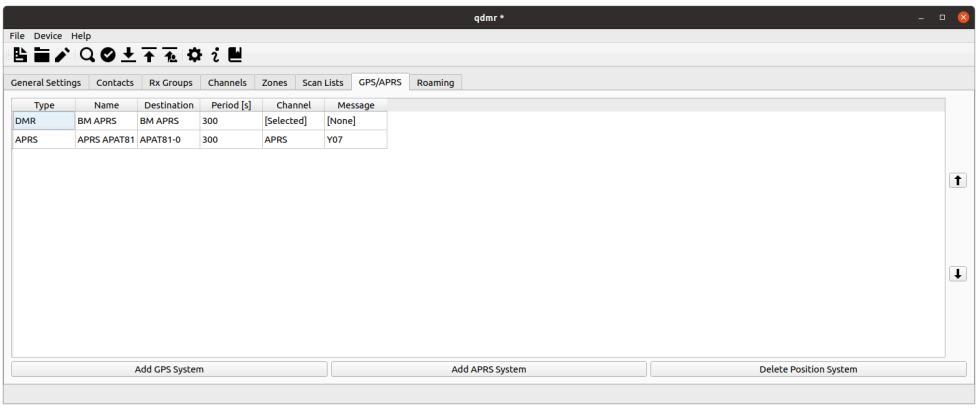
Channels can be added to the scan list by clicking on the Add Channel button at the bottom. Similarly, channels can be removed by selecting them and clicking on the Remove Channel button. Like for all

other lists, the channels can be moved around within the list by selecting channels and using the up and down buttons to the right.

If Show Commercial Features is enabled in the settings dialog (see the section called “Application Settings Dialog”), a tab bar is shown at the top. There you can also access the device specific settings for the scan list.

Setup GPS/APRS Position Reporting

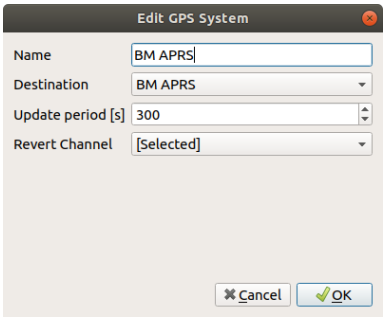
The GPS/APRS tab allows to specify several so-called positioning systems. A positioning system is a collection of settings on how GPS information is sent to *APRS* network.



The list GPS/APRS systems.

Edit/Create GPS Systems

Double-clicking a GPS system or clicking on the Add GPS System button will open the GPS system edit dialog.



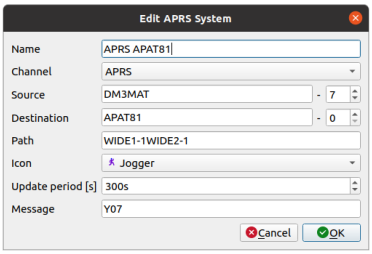
The digital APRS editing dialog.

This dialog allows to specify how the GPS information is sent through the DMR network. That is the Name field specifies the name of the GPS system. The Destination field specifies the private-call contact, the information is sent to. For example 262999 in the Brandmeister network. The Update Period specifies the period in which the current GPS information is sent to the contact. The *Revert Channel* specifies to which channel the radio should switch to, to send the position information. This should always be Selected. That is, the radio will always send the information on the currently selected channel.

If Show Commercial Features is enabled in the settings dialog (see the section called “Application Settings Dialog”), a tab bar is shown at the top. There you can also access the device specific settings for the GPS system.

Edit/Create APRS System

Double-clicking an APRS system or clicking on the Add APRS System button will open the *APRS* system dialog.



The analog APRS editing dialog.

This dialog allows to specify how the GPS information is sent through the APRS network. That is the Name field specifies the name of the APRS system. The Channel field specifies on which channel the APRS information should be send. This must be an analog channel with one of the typical APRS frequencies [https://en.wikipedia.org/wiki/Automatic_Packet_Reporting_System#Technical_information]. The Source field specifies your call and the source *SSID*, while the Destination field specifies the destination call and SSID. Path is an optional string containing the APRS path for the packet. The Icon combo-box allows to select an icon for the packets. These icons are then shown on APRS maps [https://aprs.fi]. The Update Period specifies how frequently an APRS packet should be send. Finally, the Message field allows to set an optional text message for the packet.

If Show Commercial Features is enabled in the settings dialog (see the section called “Application Settings Dialog”), a tab bar is shown at the top. There you can also access the device specific settings for the APRS system.

Roaming

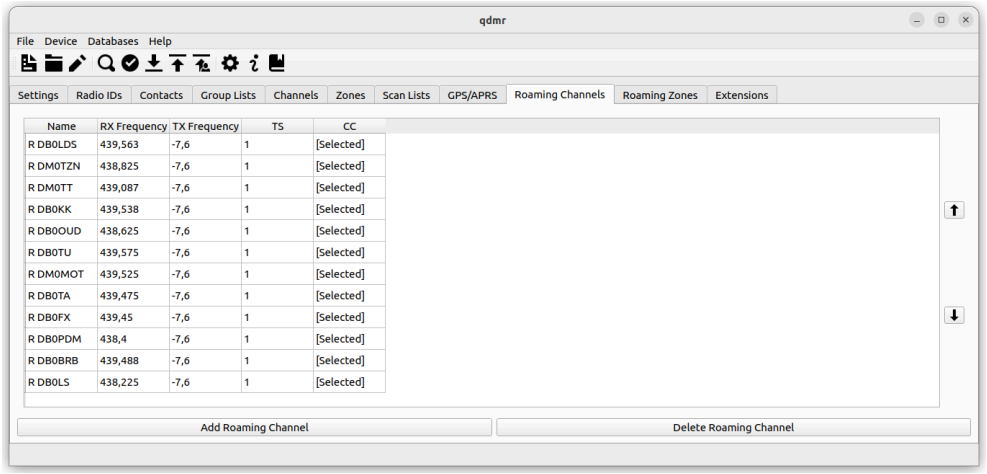
Roaming is a feature that allows DMR radios to select an alternative repeater once you leave the range of the currently selected one. To do that, you have to specify so-called *roaming zone*s. Within these zones, you collect all repeaters that provide access to a particular talk group. When roaming is enabled, the radio will check periodically, whether the current repeater is still reachable. If not, the strongest repeater from the selected roaming zone will then be selected instead.

In order to stay connected to a particular talk group, not the entire channel settings must be changed. But only those properties, that are specific to the repeater we change to. These are the RX and TX frequencies, color codes and sometimes the time-slot. The latter is necessary, as those repeaters may be located in other regions. Usually — at least for Brandmeister repeaters — the time slot 2 is reserved for regional communication while time slot 1 is intended for inter-regional communication. So you may need to override the time-slot of the active channel, whenever you roam outside of the region associated with the talk group. Some talk groups, however, are over-regional anyway. For example, the world-wide talk group 91. Here, the time slot will always be TS 1 and thus does not need to be overridden.

As only some properties of the current DMR channel needs to be overridden, there are special channels called *roaming channels* that contain only those settings, that are changed during roaming.

Roaming Channels

The Roaming Channels tab collects all defined roaming channels.



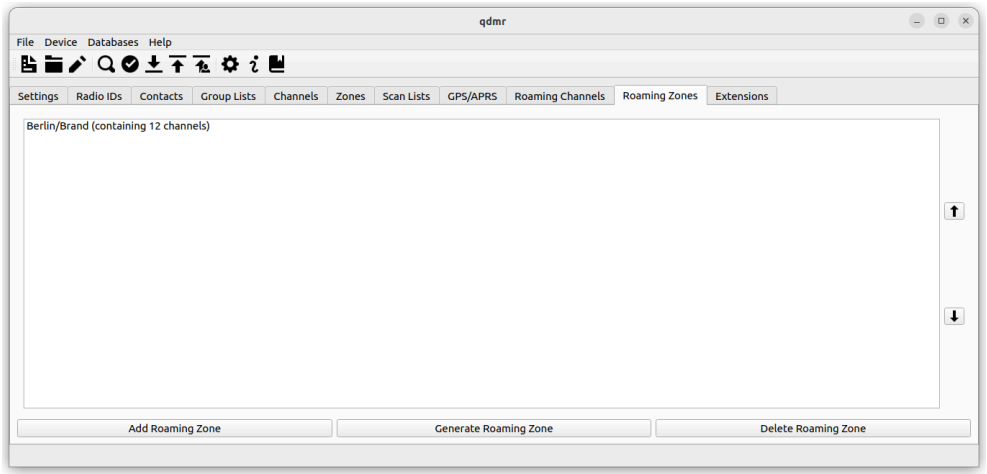
List of roaming channels.

Each roaming channel has a name, for easy reference in the roaming zones. This should be the call of the repeater. Additionally, each roaming channel has a transmit (TX) and receive (RX) frequency overriding the TX and RX frequencies of the current channel on roaming.

There are two optional settings, that may override the time slot (TS) and color code (CC) of the current channel. If [Selected] is show, the time slot and color code of the current channel is used instead during roaming.

Roaming Zones

The Roaming Zones tab collects all defined roaming zones.

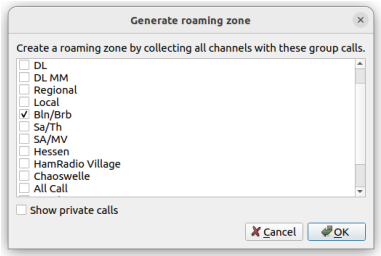


List of roaming zones.

There are two ways to create a new roaming zone. The easiest way is to generate a zone and the associated roaming channels automatically. Click on Generate roaming zone, to generate a roaming zone automatically or Add Roaming Zone to assemble one manually.

Creating a Roaming Zone

When generating a roaming zone automatically, a dialog will you to select a talk group or several talk groups to create a roaming zone for. That is, the talk group you want to stay connected to in case of losing contact to the repeater.



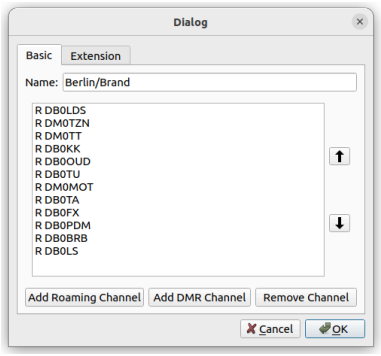
Selecting talk groups to generate a roaming zone.

Once a talk group is selected, a roaming channel created for each DMR channel, which has the selected talk group as its transmit contact. These newly created channels are then collected into a new roaming zone. The roaming zone editing dialog is then opened, allowing you to edit the newly created zone.

Alternatively, you may create a roaming zone manually. Simply click the Add Roaming Zone button and the roaming zone editing dialog will open. There you can add roaming channels manually.

Editing Roaming Zones

Double-clicking on the roaming zone or clicking on the Add Roaming Zone button will open the Roaming Zone Editor.



The roaming-zone editing dialog.

Here you can edit the name of the roaming zone as well as adding the roaming channels that are part of this zone. However, you may also add DMR channels to this zone by clicking on the Add DMR Channel button. This will not add DMR channel directly to the roaming zone, but will create a new roaming channel from this DMR channel.

If Show Commercial Features is enabled in the settings dialog (see the section called “Application Settings Dialog”), a tab bar is shown at the top. There you can also access the device specific settings for the roaming zone.

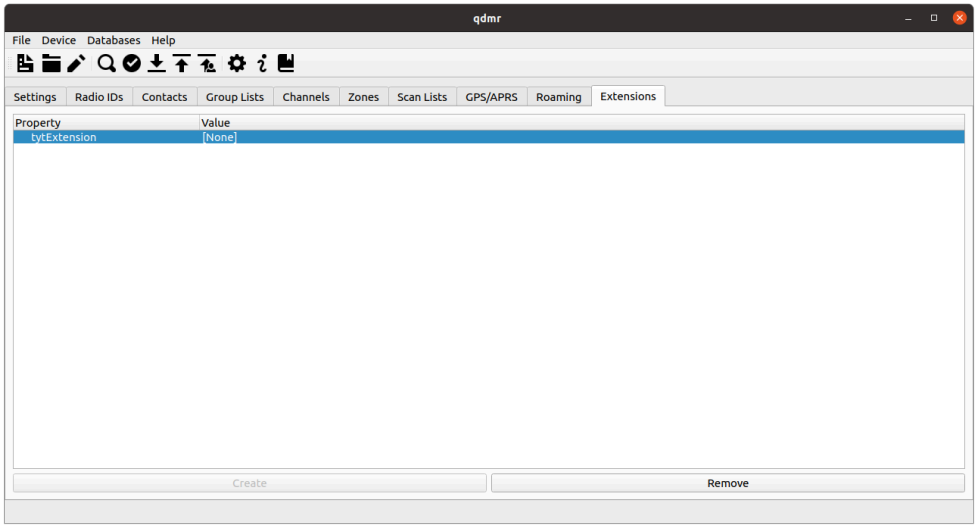
Edit Device Specific Extensions

Since version 0.9.2, qdmr supports the editing of so-called device-specific extensions. These extensions were introduced with version 0.9.0, to allow for configuring all features of a radio, although not represented within the common codeplug structure used by qdmr, to program all supported radios.

These extensions can be added to every element of a codeplug. For example contacts, channels, zones etc. They are stored in a common way to allow for an unified method for manipulating them. All extensions are displayed in a so-called “Extension View”, a tree-like structure present for all codeplug elements.

As qdmr is intended to be a clean and easy-to-use CPS for all radios, these device-specific settings are normally hidden. To show them, you need to enable the Show device extensions option in the settings dialog (see the section called “Application Settings Dialog”).

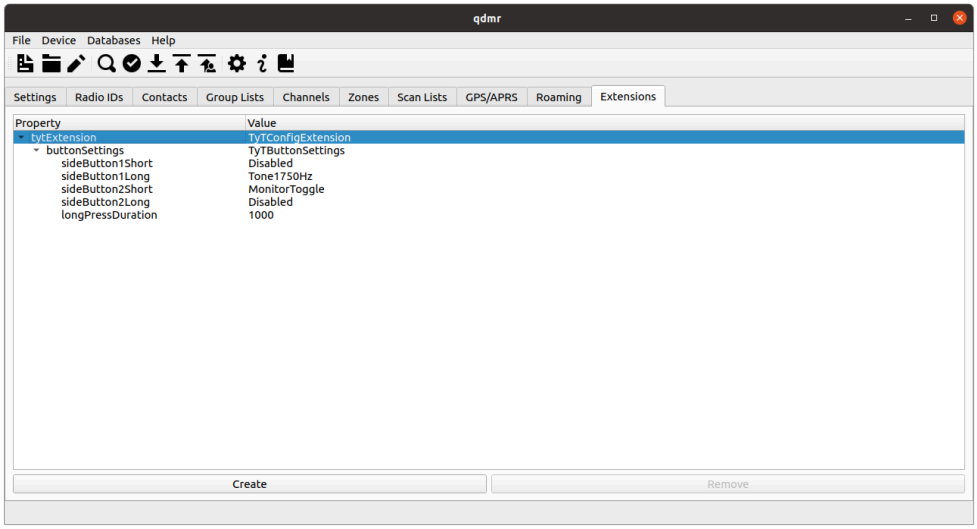
As this extension view is common to all codeplug elements, this section will discuss the usage on the codeplug extensions. When Show device extensions is enabled in the settings dialog, the codeplug extensions are shown in the last tab called Extensions.



Extension view for the codeplug extensions.

Unless a codeplug is read from a device, the extensions are usually not set (see screen shot above). In these cases, the Value column of the extension view shows [None]. You can add these extensions to the codeplug by selecting the corresponding extension and click on the Create button at the bottom of the extension view.

Similarly, an extension can be removed from the codeplug by selecting it and clicking on the Remove button at the bottom of the extension view.



Example for codeplug extensions for TyT devices.

Once an extension is added to the codeplug, it can be expanded to view and edit the contained settings. To edit a particular setting, simply double-click on the value.

Programming the radio

Once the *code plug* is finished, it can be programmed onto the radio. Just select the write cod plug button in the tool-bar at the top of the window or select write from the device menu.

In a first step, qdmr will try to detect a connected radio. This will be done automatically (unless disabled in the settings menu) if there is only one radio connected and it is save to access the USB device. Some radios use some generic USB to serial chips in their cable instead of connecting the micro controller of the radio directly to the computer. This way, it is not possible for qdmr to safely assume, that the found serial port is actually a radio. If such a generic interface is detected, qdmr will ask which interface the radio is connected to.

Once a radio is found it will verify the code plug with that radio. That is, it will check whether any limits are exceeded. For example the number of channels, contacts, group lists, etc.

There are several levels of issues that can be detected when verifying a code plug with a radio. The lowest level is the Information. These are just messages generated to inform you about minor changes made to the code plug to fit it into the specific radio. For example when zones are split. These information are usually ignored and qdmr will proceed writing the code plug.

Warnings are one level more severe. They are issued if changes are made that may change the behavior of the code plug. The result, however, will still be a working code plug. They are usually issued when names are too long. When warnings are issued, qdmr will not automatically proceed writing the code plug. The user, however, can ignore the warnings and continue. In the application settings (see the section called “Application Settings Dialog”), you may choose to always ignore verification warnings. In this case, qdmr will write the code plug automatically, even if there were some warnings.

Finally Errors are the most severe verification issues. They simply prevent writing the code plug to the device. The user cannot ignore errors as they would result in invalid and even damaging code plugs being written to the device.

If, however, everything fits into the radio, qdmr will start writing the binary code plug to the device.

Writing the code plug is a two-step process. First, the current code plug is read from the radio. This includes all settings. Then the device-specific code plug is updated and then re-uploaded to the device. This two-step process will maintain all device-specific settings made earlier unless explicitly set within qdmr.

During the reading or writing, the qdmr GUI will turn gray (inactive) to prevent any changes to the code plug during the transfer. However, a progress-bar is shown in the bottom-right to indicate the progress.

Permissions

When running qdmr or **dmrconf** under Linux, you may need to change the permissions to access USB devices. Along with the software, a **udev** rules file was installed. This file specifies that members of the `dialout` group have access to the radios. Consequently, you need to be a member of this group.

You can check your group membership with **groups**. This command lists all groups your user is a member of. This list should contain `dialout`.

If your user is not yet a member of the `dialout` group, you can add your user to it by calling

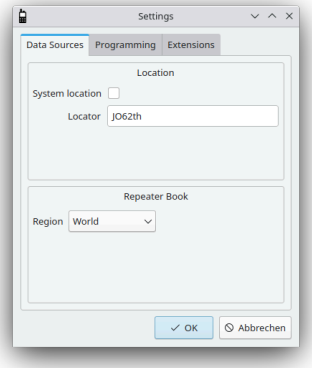
```
sudo adduser YOUR_USER dialout
```

Application Settings Dialog

The application settings dialog controls the behavior of qdmr. The dialog is divided into 3 sections, accessible via the tabs on the top.

Data Sources

The Data Sources tab collects the settings for the location service and repeater database.



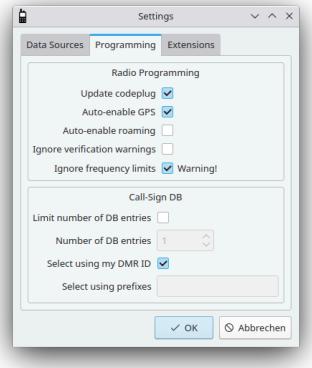
The settings dialog: Data sources

The first section concerns the location of the user. You may enter your *Maidenhead Locator* here or you may enable System location. The latter tries to obtain the current location from the operating system. This information is then used in the channel editors (see the section called “Creating Channels”) to provide auto-completion for repeaters nearby.

The second section allows to set the source for the repeater database. This enables qdmr to automatically complete some information for a repeater if its callsign is entered as the name, when creating a new channel. For some weird reason, the single data source Repeater Book [https://www.repeater-book.com/] provides two identical APIs for North America and the rest of the World. You must select your source accordingly.

Radio programming settings

The second tab controls, how the radios are programmed. That is, how the codeplug is assembled and also, how the callsign database gets curated.



The settings dialog: Programming

The Radio Interfaces section contains settings, controlling, how the radios are accessed. For now, there is only one setting, called disable auto-detect. This will disable all means to detect and identify connected radios. You will then have to select an interface and choose a radio model, every time you access it.

The second section specifies how codeplugs are assembled. The first option Update codeplug specifies whether a codeplug is generated from scratch or whether the codeplug currently programmed on the radio gets updated. qdmr does not implement all settings possible for all radios, consequently Update code plug should be chooses to maintain all settings of the radio that are not touched by qdmr.

For some radios, the GPS and roaming functionality must be enabled explicitly. The Auto-enable GPS and Auto-enable roaming options can be used to automatically enable GPS or roaming. If selected, whenever any channel has a GPS/APRS system or a roaming zone associated with it, the GPS and/or roaming gets enabled globally.

As described in the section called “Programming the radio”, the upload of a code plug will be paused if some verification warnings are issued. The Ignore verification warnings option allows to continue silently even in the presence of verification warnings. The may be needed for some radios with some rather short communication timeout. The radio may reset the connection to the computer while the warnings are shown. To prevent this, this option might be used.

The Ignore frequency limits option does exactly what it says. Usually, programming a channel outside of the radios frequency range would issue an error. However, many radios are able to receive and even transmit outside of the frequency range specified by the manufacturer. But be aware, that transmitting outside the declared frequency range may destroy the radio!

The Call-Sign DB section collects options that control the automatic curation of the call-sign DB. Many radios allow to write a large database of call-signs and DMR IDs to the radio. These DBs are then used to resolve DMR IDs to call-signs, names etc. and display them.

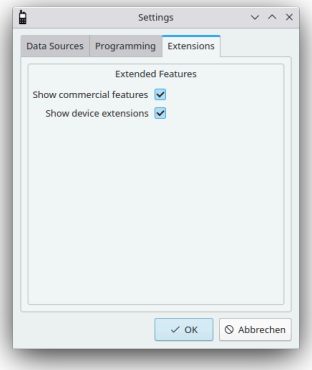
Usually, curating these databases is a cumbersome task. qdmr tries to automate this task. Usually, qdmr will select as many call-signs from the global database it can fit into the radio. Although modern radios will provide a huge amount memory, not all registered IDs can be programmed. In these cases, qdmr will select only the *closest* IDs to your DMR ID (default Radio DMR ID, see the section called “General configuration”). The DMR IDs are compared by the longest matching prefix. This makes sense as DMR IDs are not random. They share the same prefix for countries and regions. This way, qdmr will first select all IDs from the same region followed by all IDs from the same country etc. Of cause, there is no rule without any exceptions. Some countries have several prefixes assigned.

The Limit number of DB entries option and Number of DB entries field allow to limit the number of DB entries written to the device. If the Limit number of DB entries option is disabled, as many entries are written to the device as it can hold.

The Select using my DMR ID option and the Select using prefixes field can be used to control the selection of entries. If the Select using my DMR ID option is enabled, the aforementioned algorithm is used to select the entries based on the default DMR ID. If this option is disabled, a list of prefixes must be specified in the Select using prefixes field. This must be a list of comma-separated prefixes like `262, 263`. Whitespace are ignored. Then the DMR IDs closest to these prefixes are used to assemble the final call-sign DB.

Extension settings

The last tab collects settings, controlling several extended settings for the devices.



The settings dialog: Extensions

DMR originated as a standard for commercial radios. Consequently, there are many features that are not relevant or even illegal for ham-radio use (e.g., encryption). However, some operators use their handhelds for both ham-radio and commercial applications. Consequently, qdmr cannot ignore commercial features. It can, however, hide them. Enabling the Show commercial features options will show these features.

Starting with qdmr version 0.9.0, there are some device specific settings, allowing to configure features only present in one device and not represented in the general code plug. These are called *extensions* and are usually hidden. To show these device specific extension, select Show device extensions.

Chapter 3. Extensible Codeplug File Format

Tip

This document describes the extensible codeplug file format using YAML. If you are unfamiliar with YAML, consider reading a YAML introduction [https://learn.getgrav.org/16/advanced/yaml] first. The documentation for the *old* table based `conf`-file format, can be found in section Chapter 6, *Table Based Codeplug Format*.

The introduction of device specific settings (with version 0.9.0) required an extensible codeplug file format. The *old* table based format did not allow for any extension without braking backward compatibility. The new YAML based format allows for exactly that: Some means to extend the format for device specific settings without breaking the format while maintaining some degree of readability. (Yes, some users use the command line tool and edit their codeplug in a text editor.)

There are several levels, at which device specific extensions may appear within the codeplug. There are global extensions that apply to the entire codeplug. These extensions are located at the top level. There might also be extensions to single channels, contacts, zones, etc. These extensions are then located under the specific element that gets extended.

Radio settings

The radio settings section contains all radio-wide settings. For example the microphone level, boot text etc. For now, there aren't many.

As an example, consider the following general settings

Example 3.1. General radio-wide settings.

```
settings:
  micLevel: 6
  speech: false
  introLine1: qDMR
  introLine2: DM3MAT
  defaultID: id1
  squelch: 1
  vox: 0
  power: High
```

Here, the microphone amplification is set to 6, the speech synthesis is disabled, the two boot text lines are set to “qDMR” and “DM3MAT” respectively and the default DMR radio ID is set to “id1”. The latter is the id of a radio ID defined below.

Also, the radio-wide default squelch, VOX and power level is set. These values can be referenced later in channels. Also, some radios do not allow for these settings to be applied on a per-channel basis. For these radios, these values are used.

Radio-wide Setting Fields

`micLevel` Specifies the microphone amplification. Must be an integer between 1 and 10.

`speech` Enables/disables the speech synthesis. Some radios can announce the current channel etc. for the visually impaired. To enable that feature (if supported by the radio) set this field to `true`. Must be a boolean value.

`introLine1`, `introLine2` Sets the two boot text lines (if supported by the radio). These text lines will show up on the boot of the radio. If the radio is set to show a picture during boot, these lines are not shown.

`defaultID` Specifies which radio ID will be used as the default DMR ID (see the section called “Radio IDs” below). If none is specified, the first defined DMR radio ID will be used.

`squelch` Specifies the default squelch level. This value may be referenced in channels or represent the radio-wide squelch setting. Any value in [0-10] is valid, where 0 implies an open squelch (if supported by the radio).

`vox` Specifies the default VOX sensitivity. This value may be referenced in channels or represent the radio-wide VOX sensitivity. Any value in [0-10] is valid here, where 0 disables the VOX.

`power` Specifies the default transmit power. This value may be referenced in channels or represent the radio-wide power setting. Possible values are `Min`, `Low`, `Mid`, `High` and `Max`.

Radio IDs

The `radioIDs`-element specifies a list of radio IDs. Each radio ID is a map that contains a single entry to specify the type. The key of the entry specifies the type name and the value specifies the actual radio ID definition. Currently only the type `dmr` is supported.

As an example consider this radio ID list, containing only a single ID.

Example 3.2. Radio ID definitions

```
radioIDs:
  - dmr: {id: id1, name: DM3MAT, number: 2621370}
```

This DMR radio ID got the identifier `id1`, name `DM3MAT` and number `2621370`.

DMR Radio IDs (dmr)

The DMR radio ID definition consist of an optional `id` (necessary to reference that ID later), a `name` and the DMR ID `number`.

DMR Radio ID Fields

`id` Specifies the identifier of the radio ID. This identifier can later be used to associate the radio ID to channels. Any unique string is valid.

`name` Specifies the name of the radio ID. This name may also be used as the radio name. Any non-empty string is valid.

`number` Specifies the DMR ID for this radio ID. That is any integer between 0 and 16777215.

Contacts

The `contacts` element specifies a list of all contacts. Each contact is a map that contains a single entry to specify the contact type. The key specifies the type name and the value specifies the actual contact definition. Currently there are two possible contact types.

As an example consider this contact list, containing 4 contact definitions. One for each type.

Example 3.3. Contact definitions

```
contacts:
  - dmr: {id: ww, name: WW, type: GroupCall, number: 91}
  - dmr: {id: cont24, name: DM3MAT Hannes, type: PrivateCall, ring: true, number: 2621370}
  - dmr: {id: cont12, name: All Call, type: AllCall, number: 16777215}
  - dtmf: {id: dtmf1, name: DTMF example, number: "#123*"}
```

DMR Contacts (dmr)

A DMR contact is a simple object and is usually defined in one line. Each contact contains an optional `id` that will be used to reference this contact throughout the codeplug (e.g., in channels, group lists, etc.).

DMR Contact Fields

`id` Specifies the identifier of the contact. This identifier can later be used to reference the contact. Any unique string is valid.

`name` Specifies the name of the contact. Any string is valid.

`type` Specifies the type of the contact. Must be one of `PrivateCall`, `GroupCall` or `All-Call`.

`number` Specifies the DMR ID for this contact. That is, any integer between 0 and 16777215. This element is mandatory for all types except for the `all`-call. For the all-call, the default number 16777215 will be used.

`ring` If `true`, the radio will ring whenever a call from this contact is received (if supported by the radio). Optional, if omitted set to `false`.

Analog DMTF Contacts (dtmf)

An analog DTMF contact can be used to store commonly used DTMF sequences. For example, it may be used to control the *EchoLink* feature of a repeater.

`id` Specifies the identifier of the contact. This identifier can later be used to reference the contact. Any unique string is valid.

`name` Specifies the name of the contact. Any string is valid.

`number` Specifies the DTMF ID for this contact. That is any combination of numbers `0-9` and symbols `A-D`, `*`, #. used.

`ring` If `true`, the radio will ring whenever a call from this contact is received (if supported by the radio). Optional, if omitted set to `false`.

Group lists

A group list collects several digital (DMR) contacts that should be received on a channel associated with this group list. Consequently, a group list consists of a name and a list of contact IDs. The `groupLists` element is then a list of several group list definitions.

As an example, consider the two group lists below:

Example 3.4. Group list definition

```
groupLists:
  - {id: grp1, name: DL, contacts: [cont1, cont2, cont3]}
  - {id: grp2, name: Berlin/Brand, contacts: [cont6, cont5, cont7]}
```

The first group list has the internal ID *grp1*. This ID can then be used later to reference this group list. The name is set to *DL*. The list of contacts of the group list is then defined as a list containing the IDs of the referenced contacts.

Group list fields

`id` Specifies the ID of the group list. This ID can then be used to refer to the group list within digital channels. Any unique string is possible here.

`name` Specifies the name of the group list. Any string is possible here.

`contacts` A list of digital contact IDs. See the section called “Contacts” above.

Channels

The `channels` list contains all channels defined within the codeplug. Usually, DMR radios support at least two different channel types. Digital channels (DMR) and analog channels (FM). To distinguish these two types, each entry of the channel list contains a map with a single entry. The key specifies the type (either `digital` or `analog`) while the value contains the actual channel definition.

As an example consider the following two channel definitions:

Example 3.5. Channel definition

```
channels:
  - digital:
      id: ch5
      name: BB DB0LDS TS2
      rxFrequency: 439.5625
      txFrequency: 431.9625
      power: High
      timeout: 0
      rxOnly: false
      vox: !default
      scanList: scan1
      admit: ColorCode
      colorCode: 1
      timeSlot: TS2
      groupList: grp2
      contact: cont7
      radioID: !default
      roaming: !default
  - analog:
      id: ch76
      name: DB0LDS
      rxFrequency: 439.5625
      txFrequency: 431.9625
      power: High
      timeout: 0
      rxOnly: false
      vox: !default
      squelch: !default
      admit: Always
      bandwidth: Narrow
      rxTone: {ctcss: 67}
      txTone: {ctcss: 67}
```

The first channel is a digital (DMR) channel. Its ID is *ch5* and its name *BB DB0LDS TS2*. The second channel is an analog channel (FM) with ID *ch76* and name *DB0LDS*. Please note, that there are some common attributes like `rxFrequency`, `txFrequency`, `power`, `timeout`, `rxOnly`, `vox` and `scanList` but also type-specific settings like `timeSlot` or `rxTone` which apply only to digital and analog channels respectively.

Common attributes

The following attributes are common for all channel types.

Common channel fields

`id`	Specifies the ID of the channel, this can be used later to reference this channel in the section called “Zones” and the section called “Scan lists”. Any unique string is valid here.
`name`	Specifies the name of the channel. Any string is valid here.
`rxFrequency`, `txFrequency`	Specifies the RX or TX frequency in MHz. Any floating point number is valid here.
`power`	Specifies the transmit power of the channel. Must be one of `Min`, `Low`, `Mid`, `High` or `Max`.
`timeout`	Specifies the transmit timeout in seconds. Any integer is valid here. Omitting this field or setting it to `0` will disable the timeout. Setting it to `!default`, the global transmit timeout will be used. See the section called “Radio settings”.
`rxOnly`	If set to `true`, the channel can only receive. Omitting it or setting it to `false` will allow transmission on the channel.
`vox`	Specifies the VOX sensitivity for this channel. If set to `!default`, the global VOX sensitivity will be used. See the sec-

tion called “Radio settings”. Omitting this value or setting it to 0, will disable the VOX for this channel.

`scanList` Specifies the optional scan list for this channel. If set, this must be a reference to a the section called “Scan lists”.

Digital channel attributes

The following attributes apply only to digital (DMR) channels.

Digital channel fields

`admit` Specifies the admit criterion for the channel. This must be one of `Always`, `Free` or `ColorCode`.

`colorCode` Specifies the color code of the channel, any number between 0-16 is valid here.

`timeSlot` Specifies the time slot of the channel. Must be one of `TS1` or `TS2`.

`groupList` Specifies the RX group list for this channel. This must be a reference to a group list. See the section called “Group lists” above.

`contact` Specifies the default transmit contact. This must be a reference to a digital contact. See the section called “Contacts” above. If omitted, no default transmit contact is associated with the channel. Some radios require all channels to have a transmit contact.

`aprs` Specifies the positioning system for this channel. If set, this must be a reference to a positioning system (DMR or APRS). See the section called “Positioning Systems” below.

`roaming` Specifies the roaming zone for this channel. If set it must be a reference to a roaming zone. See the section called “Roaming” below. To use the default roaming zone here, consider using the `!default` tag instead of a direct reference to a specific zone. If omitted, no roaming zone (also not the default one) is associated with the channel.

`radioID` Specifies the radio ID for this channel. If set, it must be a reference to one of the radio DMR IDs. See the section called “Radio IDs”. To use the default radio ID here, consider using the `!default` tag instead of a direct reference to a specific ID. If omitted the default radio ID is assumed.

Analog channel attributes

The following attributes apply only to analog (FM) channels.

Analog channel fields

`admit` Specifies the admit criterion for the channel. Must be one of `Always`, `Free` or `Tone`.

`squelch` Specifies the squelch level for the channel. Must be an integer in the range of [1-10]. If set to 0, the squelch gets disabled. If set to `!default`, the global squelch setting is used. See the section called “Radio settings” above.

`bandwidth` Specifies the bandwidth of the channel. Must either be `Narrow` or `Wide` for 12.5kHz or 25kHz respectively.

`rxTone` Specifies the receive sub-tone setting for this channel. That is, the squelch will only open when a certain subtone is received along with the signal. As there are two common subtone standards, this attribute is a map with a single entry. The key specifies the

type (either `ctcss` or `dcs`) while the value specifies the actual subtone. For CTCSS tones, the value is the subtone frequency in Hz. For DCS it is the code number as an integer. For inverted DCS codes, use negative numbers.

`txTone` Specifies the transmit sub-tone setting for this channel. For details see `rxTone` above.

`aprs` Specifies the APRS positioning system of this channel. If set, it must be a reference to an analog APRS system.

Zones

The `zones` element collects all zones defined within the codeplug. It is just a list of zone definitions. Each zone has an ID, name and one or two lists of channels. One for VFO A and one for VFO B. Depending on the radio, a zone will be split automatically into two zones if the radio handles separate zones for each VFO.

As an example, consider the following zone:

Example 3.6. Zone definition

```
zones:
  - id: zone1
    name: KW
    A: [ch1, ch4, ch5, ch2, ch6, ch11, ch12, ch13, ch15, ch16, ch76, ch77, ch78, ch79]
    B: [ch97, ch101, ch102, ch103, ch68, ch69, ch70, ch71, ch72, ch73, ch74, ch75]
```

This zone has the name *KW* and contains two lists of channels. One for each VFO. On radios, where each VFO is assigned a zone individually, this zone will be split into two: *KW A* and *KW B* to match the radios configuration.

Zone fields

`id` Specifies the ID of the zone. For now, there are no codeplug elements that refer to zones.

`name` Specifies the name of the zone. Any string is valid here.

`A` Specifies the channel list for VFO A. This must be a list of references to channels.

`B` Optional channel list for VFO B. If present, must be a list of references to channels.

Scan lists

Scan lists are simple lists of channels to scan. A scan list might be associated with a channel.

As an example, consider the following scan list:

Example 3.7.

```
scanLists:
  - id: scan1
    name: KW
    channels: [ch4, ch5, ch76, ch77, ch97, ch102, ch68, ch112]
```

This scan list has the ID *scan1*, the name *KW* and contains several channels (both analog and digital).

Scan list fields

`id` Specifies the ID of the scan list. This ID can then be used to reference this scan list in the section called “Channels”.

`name` Specifies the name of the scan list. Any string is valid here.

`primary` Specifies the primary priority channel. Usually this channel is scanned very frequently. If set, this must be a reference to a channel. If the tag `!selected` is used here, the channel from which the scan got started is used as the primary priority channel.

`secondary` Specifies the secondary priority channel. Usually this channel is scanned frequently. If set, this must be a reference to a channel. If the tag `!selected` is used here, the channel from which the scan got started is used as the secondary priority channel.

`revert` Specifies the revert channel. That is, the channel to transmit on irrespective of the current channel being scanned. If set, this must be a reference to a channel. If the tag `!selected` is used here, the channel from which the scan got started is used as the transmit channel. If omitted the radio will transmit on the currently scanned channel.

`channels` Specifies the list of channels to scan. Must be a list of channel IDs.

Positioning Systems

Some radios allow to send the current position using DMR or analog APRS. Consequently, there are two types of positioning systems `dmr` and `aprs`.

As an example, consider these two position reporting systems:

Example 3.8.

```
positioning:
  - dmr:
      id: aprs1
      name: BM ARPS
      period: 300
      contact: cont21
  - aprs:
      id: aprs2
      name: APRS APAT81
      period: 300
      revert: ch104
      icon: Jogger
      message: Y07, QRG 144.675
      destination: APAT81-0
      source: DM3MAT-7
      path: [WIDE1-1, WIDE2-1]
```

The first specifies a digital (DMR) positioning system while the latter defines a APRS system.

Common attributes

The following attributes apply to both, analog APRS and digital DMR position reporting systems.

Common fields

`id` Specifies the ID of the system. This can be used reference this system in the section called “Channels”. Any unique string is valid here.

`name` Specifies the name of the position reporting system. Any string is valid here.

`period` Specifies the update period in seconds. If omitted or set to `0`, the system will not send any updates periodically.

DMR position reporting system attributes

The following attributes apply only to digital (DMR) position reporting systems.

DMR position reporting system fields

`contact` Specifies the digital contact, the GPS information is sent to. This must be a reference to a digital contact.

`revert` Specifies the revert channel. That is, the channel the data is sent on. If set, it must be a reference to a digital channel. If omitted or set to `!selected`, the currently active channel will be used to send the GPS information.

APRS attributes

The following attributes apply only for APRS position reporting systems.

Analog position reporting system (APRS) fields

`revert` Specifies the revert channel. That is, the channel the APRS information is sent on. This must be a reference to an analog channel. Usually a dedicated analog APRS channel is defined and referenced here.

`destination` Specifies the destination call and SSID, the information is sent to. This must be a string in the form `CALL-SSID`. Please note that the call must match the format defined by the AX.25 protocol. That is, it may only contain letters and numbers and cannot exceed the length of 6.

`source` Specifies the source call and SSID. See `destination` for details.

`path` Specifies the optional packet path. If set, this must be a list of `CALL-SSID` strings.

`icon` Specifies the icon name to use. The icon name will be fuzzy matched. See for a complete list [https://dm3mat.darc.de/qdmr/libdmrconf/classAPRSSystem.html] of icon names.

`message` Specifies an optional message sent along with the position information.

Roaming

The roaming configuration consists of a list of roaming channels and roaming zones. While roaming channels specify those channel settings, that might be overridden during roaming, the roaming zone specifies a collection of roaming channels, that can be used to maintain a connection to a particular talk group.

Roaming Channels

Roaming channels contain those settings of a DMR channel, that are specific for a particular repeater. This allows to override these settings in a channel, when the repeater of that channel gets out of range. Then, the radio may switch to a different repeater to maintain the connection to the current talk group.

Example 3.9.

```
roamingChannels:
  - id: rch1
    name: R DB0LDS
    rxFrequency: 439.5625
    txFrequency: 431.9625
  - id: rch2
    name: R DB0AFZ
    rxFrequency: 439.97500
    txFrequency: 430.57500
    colorCode: 1
    timeSlot: 1
```

The example above shows the definition of two roaming channels. Both are used to stay in contact with a regional talk group within the Brandmeister network. While the repeater DB0LDS is located within this region, DB0AFZ is not. Consequently, the time-slot of the channel needs to be overridden.

Roaming channel fields

`id` — Specifies the ID of the roaming channel. This ID can then be used to refer to this channel within a roaming zone. Any unique string is valid here.

`name` — Specifies the name of the roaming channel. Any string is valid here.

`rxFrequency`, `txFrequency` — Specifies the receive and transmit frequencies of the channel.

`colorCode` — Specifies the color code of the channel. If set, overrides the color code of the current channel. If omitted, the color code of the selected channel is used during roaming.

`timeSlot` — Specifies the time slot of the channel. If set, overrides the time slot of the current DMR channel. If omitted, the time slot of the selected channel is used during roaming.

Roaming Zones

Roaming zones are collections of roaming channels, that are scanned for the strongest signal to maintain connection to a particular network or talk group.

As an example, consider the following roaming zone:

Example 3.10.

```
roamingZones:
  - id: roam1
    name: Berlin/Brand
    channels: [rch1, rch2, rch3]
```

This zone has the ID *roam1* and the name *Berlin/Brand*. This group collects all roaming channels that provide access to the *Berlin/Brandenburg* talk group.

Roaming zone fields

`id` — Specifies the ID of the roaming zone. This ID can then be used to refer to this zone. Any unique string is valid here.

`name` — Specifies the name of the roaming zone. Any string is valid here.

`channels` — Specifies the member channels for this roaming zone. This must be a list of references to roaming channels.

Chapter 4. Device specific extensions

The sole reason for introducing a new YAML based codeplug format was the ability to extend the file format with device specific settings without breaking the format. This was simply impossible with the *old* table based text files.

The present YAML based codeplug file format is extensible at almost any level. That is, device specific elements can be added to single codeplug elements like channels, zones or contacts but also to the codeplug itself. The latter allows to extend the codeplug with new elements.

OpenGD77 Codeplug Extensions

This chapter documents the extensions and settings specific to radios running the OpenGD77 firmware.

Note

The OpenGD77 codeplug is a specialization of the GD77 codeplug. Consequently all extensions described in the section called “Radioddity™ Codeplug Extensions” are also applicable to the OpenGD77 codeplug.

Channel extension

This extension allows to specify some channel settings specific for devices running the OpenGD77 firmware. This extension can be added to any channel, analog and digital. For now, this extension only allows to specify the power for the channel in more detail.

Example 4.1.

```
channels:
  - digital:
      id: ch1
      name: Example channel
      # all the other channel settings
      openGD77:
        power: P750mW
```

The OpenGD77 channel extension is a mapping named `openGD77`. It contains the device specific settings for that channel.

Channel attributes

For now, there are only few attributes specifying more detailed power settings and scan behavior for the channel.

Channel extension fields

`power` Specifies the detailed power settings for the channel. This is one of the following strings `Global`, `P50mW`, `P250mW`, `P500mW`, `P750mW`, `P1W`, `P2W`, `P3W`, `P4W`, , `P5W`, `Max`. Where `Global` implies that the global power settings is used and `Max` the maximal possible power is used (usually 7W on VHF and 5.5W on UHF). This power setting overrides the common channel power setting. See the section called “Channels”.

`scanZoneSkip` `scanAllSkip` In contrast to the original GD-77 firmware, OpenGD77 does not implement scanning by scan lists. Instead, either all channels defined in the radio or all channels within the current zone can be scanned. To exclude some channels from these scans, the `scanZoneSkip` and `scanAllSkip` flags can be used. When enabled, the channel will be excluded from the respective scan.

DMR contact extension

This extensions allows to specify some DMR contact attributes for devices running the OpenGD77 firmware. This extension is only applicable to DMR (digital) contacts.

Example 4.2.

```
contacts:
  - dmr:
    name: Example contact
    number: 1234567
    openGD77:
      timeSlotOverride: TS1
```

The OpenGD77 contact extension is a mapping named `openGD77`. It contains the device specific settings for that DMR contact.

Contact attributes

For now, there is only one attribute allowing to override the time slot of a channel whenever this contact is selected as the destination contact for that channel.

Channel extension fields

`timeSlotOverride` OpenGD77 allows to override the time slot settings for each channel on the bases of the selected transmit contact. If the contact has a time slot override set, this time slot is used instead of the channel time slot. This attribute is either `None`, `TS1` or `TS2`. If `None` is set, the channel time slot will not be overridden.

Radioddity™ Codeplug Extensions

This section collects all device specific extensions to the codeplug for Radioddity™ and some Baofeng™ devices.

These extensions are applicable to Radioddity™ GD77, GD73 and Baofeng/Radioddity RD-5R radios.

Radio settings extension

This extension allows to set the device-specific general settings for Radioddity™ radios. It extends the `settings` section of the codeplug and is split into several sub-extensions.

General radio settings

This section describes the top-level generic radio settings for Radioddity devices.

Example 4.3. Radio settings extension example showing default values.

```
settings:
  [...]
  radioddity:
    monitorType: Open
    loneWorkerResponseTime: 1min
    loneWorkerReminderPeriod: 10s
    groupCallHangTime: 3000ms
    privateCallHangTime: 3000ms
    downChannelModeVFO: false
    upChannelModeVFO: false
    powerSaveMode: true
    preambleDuration: 360ms
    wakeupPreamble: true
    allLEDsDisabled: false
    quickKeyOverrideInhibited: false
    txOnActiveChannel: false
    animation: false
    scanMode: Time
    repeaterEndDelay: 0s
    repeaterSTE: 0s
```

Radio settings fields

`monitorType` — Specifies the monitor type. Must be either `Open` or `Silent`. Default `Open`.

`loneWorkerResponseTime` — When lone worker is enabled, specifies the time in minutes before the radio will start to remind the user. Default 1min. See also Lone Worker.

`loneWorkerReminderPeriod` — Specifies the period in seconds for the lone worker reminder. Default 10s. See also Lone Worker.

`groupCallHangTime` — Specifies the group-call hang time in milliseconds. This is the time-period the user can answer a received group call by pressing PTT. After this time has passed, a press on the PTT button will call the default contact on the selected channel. Default 3000ms. See also Hang Time.

`privateCallHangTime` — Specifies the private-call hang time in milliseconds. This is the time-period the user can answer a received private call by pressing PTT. After this time has passed, a press on the PTT button will call the default contact on the selected channel. Default 3000ms. See also Hang Time.

`downChannelModeVFO` — If `true`, the channel-up button will tune the VFO. If `false`, it will step through the channels. Default `false`.

`upChannelModeVFO` — If `true`, the channel-donw button will tune the VFO. If `false`, it will step through the channels. Default `false`.

`powerSaveMode`	Puts the radio is a sleep mode when in idle state (no traffic on the channels). This allows for some power saving. However, the radio will need some time to wake up. Consequently, all other radios in the network need to transmit a wake-up preamble. Default `true`.
`wakeupPreamble`	Enables the transmission of a short wake-up preamble allowing receiving radios to wake-up in time for the actual transmission. Default `true`.
`preambleDuration`	This sets the preamble duration in milliseconds. Default 360ms.
`powerSaveDelay`	Specifies the delay, before an idle radio enters power save mode. (GD-73 only)
`allLEDsDisabled`	If `true`, all LEDs are disabled. Default `false`.
`quickKeyOverrideInhibited`	If `true`, allows the user to transmit on a busy channel irrespective of the channels admit criterion by double pressing the PTT. Default `false`.
`txInterrupt`	WTF?!?
`txOnActiveChannel`	If `true`, the radio will transmit on the currently active channel (if double-wait) is enabled. Default `false`.
`scanMode`	Specifies the scan mode. Must be one of `Time`, `Carrier` or `Search`. Default `Time`.
`repeaterEndDelay`	Specifies the delay after the end of a repeater transmission in seconds. Default 0s, off.
`repeaterSTE`	Specifies the repeater STE (what ever that means) in seconds. Default 0s, off.
`language`	Specifies the UI language. Must be one of `Chinese` or `English`.

Button settings

This section describes how the buttons are configured for Radioddity devices.

Example 4.4. Button settings extension example showing default values.

```
settings:
  [...]
  radioddity:
    [...]
    buttons:
      longPressDuration: 1 s
      funcKey1Short: ZoneSelect
      funcKey1Long: ToggleFMRadio
      funcKey2Short: ToggleMonitor
      funcKey2Long: ToggleFlashLight
      funcKey3Short: BatteryIndicator
      funcKey3Long: ToggleVox
```

Button settings fields

`longPressDuration`	Sets the duration, a button must be pressed, to be considered as a “long press”. This interval is usually expressed in ms. E.g., 1000ms.

`funcKey1Short`, `funcKey1Long`	Short and long-press functions for the programmable function key 1. This is the side key 1 on the GD77 and RD-5R and the **P1** button on the GD73.
`funcKey2Short`, `funcKey2Long`	Short and long-press functions for the programmable function key 2. This is the side key 2 on the GD77 and RD5R and the **P2** button on the GD73.
`funcKey3Short`, `funcKey3Long`	Short and long-press functions for the programmable function key 3. This is the top key on the GD77 and RD5R This button is not present on the GD73.

Button functions

`None`	Disables the button. No function is associated with it.
`ToggleAllAlertTones`	Enables or disables all alert tones. Only present in GD77 and RD5R radios.
`EmergencyOn`, `EmergencyOff`	Why not toggle? Either enables or disables an emergency.
`ToggleMonitor`	Enables/toggles the monitor. This is device specific, on some radios the monitor function latches, on most not. Then, the monitor is enabled, as long as the button is pressed.
`OneTouch1`, `OneTouch2`, `OneTouch3`, `OneTouch4`, `OneTouch5`, `OneTouch6`	Triggers one specific one-touch action. Not all radios have 6 of these. The GD-77 and RD-5R have 6 one-touch actions, while the GD-73 has only 5.
`ToggleTalkaround`	Enables/disables the talkaround feature for repeater channels. The radio then also transmits on the RX frequency. Consequently bypassing the repeater.
`ToggleScan`	Enables/disables the scan.
`ToggleEncryption`	Enables/disables the encryption for the channel, if configured.
`ToggleVox`	Enables/disables the VOX for the channel, if configured.
`ZoneSelect`	Brings up the zone selection dialog.
`BatteryIndicator`	Shows the battery charge indicator.
`ToggleLoneWorker`	Enables/disables lone-worker feature, if configured.
`PhoneExit`	WTF?!?
`ToggleFlashLight`	Enables/Disables the flash light. Not all devices have one.
`ToggleFMRadio`	Enables/disables the FM broadcast radio.
`RadioCheck`,`RadioDisable`, `RadioEnable`	If configured, the radio will transmit tones, that cause other radios --- if configured to do so --- to either response, disable or reenable themselves. This allows to control other radios remotely.
`TBST`	Sends the TBST tone (usually 1750Hz). Some radios have a fixed button combo for that.
`CallSwell`	WTF?!?

Tone settings

This section collects some settings relates to tones and other audio stuff.

Example 4.5. Tone settings extension example showing default values.

```
settings:
  [...]
  radioddity:
    [...]
    tone:
      lowBatteryWarn: false
      lowBatteryWarnInterval: 30 s
      lowBatteryWarnVolume: true
      keyTone: false
      keyToneVolume: true
      callAlertDuration: 2 min
      resetTone: false
      unknownNumberTone: false
      artsToneMode: Once
      digitalTalkPermitTone: false
      analogTalkPermitTone: false
      selftestTone: true
      channelFreeIndicationTone: false
      allTonesDisabled: false
      txExitTone: false
      fmMicGain: 1
```

Tone settings fields

`lowBatteryWarn` — Enables the low battery-charge warning. This can either be a notification on the screen or a waring tone. The warning interval and tone-volume might be set by `lowBatteryInterval` and `lowBatteryWarnVolume`.

`lowBatteryWarnInterval` — Specifies the interval, at which low battery warning are issued.

`lowBatteryWarnVolume` — Specifies the volume of the low-battery warning tone in a range from 1 to 10.

`keyTone`, `keyToneVolume` — If `true`, the key-pad tones are enabled. Don't do it. The volume of these tones might be set using `keyToneVolume` in a range from 1 to 10.

Boot settings

This section collects some settings related to booting the radio.

Example 4.6. Boot settings extension example showing default values.

```
settings:
  [...]
  radioddity:
    [...]
    boot:
      displayMode: Text
      bootPassword: ""
      progPassword: ""
```

Boot settings fields

`display` — Specifies what to display during boot. Must be one of `None`, `Text` or `Image`.

`bootPassword`, `progPassword` — Specifies the boot and programming passwords. The former (usually only numbers) must be entered, when the radio boots.

The latter must be entered in the CPS to program or read the codeplug.

TyT™ Codeplug Extensions

This section collects all device specific extensions to the codeplug for TyT™, Retevis™ and some Baofeng™ devices.

These extensions are applicable to TyT™ MD-390, MD-UV380, MD-UV390, MD-2017, Retevis™ RT8, RT3S, RT82, RT84 and Baofeng™ DM-1701 radios.

Channel extension

This extension to the codeplug allows to set device specific channel settings for many TyT™ radios (and therefore also many Retevis™ radios). Not all settings are present in all radios. Unsupported settings are ignored during encoding or set to the default value during decoding.

Example 4.7. Example channel settings specifying the default values for TyT™ Retevis™ devices.

```
tyt:
  loneWorker: false
  autoScan: false
  talkaround: true
  dataCallConfirmed: false
  privateCallConfirmed: false
  emergencyAlarmConfirmed: false
  displayPTTId: true
  rxRefFrequency: Low
  txRefFrequency: Low
  tightSquelch: false
  compressedUDPHeader: false
  reverseBurst: true
  killTone: Off
  inCallCriterion: Always
  allowInterrupt: false
  dcdm: false
  dcdmLeader: false
```

Common channel setting fields

`loneWorker` — If `true`, the lone worker feature is enabled. See also Lone Worker

`autoScan` — If `true`, the auto-scan feature is enabled.

`talkaround` — If `true`, talk around is enabled. That is, the radio will receive on the input and transmit on the output frequency of the repeater effectively bypassing the repeater. See also Talkaround.

`dataCallConfirmed`, `privateCallConfirmed`, `emergencyAlarmConfirmed` — Enables the confirmation of data, private and emergency calls. These fields are usually disabled as the radio will first establish the connection before the actual call can be started.

`displayPTTId` — If `true`, the received analog PTT will be shown.

`rxRefFrequency`, `rxRefFrequency` — Specifies some weird reference frequency setting for RX and TX. By default the value `Low` is used. Possible values are `Low`, `Medium` and `High`.

Channel settings for TyT™ MD-390 and Retevis™ RT8

`tightSquelch` — If set to `true`, the silent squelch is used.

`compressedUDPHeader` Some unknown flag. Usually disabled.

`reverseBurst` If enabled (default), avoids a noise burst at the end of a transmission, when the squelch is controlled by a sub tone.

Channel settings for TyT™ MD-UV390 and Retevis™ DM-2017

`killTone` Specifies the kill tone. Possible values are `Off`, `Tone259_2Hz` and `Tone55_2Hz`. Disabling or setting the kill tone to 259.2Hz or 55.2Hz, respectively.

`inCallCriterion` Specifies the in-call criterion. Possible values are `Always`, `AdmitCriterion` and `TXInterrupt`.

`allowInterrupt` Enables/disables interrupts if `inCallCriterion` is set to `TXInterrupt`.

`dcdm` If enabled, the dual-capacity direct mode is used for simplex operation. See also DCDM.

`dcdmLeader` If DCDM is enabled, this flag specifies whether this radio is the channel leader. That is, if this radio provides the clock for the time-slots.

`dmrSquelch` Allows to set the squelch level for DMR channels. This prevents the LED to light all the time. The radio, however, remains silent irrespective of this setting.

Scan-list settings extension

This extension to the codeplug allows to specify device specifics settings for scan-lists.

Example 4.8. Scan-list settings example showing the default values.

```
tyt:
  holdTime: 500
  prioritySampleTime: 2000
```

Scan-list settings fields

`holdTime` Specifies the hold time in ms.

`prioritySampleTime` Specifies the sample-time in ms for priority channels.

Button settings extension

This extension to the codeplug allows to specify the function for each of the programmable buttons on the radio. Please note that not all TyT™ radios have all buttons described here.

Example 4.9.

```
tytExtension:
  buttonSettings:
    longPressDuration: 1000
    sideButton1Short: Disabled
    sideButton1Long: Tone1750Hz
    sideButton2Short: MonitorToggle
    sideButton2Long: Disabled
    sideButton3Short: Disabled
    sideButton3Long: Disabled
    progButton1Short: Disabled
    progButton1Long: Disabled
    progButton2Short: Disabled
    progButton2Long: Disabled
```

Button settings fields

`sideButton1Short`, `sideButton1Long`, `sideButton2Short`, `sideButton2Long`, `sideButton3Short` (Retevis RT84, Baofeng DM-1701), `sideButton3Long` (Retevis RT84, Baofeng DM-1701), `progButton1Short` (Retevis RT84, Baofeng DM-1701), `progButton1Long` (Retevis RT84, Baofeng DM-1701), `progButton2Short` (Retevis RT84, Baofeng DM-1701), `progButton2Long` (Retevis RT84, Baofeng DM-1701)	Specifies the different functions for each button press. Must be one of: `Disabled`, `ToggleAllAlertTones`, `EmergencyOn`, `EmergencyOff`, `PowerSelect`, `MonitorToggle`, `OneTouch1`-`OneTouch6`, `RepeaterTalkaroundToggle`, `ScanToggle`, `SquelchToggle`, `PrivacyToggle`, `VoxToggle`, `ZoneIncrement`, `BatteryIndicator`, `LoneWorkerToggle`, `RecordToggle`, `RecordPlayback`, `RecordDeleteAll`, `Tone1750Hz`, `SwitchUpDown`, `RightKey`, `LeftKey` or `ZoneDecrement`.
`longPressDuration`	Specifies the long-press duration in milliseconds.

Menu settings extension

This extension to the codeplug allows to specify which menu items are enabled. This feature is frequently used in commercial applications to restrict the user to change certain settings in the radio.

Example 4.10.

```
tytExtension:
  menuSettings:
    hangtimeIsInfinite: false
    hangTime: 10
    textMessage: true
    callAlert: true
    contactEditing: true
    manualDial: true
    remoteRadioCheck: true
    remoteMonitor: true
    remoteRadioEnable: true
    remoteRadioDisable: true
    scan: true
    scanListEditing: true
    callLogMissed: true
    callLogAnswered: true
    callLogOutgoing: true
    talkaround: true
    alertTone: true
    power: true
    backlight: true
    bootScreen: true
    keypadLock: true
    ledIndicator: true
    squelch: true
    vox: true
    password: true
    displayMode: true
    radioProgramming: true
    gpsInformation: true
```

Menu settings fields

`hangtimeIsInfinite`, `hangTime`	Specify the menu hang time. That is the duration, the menu is shown without any user action. If `hangtimeIsInfinite` is `true`, the menu is shown indefinitely, irrespective of the `hangTime` setting. If `hangtimeIsInfinite` is `false`, the `hangTime` specifies the menu hang time in seconds.

`textMessage`	Enables the text-messaging menu item.
`callAlert`	Enables the call alert menu item.
`contactEditing`	Enables the contact editing menu item.
`manualDial`	Enables manual dial.
`remoteRadioCheck`	Enables the remote radio-check menu item.
`remoteMonitor`	Enables the remote monitor menu item.
`remoteRadioEnable`, `remoteRadioDisable`	Enables the remote radio enable and disable menu items.
`scan`, `scanListEditing`	Enables the scan and scan list editing menu items.
`callLogMissed`, `callLogAnswered`, `callLogOutgoing`	Enables the menu items for the list of missed, answered and outgoing calls.
`talkaround`	Enables the talkaround menu item.
`alertTone`	Enables the alert tone settings menu item.
`power`	Enables the power settings menu item.
`backlight`	Enables the backlight settings menu item.
`bootScreen`	Enables the boot-screen settings menu item.
`keypadLock`	Enables the keypad lock settings menu item.
`ledIndicator`	Enables the LED indicator settings menu item.
`squelch`	Enables the squelch settings menu item.
`vox`	Enables the VOX settings menu item.
`password`	Enables the password settings menu item.
`displayMode`	Enables the display mode settings menu item.
`radioProgramming`	Enables radio programming from the keypad. E.g., editing channels etc.
`gpsInformation`	Enables the GPS information menu item. This setting has only an effect on radios supporting GPS.

Radio settings extension

This extension allows to set the device-specific general settings for TyT™ and Retevis™ radios. It extends the `settings` section of the codeplug.

Example 4.11. Radio settings extension example showing default values.

```
settings:
  [...]
  tyt:
    montitorType: Open
    allLEDsDisabled: false
    talkPermitToneDigital: false
    talkPermitToneAnalog: false
    passwordAndLock: false
    channelFreeIndicationTone: true
    allTonesDisabled: false
    powerSaveMode: true
    wakeupPreamble: true
    bootPicture: true
    channelMode: true
    channelModeA: true
    channelModeB: true
    txPreambleDuration: 600
    channelHangTime: 3000
    groupCallHangTime: 3000
    privateCallHangTime: 3000
    lowBatteryWarnInterval: 120
    callAlertToneContinuous: false
    callAlertToneDuration: 0
    loneWorkerResponseTime: 1
    loneWorkerReminderTime: 10
    digitalScanHangTime: 1000
    analogScanHangTime: 1000
    backlightAlwaysOn: false
    backlightDuration: 10
    keypadLockManual: true
    keypadLockTime: 60
    powerOnPasswordEnabled: false
    powerOnPassword: 0
    radioProgPasswordEnabled: false
    radioProgPassword: 0
    pcProgPassword: null
    privateCallMatch: true
    groupCallMatch: true
```

Radio settings fields

`montitorType`	Specifies the monitor type. Possible values are `Open` and `Silent`. This setting only affects analog channels. If `Open` is specified, the squelch will open if monitoring is enabled.
`allLEDsDisabled`	If enabled, all LEDs are disabled.
`talkPermitToneDigital`	If enabled, a talk-permit tone will sound on digital channels.
`talkPermitToneAnalog`	If enabled, a talk-permit tone will sound on analog channels.
`passwordAndLock`	Enables and disables the passwords and keypad locks globally. They can also be enabled/disabled individually below.
`channelFreeIndicationTone`	If enabled, a tone will sound after a transmission has ended to indicate that the channel is free again.
`allTonesDisabled`	If enabled, all tones (talk permit, channel free, etc.) are disabled globally.
`powerSaveMode`	Puts the radio is a sleep mode when in idle state (no traffic on the channels). This allows for some power saving. However, the radio will need some time to wake up. Consequently, all other radios in the network need to transmit a wake-up preamble.
`wakeupPreamble`	Enables the transmission of a short wake-up preamble allowing receiving radios to wake-up in time for the actual transmission.

`bootPicture`	If enabled, a picture is shown during boot rather than the boot text (see `introLine1` and `introLine2` in the general settings section).
`channelMode, channelModeA, channelModeB`	Controls the mode of the radio, VFO A and VFO B. If `channelMode` is `true`, the entire radio operates in channel mode (memory mode). This also overrides the `channelModeA` and `channelModeB` settings. If `channelMode` is set to `false`, the `channelModeA` and `channelModeB` specify the mode for the VFO A and B, respectively.
`txPreambleDuration`	Specifies the transmit preamble duration in milliseconds. If this preamble is the wake-up preamble, is unknown.
`channelHangTime`	Specifies the channel hang time in milliseconds.
`groupCallHangTime`, `privateCallHangTime`	Specifies the group-call hang time in milliseconds. This is the time a group or private call can be answered directly even if that group call is not the default contact of the channel. See also Hang Time.
`lowBatteryWarnInterval`	Specifies the period of the low-battery warn tone in seconds.
`callAlertToneContinuous`	If enabled, a call-alert tone will sound until the operator reacts. This setting overrides the `callAlertToneDuration` setting.
`callAlertToneDuration`	Specifies the call alert-tone duration in seconds. If `callAlertToneContinuous` is enabled, the alert tone will be continuous irrespective of this setting.
`loneWorkerResponseTime`	Sets the lone-worker response time in minutes. See also Lone Worker.
`loneWorkerReminderTime`	Sets the lone-worker reminder time in minutes. See also Lone Worker.
`digitalScanHangTime`, `analogScanHangTime`	Specifies the time in milliseconds, the radio will continue monitoring a DMR or FM channel after the transmission on that channel ended.
`backlightAlwaysOn`	If enabled, the backlight will stay on.
`backlightDuration`	Specifies the backlight duration in seconds.
`keypadLockManual`	If enabled, the keypad lock is enabled manually. If not, the keypad lock gets enabled automatically after a specified period (see `keypadLockTime`).
`keypadLockTime`	Specifies the time, after which the keypad lock is engaged automatically unless `keypadLockManual` is enabled.
`powerOnPasswordEnabled`	Enables the power-on password.
`powerOnPassword`	Specifies the power-on password. An 8-digit number. This password must then be entered when booting the radio.
`radioProgPasswordEnabled`	Enables the radio-programming password.
`radioProgPassword`	Sets the radio-programming password. An 8-digit number. This password must then be entered when making any changes to the radio/channel settings through the keypad.

`pcProgPassword` Specifies the PC programming password. This password is then needed when programming the radio through the CPS. An empty string disables this password.

`privateCallMatch`, `groupCallMatch` If `true`, the private and group call IDs must match.

AnyTone™ Codeplug Extensions

This chapter documents the extensions and settings specific to AnyTone™ radios. Please note, that not all attributes are applicable to all devices.

DMR contact extension

This extensions allows to specify some DMR contact attributes for AnyTone™ devices. This extension is only applicable to DMR (digital) contacts.

Example 4.12.

```
contacts:
  - dmr:
    name: Example contact
    number: 1234567
    anytone:
      alertType: Online
```

The AnyTone™ contact extension is a mapping named `anytone`. It contains the device specific settings for that DMR contact.

Contact attributes

For now, there is only one attribute allowing to override the alert type for a DMR contact.

Channel extension fields

`alertType` The alert type specifies the notification type when a call from that contact is received. To this end, this field overrides the `ring` setting of the common contact settings. This attribute is either `None`, `Ring` or `Online`. If `None` is set, the notification is disabled.

Channel extensions

This extensions allows to set some device specific channel settings for AnyTone™ devices. There are some general settings, valid for FM and DMR channels as well some FM and DMR specific settings.

Example 4.13. Examples of the AnyTone™ extensions for FM and DMR channels.

```
channels:
  - dmr:
    name: Example DMR channel
    # ...
    anytone:
      talkaround: false
      frequencyCorrection: 0
      handsFree: false
      callConfirm: false
      sms: true
      smsConfirm: false
      dataACK: true
      simplexTDMA: false
      adaptiveTDMA: false
      loneWorker: false
      throughMode: false
  - fm:
    name: Example FM channel
    # ...
    anytone:
      talkaround: false
      frequencyCorrection: 0
      handsFree: false
      reverseBurst: false
      rxCustomCTCSS: false
      txCustomCTCSS: false
      customCTCSS: 0.0
      squelchMode: Carrier
      scrambler: false
```

Common channel attributes

This section describes the common channel attributes. These attributes are shared between FM and DMR channels.

`talkaround` — If `true`, talkaround is enabled for this channel. This will bypass the repeater on repeater channels and allows to talk to other participants directly.

`frequencyCorrection` — Specifies some sort of correction to the frequency in some unknown units. Consider contacting me, if you know more details about it.

`handsFree` — If `true`, the hands-free feature is enabled for this channel. The actual configuration of the hands-free feature happens in the general settings extension. This setting is only applicable to the AT-D578UV device.

`fmAPRSFrequency` — Optional alternative FM APRS frequency specified in the AnyTone extension to the APRS settings.

FM channel attributes

These attributes are only valid for FM channels.

`reverseBurst` — If `true`, a CTCSS phase-reverse burst is sent at the end of transmission. This phase reversal of the CTCSS tone at the end of the transmission can be used to prevent a noise tail at the end of the transmission on the receiver side. See Reverse CTCSS [https://en.wikipedia.org/wiki/Continuous_Tone-Coded_Squelch_System#Reverse_CTCSS] for details.

`rxCustomCTCSS` — If `true`, the custom CTCSS frequency (see `customCTCSS`) is used to control the squelch.

`txCustomCTCSS` — If `true`, the custom CTCSS frequency (see `customCTCSS`) is transmitted.

`customCTCSS` — Specifies a custom CTCSS frequency in Hz. The frequency can be specified with a resolution of 0.1Hz.

`squelchMode` Specifies the squelch mode. Must be one of `Carrier`, `SubTone`, `OptSig`, `SubToneAndOptSig` or `SubToneOrOptSig`.

`scrambler` If `true`, the analog scrambler is enabled. It is not legal to use scramblers in HAM radio.

DMR channel attributes

This section describes the attributes, applicable to DMR channels on AnyTone™ devices.

`callConfirm` If `true`, the call confirmation is enabled.

`sms` If `true`, the SMS reception is enabled.

`smsConfirm` If `true`, the SMS confirmation is enabled.

`dataACK` If `true`, the transmission of data acknowledgement is enabled.

`simplexTDMA` If `true`, TDMA mode for simplex channels is enabled. This mode is also known as DCDM (dual-capacity direct mode, see DCDM)

`adaptiveTDMA` If `true`, the adaptive TDMA mode is enabled. This makes only sense, if `simplexTDMA` is enabled too. In this case, the radio is able to receive both simplex TDMA as well as normal simplex DMR on the channel.

`loneWorker` If `true`, the lone-worker feature (see Lone Worker) is enabled for this channel. Usually not used in HAM radio.

`throughMode` If `true`, the through mode is enabled. What ever that means. If you know more about it, consider extending this section.

Zone extension

This extensions allows to specify some zone attributes for AnyTone™ devices.

Example 4.14.

```
zones:
    - id: zone1
      name: Zone Name
      A: [ch1, ...]
      B: [ch2, ...]
      anytone:
        hidden: false
```

The AnyTone™ zone extension is a mapping named `anytone`. It contains the device specific settings for that zone.

Zone attributes

For now, there is only one attribute allowing to hide the zone from the zone list in the radio.

Zone extension fields

`hidden` If `true`, the zone is hidden in the zone list on the radio.

Settings extension

This extensions allows to specify some device specific settings for AnyTone™ devices. As there are a myriad of settings for these devices, the settings extension is split into several sub-extensions, grouping the settings by domain.

The AnyTone™ settings extension is a mapping named `anytone`. It contains the device specific settings for AnyTone devices.

General settings attributes

General settings extension fields

`subChannel`	If `true`, the sub-channel is enabled.
`selectedVFO`	Specifies the currently selected VFO. Must be one of `A` or `B`, selecting VFO A and B, respectively.
`modeA`, `modeB`	Specifies the VFO mode for VFOs A and B. Must be one of `Memory` or `VFO`. The former selects the memory/channel mode while the latter selects the VFO mode.
`zoneA`, `zoneB`	Specifies the current zones for VFO A and B. Must be a reference to a zone.
`vfoScanType`	Specifies scan-type for the VFO mode. Must be one of `Time`, `Carrier` or `Stop`. Default is `Time`. If `Time` is selected, the scan is stops for a specified period. If `Carrier` is selected, the scan stops as long as a carrier is detected. If `Stop` is selected, the scan stops entirely.
`minVFOScanFrequencyVHF`, `maxVFOScanFrequencyVHF`, `minVFOScanFrequencyUHF`, `maxVFOScanFrequencyUHF`	Specifies the frequency ranges for the VFO scan in the VHF and UHF bands. These frequency can be specified arbitrarily. If a unit-less floating point value is given, it is interpreted as in MHz. If a unit-less integer value is given, it is interpreted as in Hz.
`keepLastCaller`	If `true`, the last caller ID is kept even, when the channel is changed. Seriously, who needs that option? Just keep it or don't. Why is it so important? AnyTone codeplugs are full of these unnecessary settings.
`vfoStep`	Specifies the VFO tuning step-size.
`steType`, `steFrequency`, `steDuration`	Specifies the squelch-tail elimination type, frequency and duration. The type must be one of `Off`, `Silent`, `Deg120`, `Deg180` or `Deg240`. The frequency is specified as a floating point number in Hz. The duration is specified in ms. This option is applicable to AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV devices.
`tbstFrequency`	Specifies TBST/pilot-tone frequency used to enable/open a repeater. This should be one of the common frequencies like 1000, 1450, 1750 and 2100 Hz. This option is applicable to AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV devices.
`proMode`	If `true`, enables the professional mode. If enabled, some menu options are hidden. This option is applicable to AT-D878UV, AT-D878UV II and DMR-6X2UV devices.
`maintainCallChannel`	If `true`, maintains the call-channel. Whatever that now means. If you find it out, let it me know. This option is applicable to AT-D878UV, AT-D878UV II and DMR-6X2UV devices.

`bootSettings`, `powerSaveSettings`, `keySettings`, `toneSettings`, `displaySettings`, `audioSettings`, `menuSettings`, `autoRepeaterSettings`, `dmrSettings`, `roamingSettings`, `gpsSettings`, `simplexRepeaterSettings`	Many more funny settings, grouped by domain. For details, see the descriptions below.

Boot settings

Collects some settings, that control the startup of the radio. These settings are collected under the `bootSettings` key within the `anytoneSettings` entry.

Boot settings extension fields

`bootDisplay`	Specifies what is seen on the display during boot. Must be one of `Default`, `CustomText` or `CustomImage`.
`bootPasswordEnabled`, `bootPassword`	Enables/disables and specifies the boot password. The boot password is a string of up to 8 number chars.
`defaultChannel`, `zoneA`, `channelA`, `zoneB`, `channelB`	If `true`, `defaultChannel` enables the specified zones and channels to be set on startup. `zoneA` and `zoneB` specify the startup zone for VFOs A and B, respectively, and must reference a defined zone. While `channelA` and `channelB` specify the startup channels for each VFO. If no channels are defined, the VFO is chosen.
`priorityZoneA`, `priorityZoneB`	Specifies the priority zones for each VFO. These must reference a defined zone.
`defaultRoamingZone`	Specifies the startup roaming zone. Must refer to a defined roaming zone. This option is applicable to AT-D878UV, AT-D878UV II and AT-D578UV devices.
`gpsCheck`	Enables the GPS check on boot. This option is applicable to AT-D878UV and AT-D878UV II devices.
`reset`	Enables a MCU reset on boot. This option is applicable to AT-D878UV and AT-D878UV II devices.

Power-save settings

Collects some settings, that configure the power-save features. These settings are collected under the `powerSaveSettings` key within the `anytoneSettings` entry.

Power-save setting extension fields

`autoShutdown`	Specifies the time-out for the automatic shut down. If `0`, the automatic shut-down is disabled.

`resetAutoShutdownOnCall` If `true`, the automatic showdown timer is reset on every call.

`powerSave` Specifies the power-save mode. Must be one of `Off`, `Save50` or `Save66`. The latter two will save 50% or 66%, respectively.

`atpc` Enables the ATPC (Adaptive Transmission Power Control). It reduces the transmission power, if a strong signal is received.

Keypad settings

Collects some settings, that configure the keypad, lock etc. These settings are collected under the `keySettings` key within the `anytoneSettings` entry.

This extension also allows to assign specific functions to the programmable function keys on the radio. These are the side-keys, top-key and the front buttons labeled **P1** and **P2**. The following list shows all functions, that can be assigned to each of these buttons. Some function are not available on all devices.

Key functions

`Off` No function at all.

`Voltage` Shows the battery voltage. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.

`Power` Toggles though the power settings. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.

`Repeater` Toggles the talk-around feature. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.

`Reverse` Swaps the RX and TX frequencies. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.

`Encryption` Shows the encryption key selection. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.

`Call` Performs an FM call with the configured DMTF/two-tone/five-tone ID. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.

`VOX` Shows the VOX level selection. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II and DMR-6X2UV.

`ToggleVFO` Toggles between VFO and memory/channel mode. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV.

`SubPTT` PTT for the sub channel. Only applicable for **PF1**, **PF2** and **PF3** keys. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II and DMR-6X2UV.

`Scan` Starts/stops a scan. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.

`WFM` Toggles the WFM (or Air band) receiver. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.

`Alarm`	Starts/stops an alarm call. Applicable to AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`RecordSwitch`	Enables/disables recoding function. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`Record`	Start/stops a recoding. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`SMS`	Start writing an SMS. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`Dial`	Start manual dial. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`GPSInformation`	Applicable to AT-D868UV, AT-D578UV and DMR-6X2UV.
`Monitor`	Enables the (FM) monitor. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`ToggleMainChannel`	Toggles between VFOs A and B for the main channel. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`HotKey1`, `HotKey2`, `HotKey3`, `HotKey4`, `HotKey5`, `HotKey6`	Triggers one of the programmed hot-key function. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`WorkAlone`	Toggles the lone-worker mode. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`SkipChannel`	During a scan, use this function to skip the current channel. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`DigitalMonitor`	Toggles the digital monitor. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`SubChannel`	Enables/disables the sub-channel. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`PriorityZone`	Change to the priority zone. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`VFOScan`	Toggles the VFO scan. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`MICSoundQuality`	Toggle “enhanced” sound mode in DMR mode. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`LastCallReply`	Allows to answer the last call, after the hang-time has passed. Applicable to AT-D868UVE, AT-D878UV, AT-D578UV and DMR-6X2UV.
`ChannelType`	Change the channel type. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.

`SimplexRepeater`	Enables/disables the simplex repeater feature. Applicable to DMR-6X2UV.
`Ranging`	Applicable to AT-D868UV, AT-D578UV and DMR-6X2UV
`Roaming`	Manually starts roaming. Applicable to AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`ChannelRanging`	Applicable to AT-D868UV, AT-D578UV and DMR-6X2UV
`MaxVolume`	Shows the maximum volume setting. Applicable to AT-D868UV, AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`Slot`	Changes the time-slot, AT-D868UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`Squelch`	Shows squelch settings. Applicable to AT-D578UV and DMR-6X2UV.
`APRSTypeSwitch`	Applicable to AT-D578UV.
`Zone`	Shows the zone selection dialog. Applicable to AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`RoamingSet`	Shows the roaming settings(?). Applicable to AT-D878UV, AT-D878UV II and DMR-6X2UV.
`APRSSet`	???
`Mute`	Mutes the radio for a specified interval. Applicable to AT-D878UV, AT-D878UV II and DMR-6X2UV.
`MuteA`, `MuteB`	Mutes the VFO A/B for a specified interval. Applicable to AT-D578UV.
`XBandRepeater`	Enables the cross-band repeater function. Applicable to AT-D578UV.
`Speaker`	Applicable to AT-D578UV.
`CtcssDcsSet`	Shows the CTCSS/DCS settings for FM channels. Applicable to AT-D878UV, AT-D878UV II and DMR-6X2UV.
`TBSTSend`	Sends the TBST tone. Applicable to AT-D878UV, AT-D878UV II, AT-D578UV.
`Bluetooth`	Enables/disables bluetooth function. Applicable to AT-D878UV, AT-D878UV II, AT-D578UV.
`GPS`	Enables/disables GPS. Applicable to AT-D878UV, AT-D878UV II, AT-D578UV.
`ChannelName`	Toggle channel name and frequency display. Applicable to AT-D878UV, AT-D878UV II, AT-D578UV.
`CDTScan`	Starts CTCSS/DCS scan for FM channel. Applicable to AT-D878UV, AT-D878UV II, AT-D578UV.
`APRSSend`	Transmits the positioning information. Applicable to AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.

`APRSInfo`	Shows received APRS information. Applicable to AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`GPSRoaming`	Applicable to AT-D578UV.
`NoiseReductionTX`	Applicable to AT-D578UV.

The following list contains all keypad setting fields.

Keypad settings extension fields

`funcKey1Short`, `funcKey1Long`, `funcKey2Short`, `funcKey2Long`, `funcKey3Short`, `funcKey3Long`, `funcKey4Short`, `funcKey4Long`, `funcKey5Short`, `funcKey5Long`, `funcKey6Short`, `funcKey6Long`	Specifies the functions associated with the function keys **P1**-**P6** for a short and long press. Must be one of the items in the key function list above. The function keys 1-6 are labeled **P1**-**P6** and are located either below or next to the display.
`funcKeyAShort`, `funcKeyALong`	Specifies the function associated with the programmable function key **PF1** or **A**. This button is located right below the PTT button (on handheld radios) or labeled **A** on the hand microphone (AT-D578UV).
`funcKeyBShort`, `funcKeyBLong`	Specifies the function associated with the programmable function key **PF2** or **B**. This button is the second one below the PTT button (on handheld radios) or labeled **B** on the hand microphone (AT-D578UV).
`funcKeyCShort`, `funcKeyCLong`	Specifies the function associated with the programmable function key **PF3** or **C**. THis button is the colorful one on the top (on handheld radios) or labeled **C** on the hand microphone (AT-D578UV).
`funcKeyDShort`, `funcKeyDLong`	Specifies the function associated with the programmable function key **D**. This button is located on the hand microphone (AT-D578UV only).
`longPressDuration`	Specifies the minimum duration of a “long press” on one of the programmable function keys.
`autoKeyLock`	Enables/disables the automatic key lock.
`knobLock`, `keypadLock`, `sideKeysLock`	Specifies, whether the knob, keypad and side-keys are locked, when the key-lock is enabled.
`forcedKeyLock`	Don't use. Prevents the manual key-lock release. Only applicable to commercial applications.

Tone settings

Collects some settings, that configure signal tones. These settings are collected under the `toneSettings` key within the `anytoneSettings` entry.

Tone settings extension fields

`keyTone`	Enables the key-pad tones. Should be disabled.
`keyToneLevel`	Specifies the key-tone level, 0 means user adjustable.

`smsAlert`	Enables the SMS alert tone.
`callAlert`	Enables the call alert tone. See also `callMelody`.
`dmrTalkPermit`, `fmTalkPermit`	Enables the DMR talk permit tone.
`dmrReset`	Enables the DMR reset tone. See also `resetMelody`.
`dmrIdle`, `fmIdle`	Enables the idle tone (for DMR and FM channels). See also `idleMelody`.
`startup`	Enables a startup tone.
`tot`	Transmit time-out warning tone.
`callMelody`, `idleMelody`, `resetMelody`, `callEndMelody`	Specifies the melody played on each of these occasions. These melodies can contain up to 5 notes (and pauses). For the sake of simplicity, qdmr supports the LilyPond musical notation for these melodies. E.g., `a8 b e2 des4 d`.

Display settings

Collects all display settings. These settings are collected under the `displaySettings` key within the `anytoneSettings` entry.

Display settings extension fields

`displayFrequency`	If true, the channel frequency is shown instead of the channel name.
`brightness`	Specifies the brightness of the backlight in a range 1-10.
`backlightDuration`, `backlightDurationRX`, `backlightDurationTX`	Specifies the backlight duration.
`customChannelBackground`	If `true`, the custom channel background is enabled.
`volumeChangePrompt`, `callEndPrompt`	If `true` the volume change and call-end prompts are shown.
`showClock`, `showCall`, `showContact`, `showChannelNumber`, `showColorCode`, `showTimeSlot`, `showChannelType`, `showLastHeard`	Enables/disables various display elements.
`lastCallerDisplay`	Specifies how the last caller is shown. Must be one of `Off`, `ID`, `Call` or `Both`
`callColor`	Specifies the call color. Must be one of `White`, `Black`, `Orange`, `Red`, `Yellow`, `Green`, `Turquoise`, `Blue`. Please note, that not all devices allow for all colors. Especially the AT-D868UV has only limited colors.
`standbyTextColor`, `standbyBackgroundColor`	Specifies the text or background color in stand-by. Must be one of `White`, `Black`, `Orange`, `Red`, `Yellow`, `Green`, `Turquoise`, `Blue`. Please note, that not all devices allow for

	all colors. Applicable to AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`channelNameColor`, `channelBNameColor`	Specifies the color of the channel name. Must be one of `White`, `Black`, `Orange`, `Red`, `Yellow`, `Green`, `Turquoise`, `Blue`. Please note, that not all devices allow for all colors. Applicable to AT-D878UV, AT-D878UV II and AT-D578UV.
`zoneNameColor`, `zoneBNameColor`	Specifies the color of the channel name. Must be one of `White`, `Black`, `Orange`, `Red`, `Yellow`, `Green`, `Turquoise`, `Blue`. Please note, that not all devices allow for all colors. Applicable to AT-D878UV, AT-D878UV II and AT-D578UV.
`language`	Specifies the UI language. Must be one of `English` or `German`. Applicable to AT-D878UV, AT-D878UV II, AT-D578UV and DMR-6X2UV.
`dateFormat`	Specifies the date format. Must be one of `YearFirst` (i.e., yyyy/mm/dd) or `DayFirst` (i.e., dd/mm/yyyy). Obviously, they got everything covered.

Audio settings

This entry collects all audio-related settings. These settings are collected under the `audioSettings` key within the `anytoneSettings` entry.

Audio settings extension fields

`voxDelay`	An interval specifying the delay to activate the VOX. Stored in ms resolution.
`voxSource`	Specifies the source for the VOX. Must be one of `Internal`, `External` or `Both`.
`recording`	If `true`, recording is enabled.
`enhance`	If `true`, enabled some TX audio enhancement. I don't know what it does nor if it is reasonable to enable.
`muteDelay`	Specifies some muting delay in minutes.
`maxVolume`, `maxHeadPhoneVolume`	Specifies the maximum speaker and headphone volume.
`enableFMMicGain`, `fmMicGain`	Enables and sets a separate microphone gain for FM. If disabled, the global mic gain is used for both, DMR and FM.

Menu settings

This entry collects all menu-related settings (there aren't many). These settings are collected under the `menuSettings` key within the `anytoneSettings` entry.

Menu settings extension fields

`duration`	Specifies the duration, the menu is shown.
`separator`	If `true`, the menu items are separated by a line. Again WTF AnyTone? Why the hack is an option needed for this? For me, this is just another useless option to check on every firmware update.

Auto-Repeater settings

The auto-repeater settings of anytone-devices is quiet complex. It allows to specify rules, in which frequency ranges some auto-repeater settings are applied referencing a particular offset for that rule. These offsets are specified within a separate list of frequency offsets.

These settings are needlessly complicated. It would be sufficient to specify a range and offset. AnyTone, however, decided that this would be way to simple and user friendly. However, fitting their concept into qdmr's concept of references to objects, makes it even worse. You will see.

The auto-repeater settings are specified within the `autoRepeaterSettings` item within the `anytoneSettings` item.

Auto-repeater extension fields

`directionA`, `directionB`
: Specifies the direction of the repeater offset for VFO A and B. I have absolutely no clue, why this depends on the VFO and not on the BAND (VHF/UHF) or even per offset. Only the wise people at AnyTone will know the answer.

  However, must be one of `Off`, `Positive` or `Negative`.

`vhfMin`, `vhfMax`, `uhfMin`, `uhfMax`, `vhf2Min`, `vhf2Max`, `uhf2Min`, `uhf2Max`
: Specifies the VHF and UHF frequency range for which the auto-repeater feature is defined. These can be any frequencies by the way.

  The `vhf2` and `uhf2` variants are only present for AT-D878UV, AT-D878UV II and AT-D578UV radios and allow for a second pair of frequency ranges with a different offset.

`vhf`, `uhf`, `vhf2`, `uhf2`
: References the offset for the VHF and UHF frequency range specified above. This must be a reference to one of the auto-repeater offsets defined below.

  The `vhf2` and `uhf2` references are only present for AT-D878UV, AT-D878UV II and AT-D578UV radios and allow for a different offset.

`offset`
: A list of auto-repeater offsets. As qdmr only allows references to named objects, they must be relatively complex. Each entry has an ID (to reference the offset), name (because qdmr objects have names) and the actual offset frequency. See below for a description of these elements.

Auto-repeater offsets

As mentioned above, qdmr requires any object referenced within the codeplug to be a proper object. That is with an ID and name. This makes sense, as almost all referenced elements within the codeplug are complex objects and have a display name. However, AnyTone decided to reference a mere offset frequency. Consequently, you have to give names to them in qdmr. Sorry.

The below you find a list with the properties of the offset frequencies.

Offset frequency fields

`id`
: Specifies the unique ID of the offset frequency. Used to reference it.

`name`
: Specifies the name of the offset frequency. Any string is valid and not needed. However, references are shown in qdmr with their name. So don't keep that field empty.

`offset`
: Specifies the actual frequency offset.

Example 4.15. An example for two auto-repeater frequency ranges active only on VFO A.

```
anytoneSettings:
  # ...
  autoRepeaterSettings:
    directionA: Negative   # <- Only on VFO A. Because.
    directionB: Off
    # VHF range
    vhfMin: 145.569 MHz
    vhfMax: 145.794 MHz
    vhf: offsetVHF
    # UHF range
    uhfMin: 438.55 MHz
    uhfMax: 439.4375 MHz
    uhf: offsetUHF
    # Some generic but disabled ranges.
    vhf2Min: 136 MHz
    vhf2Max: 174 MHz
    vhf2: ~
    uhf2Min: 400 MHz
    uhf2Max: 480 MHz
    uhf2: ~
    # The offsets
    offset:
      - id: offsetVHF
        name: VHF
        offset: 600 kHz
      - id: offsetUHF
        name: UHF
        offset: 7.6 Mhz
  # ...
```

DMR related settings

Some DMR related settings. Some of these should not be touched.

`groupCallHangTime`, `privateCallHangTime`, `manualGroupCallHangTime`, `manualPrivateCallHangTime`	Specifies the group and private call hang time. That is, the time, within you could answer a call by pressing the PTT, even if the group or private call is not the default transmit contact of the channel.
`preWaveDelay`, `wakeHeadPeriod`	Don't touch. Should both be 100ms.
`filterOwnID`	If `true`, your own ID is not shown in the call lists.
`smsFormat`	Specifies the SMS format. Must be one of `Motorola`, `Hytera`, `DMR`.
`monitorSlotMatch`	Specifies if and how the DMR monitor will match the time slot of the current channel. Must be one of `Off`, `Single`, `Both`.
`monitorColorCodeMatch`	If `true`, the DMR monitor will match the color code of the current channel.
`monitorIDMatch`	If `true`, the DMR monitor will match the DMR IDs to the group list of the current channel.
`monitorTimeSlotHold`	If `true`, the DMR monitor will hold the time slot first received on.
`sendTalkerAlias`	If enabled, sends the radio name as talker alias over the air.
`takerAliasSource`	Specifies the source for the talker alias display. Must be one of `Off`, `UserDB` or `Air`.
`talkerAliasEncoding`	Specifies the encoding of the taker alias. Must be one of `ISO8`, `ISO7` or `Unicode`.

GPS settings

This section collects settings concerning GPS position reporting. These settings are collected under the `gpsSettings` key within the `anytoneSettings` entry.

`units`	Specifies the unit system to use for distance reporting (ranging). This must be one of `Metric` or `Archaic`. This software was developed by a European, so don't mail me to tell me, that I am silly.
`timeZone`	Specifies the time-zone for the device. Any IANA time-zone ID is valid here. E.g., `Europe/Paris` or `UTC+5`.
`reportPosition`	If `true`, GPS position reporting is enabled.
`updatePeriod`	Specifies the update/reporting interval in seconds. Any interval specification is valid here. E.g., `5 min` or `360 s`.
`mode`	Specifies the GPS mode. Must be one of `GPS`, `Beidou`, `GPS_Beidou`, `Glonass`, `GPS_Glonass`, `Beidou_Glonass` or `All`.

Roaming settings

This section collects settings concerning the roaming. These settings are collected under the `roamingSettings` key within the `anytoneSettings` entry.

`autoRoam`	If `true`, the auto-roaming is enabled. If `false`, the roaming must be started manually.
`autoRoamPeriod`	Specifies the auto-roaming period in minutes. Any interval specification is valid here. E.g., `30 min` or `1 h`.
`autoRoamDelay`	Specifies some additional delay in seconds before the auto-roaming is started. Any interval specification is valid here like `60 s` or `1 min`.
`roamStart`,`roamReturn`	Specifies the roaming start/end condition. Must be one of `Periodic` or `OutOfRange`.
`rangeCheck`	If `true`, the periodic range check is enabled.
`checkInterval`	Specifies the range check period in seconds. Any interval specification is valid here. E.g., `3 min` or `180 s`.
`retryCount`	Number of retries to connect to a repeater before giving up and starting the roaming.
`outOfRangeAlert`	Specifies the alert type for the out-of-range notification. Must be one of `Off`, `Bell` or `Voice`
`notification`	If `true`, the repeater check notification is enabled.
`notificationCount`	The number of repeater check notifications.
`defaultZone`	Specifies the default roaming zone. Must be a roaming zone ID or `null`.
`gpsRoaming`	If `true`, GPS based roaming is enabled.

Bluetooth settings

This section collects settings concerning bluetooth feature. These settings are collected under the `bluetoothSettings` key within the `anytoneSettings` entry.

`pttLatch` If `true`, the bluetooth PTT button latches.

`pttSleepTimer` Specifies the timeout for the sleep mode of the bluetooth PTT button.

Simplex repeater settings

Collects all settings related to the DMR-6X2UV simplex-repeater feature. These settings are collected under the `simplexRepeaterSettings` key within the `anytoneSettings` entry. This extension is only relevant for the DMR-6X2UV series.

`enabled` If `true`, the simplex repeater function is enabled.

`monitor` If `true`, the traffic can be monitored on the speaker.

`timeSlot` Specifies the time-slot of the repeater. This must be one of `TS1`, `TS2` or `Channel`. The latter implies, that the repeater uses the time-slot of the channel.

FM APRS settings extension

This extensions allows to specify some additional settings for the FM APRS System. As AnyTone devices allow only for one FM APRS system, these settings can be considered radio wide.

Example 4.16.

```
positioning:
  ...
  - aprs:
    ...
    anytone:
      txDelay: 60 ms
      preWaveDelay: 0
      passAll: false
      reportPosition: false
      reportMicE: false
      reportObject: false
      reportItem: false
      reportMessage: false
      reportWeather: false
      reportNMEA: false
      reportStatus: false
      reportOther: false
      frequencies:
        - id: af1
          name: APRS US
          frequency: 144.39 MHz
```

The AnyTone™ FM APRS extension is a mapping named `anytone`. It contains the device specific settings for that FM APRS system. AnyTone devices only allow for a single FM APRS system.

FM APRS system attributes

FM APRS system extension fields

`txDelay` A delay before the start of the transmission. Specified with a resolution of 20ms. The default value is 60ms.

`preWaveDelay` A delay between the start of the carrier and the actual APRS transmission. Default 0ms.

`passAll` Basically disables the CRC check on received APRS messages.

`reportPositionreportMicEreportObject reportItemreportMessagereportWeather reportN-` A bunch of flags indicating what is being reported (?!?).

`MEAreportStatusrepor-tOther`

`frequencies` While AnyTone only allows for a single set of parameters for the FM APRS transmission and reception, it allows for several frequencies. The default frequency is specified by the revert channel of the FM APRS system.

Seven additional frequencies might be specified through this list. Each entry is a mapping, specifying a unique ID for the frequency, a name and the actual frequency. The id can then be used to reference this FM APRS frequency in the AnyTone channel extension.

SMS Extension

This extension allows for configuring the SMS (text message) feature of all DMR radios. For now, it is implemented as an extension, but might be changed into a common core setting in the future.

This extension only has two elements: the SMS format and a list of predefined text messages. The latter is important for devices that do not have a keypad and thus do not allow for entering any messages. E.g., the Radioddity GD-73.

As this extension configures a common feature, the extension is always present.

Example 4.17. Example of a single SMS pre-defined text or template.

```
sms:
  format: Motorola
  templates:
    - id: sms1
      name: RSSI
      message: RSSI
```

Common SMS settings

`format` Specifies the SMS format to be used. This must be one of `Motorola`, `Hytera` or `DMR`. If an incompatible format is chosen, you cannot receive or send text messages.

Message templates

The `templates` key defines a list of SMS templates to be programmed onto the radio. Each template is a config object with a name and ID. The latter can be used to reference the template somewhere else in the codeplug.

Each template has the following properties:

`id` Specifies the unique ID of the message. This can be used to reference the message.

`name` Specifies the display name of the message. Please note, that not all radios store descriptive names alongside with the preset messages.

`message` The actual preset message. The length might differ but is usually limited to 144 chars.

Chapter 5. Commercial Codeplug Extensions

This section collects all extensions that configure commercial features present in all radios.

DMR was developed as a digital replacement of the analog trunked radio systems used in commercial context. To this end, there are many features of that are irrelevant or outright forbidden in an amateur radio context. However, some HAM operators use their hand-held radios also for their day job and thus may need to configure some features that are only applicable in a commercial context. To this end, qdmr allows to set some of the commercial features through extensions.

Warning

Some commercial features defined within the DMR standard are illegal to use in an amateur radio context. Check your local regulations!

Example 5.1. Example commercial extension defining an encryption key.

```
commercial:
  encryptionKeys:
    - aes:
      id: key1
      name: Example key 1
      key: 01234567890123456789012345678901
```

All commercial extensions are held under the `commercial` key within the codeplug. For now, there is only one extension, the encryption extension listing the encryption keys described below.

Channel extension

This extension allows to specify some channel settings specific for commercial applications of DMR. For now, it only allow for specifying the encryption key for a digital channel.

Example 5.2.

```
channels:
  - digital:
      id: ch1
      name: Example channel
      # all the other channel settings
      commercial:
        encryptionKey: KEY_LABEL
```

The commercial channel extension is a mapping named `commercial`. It contains the settings for commercial features for that channel.

Channel attributes

For now, there is only one attribute specifying the encryption key associated with the channel.

Channel extension fields

`encryptionKey` References a encryption key defined within the commercial codeplug extension. See the section called “Encryption extension”.

Encryption extension

This extension allows to configure the commercial encryption features of DMR. Almost all DMR radios implement means for encrypting the traffic. This feature however, is usually illegal when used within amateur radio.

Example 5.3.

```
commercial:
  encryptionKeys:
    - dmr:
      id: key1
      name: Example key 1
      key: ABCD
    - rc4:
      id: key2
      name: Example key 2
      key: 0123456789
    - aes:
      id: key3
      name: Example key 3
      key: 0123456789ABCDEF0123456789ABCDEF
```

This extension is a simple list of keys held in the global `commercial` extension. Each key must either be a DMR (basic), RC4 (enhanced) or AES (advanced) key. To differentiate these key types, each list entry must be a mapping with a single entry. The name specifies the type (i.e., either `dmr`, `rc4` or `aes`). The value then specifies the properties of the key.

Common key attributes

As only the key size differs between the different key types, there are common attributes. Specifically, the ID and name of the key.

Common key fields

`id` Specifies the ID of the key. This ID can then be used to reference the key within the commercial digital channel extension. See the section called “Channel extension”.

`name` Specifies the name of the key. This property is usually not encoded in the binary codeplug.

DMR (basic) key attributes

DMR key fields

`key` Specifies the key as a HEX string. It must be at least 8bit but can be of variable size. The actual size depends on the device. Usually, a fixed size of 16 or 32bit is supported. Some devices, however, support variable sized keys.

RC4 (enhanced) key attributes

RC4 key fields

`key` Specifies the key as a HEX string. This key is fixed to a size of 40bit. That is, the hex string must be of length 10.

AES (advanced) key attributes

AES key fields

`key` Specifies the key as a HEX string. Also this key can be of variable size. Usually, these keys are 128 or 256bit. The actual size depends on the device.

Chapter 6. Table Based Codeplug Format

Warning

This chapter describes the old and deprecated table based `conf` text-file format. This format is already incomplete and will remains so. This documentation is maintained for the sake of completeness but you should not use that format in future.

qdmr will still be able to import this format but it will not save codeplugs in that format anymore. The new codeplug format is documented in Chapter 3, *Extensible Codeplug File Format.*

This configuration file format represents a generic configuration for a wide variety of radios. It is a simple text file containing simple key-value definitions like the DMR ID as well as tables like the table of channels, contacts, etc.

The aim of this config format is to be human-readable and writable. This would allow users to write config file by hand and share them easily, as well as enable users to modify shared configurations using a text editor. To this end, the format must be intuitive and to some degree self-documenting.

Within the following sections, I will describe that text format in some detail.

Line comments

To document your configuration, you may use so-called line-comments. These comments start with the character # and end at the end-of-line.

```
# A comment, everything in this line is ignored
ID: 12345678   # Another comment
```

General configuration

The general configuration settings of some radios can be overly complex with a huge amount of options. The vast majority of these settings, however, are useless for ham-radio purposes. Thus the possible settings for the general configuration of the radio are reduced to 6 key-value pairs.

The DMR ID of cause, is absolutely necessary and specifies your personal DMR number. Keep in mind, that you do NOT need to get a unique DMR ID for each radio you own! All your radios can share the same DMR ID. The DMR ID is specified using the "ID" keyword as

```
ID: 12345678
```

In rare situations, where you actually need several different radio IDs (e.g., if you use the same radio for HAM and commercial applications), you may specify them as a comma separated list. The first ID in the list will be handled as the default ID for the radio.

The radio name is a string, that the radio may display somewhere on the screen. It does not have any effect on the behavior of the radio or gets transmitted. You may set this entry to your call-sign. For example:

```
Name: "DM3MAT"
```

The two intro lines might be shown on the screen of your radio on startup. You may set these to any string you like. They are also cosmetic and don't have any effect on the behavior of your radio. For example

```
IntroLine1: "Hello"
IntroLine2: "MY0CALL"
```

The microphone sensitivity/amplification can also be set (on some radios) using the MicLevel entry. This entry is a number between 1 and 10. The larger the level, the larger the microphone amplification. This value may vary heavily from model to model.

```
MicLevel: 2
```

The "Speech" option enables the speech synthesis of the radio if supported. Possible settings are "on" and "off".

```
# Speech-synthesis ('On' or 'Off'):
Speech: Off
```

Contact table

The contact table is a list of DMR contacts like

```
Contact Name             Type    ID       RxTone
1       "DM3MAT"         Private 2621370  +
2       "DMR All Call"   All     16777215 -
3       "Simplex TG99"   Group   99       -
4       "Global"         Group   91       -
5       "EU"             Group   92       -
6       "Regional"       Group   8        -
7       "Local"          Group   9        -
8       "Deutschland"    Group   262      -
9       "Berlin/Brand"   Group   2621     -
10      "Berlin City"    Group   26212    -
11      "Brandenburg"    Group   26209    -
12      "Sachs/Thu"      Group   2629     -
13      "R Brandenburg"  Private 4044     -
```

These contacts can be personal contacts like DM3MAT, so-called all-calls and group calls. The contact table starts with the "Contact" keyword and ends with an empty line. The remaining keywords ("Name", "Type", "ID", "RxTone") are ignored, however, they are part of the self-documentation of the config file.

Following the "Contact" keyword, each line represents a single contact in the contact list. The first column represents a unique internal ID for the contact. It must not necessarily be in ascending order, any unique number will do. The second column is the name of the contact. Any string can be used here. The third column specifies the type of the contact. This must be one of the keywords "Private", "Group" or "All", meaning private, group or all-calls, respectively. The fourth column specifies the DMR ID for the contact. Please note, that an all-call requires the specific DMR ID 16777215 to work as an all-call. The last column specifies, whether an incoming call from this contact will cause a ringtone. Here "+" means enabled/yes and "-" disabled/no.

Group list table

Group lists are simple named lists of one or more contacts. These lists may include group, all or even private calls. Group lists are assigned to channels. They form a group of contacts (e.g., talk groups)

you may want to listen to on a particular channel. Usually these group lists form a collection of talk groups that are specific for a particular region.

Group lists are defined within the config file like

```
Grouplist Name            Contacts
1         "Local"          7,9
2         "Deutschland"    8
3         "Global"         4
4         "EU"             5
5         "Sachs/Thu"      13,12
6         "Simplex"        2,3,6,7
7         "Berlin/Brand"   9,10,11,13
```

The group list table starts with the keyword "Grouplist". The following keywords (Name & Contacts) are ignored, but form a kind of self-documentation for the config file.

Following the "Grouplist" keyword, each group list is defined by a single line. The first column specifies the internal unique ID for the group list. This can be any number as long as it is unique. The second column contains the name of the group list as a string. This can be any non-empty string. The third column contains the comma-separated list of contact IDs that form that group list.

Digital channel table

The digital channel table defines all digital DMR channels. As digital channels have some different options compared to analog channels, they are not defined within the same table. However, they share the same IDs. So be careful not to assign the same identifier to analog and digital channels.

The digital channel table has the form

```
Digital Name             Receive   Transmit  Power Scan TOT RO Admit  CC TS RxGL TxC GPS Roam ID
11      "DM0TT Ref"      439.0870  -7.6000   High  1    -   -  Free   1  1  7    12  -   +    -
12      "DM0TT BB"       439.0875  -7.6000   High  -    -   -  Free   1  2  7    15  1   +    -
84      "DMR_S0"         433.4500  433.4500  High  2    -   -  Free   1  1  6    9   -   +    -
85      "DMR_S1"         433.6120  433.6120  High  2    -   -  Free   1  1  6    9   -   +    -
86      "DMR_S2"         433.6250  433.6250  High  2    -   -  Free   1  1  6    9   -   +    -
87      "DMR_S3"         433.6380  433.6380  High  2    -   -  Free   1  1  6    9   -   +    -
88      "DMR_S4"         433.6500  433.6500  High  2    -   -  Free   1  1  6    9   -   +    -
89      "DMR_S5"         433.6630  433.6630  High  2    -   -  Free   1  1  6    9   -   +    -
90      "DMR_S6"         433.6750  433.6750  High  2    -   -  Free   1  1  6    9   -   +    -
91      "DMR_S7"         433.6880  433.6880  High  2    -   -  Free   1  1  6    9   -   +    -
```

The digital-channel table starts with the keyword "Digital" and ends with an empty line. The next keywords (Name, Receive, Transmit, Power, Scan, TOT, RO, Admit, CC, TS, RxGL and TxC, GPS, Roam, ID) are ignored and are maintained for the self-documentation of the configuration file.

Each channel is defined within a single line. The first column is the unique channel identifier (any unique number among analog AND digital channels). The second column specifies the channel name as a string.

The third column specifies the RX frequency in MHz and the fourth column the TX frequency in MHz. Alternatively, a TX frequency can also be specified in terms of an offset relative to the RX frequency. In this case, the offset must be prefixed with either "+" or "-".

The 5th (Power) column specifies the power level to use. Here, either the "High" or "Low" keyword must be used. The 6th (Scan) column specifies the ID of the scan-list (see below) attached to the channel. This list will be used whenever a scan is started on this channel.

The 7th column (TOT) column specifies the TX time-out-timer in seconds or "-", if disabled. The 8th column (RO) specifies whether the channel is RX only ("+") or not ("-"). If enabled, you cannot transmit on that particular channel.

The 9th (Admit) column specifies the TX admit criterion for the channel. This must be either "-" or one of the keywords "Free" and "Color". "-" indicates that there is no restriction in transmitting on that channel. The radio will transmit whenever PTT is pressed. The "Free" keyword indicates that the

radio will only transmit if the channel is free. The "Color" keyword indicates that the radio will only transmit if the channel is free and the color-code of the repeater matches the specified color-code of the channel (see next column).

The 10th (CC) column specifies the color-code of the channel. To avoid interference between neighboring radios and repeaters on the same frequency (in case of DX conditions), the repeater and radio will only react to transmissions on a channel with the matching color-code. The color-code can be any number between 0 and 15.

The 11th (TS) column specifies the time-slot for this channel. Due to the audio compression used in DMR, it is possible to operate two independent channels on a single frequency by using time-slicing. DMR uses two time-slots. This option specifies which of the two time-slots is used for the channel. On simplex channels, this time-slicing is irrelevant, as there is no central instance (the repeater) that defines what time-slot 1 or 2 is.

The 12th (GPS) column specifies the GPS or APRS system (see below) to use on that channel. The 13th (Roam) column specifies the roaming zone. This can either be '-' meaning *roaming disabled* or an ID of a roaming zone specified below. Finally, the 14th column (ID) specifies the DMR ID to use with this channel. That is either '-' for *default ID* or an index (0-based) of the ID list above.

Analog channel table

The analog channel table collects all analog (FM) channels. As digital channels have some different options compared to analog channels, they are not defined within the same table. However, they share the same IDs. So be careful not to assign the same identifier to analog and digital channels.

The analog channel table has the form

```
Analog  Name            Receive   Transmit Power Scan TOT RO Admit  Squelch RxTone TxTone Width APRS
1       "Y07"           144.6750  144.6750  High  1    -   -  Free   1       -      -      12.5  -
2       "S20"           145.3000  145.3000  High  -    -   -  Free   1       -      -      12.5  -
3       "Mobil 2m"      145.5000  145.5000  High  -    -   -  Free   1       -      -      12.5  2
4       "DB0RAG"        439.3000  -7.6000   High  1    -   -  Free   3       -      -      12.5  -
5       "DB0LOS"        438.7750  -7.6000   High  1    -   -  Free   1       -      -      12.5  -
6       "DB0LUD"        438.5750  -7.6000   High  1    -   -  Free   1       -      67     12.5  -
19      "DB0BLO"        439.2750  -7.6000   High  1    -   -  Free   3       -      -      12.5  -
20      "DB0SP-2"       145.6000  -0.6000   High  1    -   -  Free   3       -      -      12.5  -
21      "DB0SP-70"      439.4250  -7.6000   High  1    -   -  Free   3       -      -      12.5  -
```

The analog channel table starts with the "Analog" keyword and ends with an empty line. The remaining keywords right after "Analog" (i.e., "Name", "Receive", "Transmit", "Power", "Scan", "TOT", "RO", "Admit", "Squelch", "RxTone", "TxTone" and "Width") are ignored but are part of the self-documentation of the config file.

Each line within the table specifies a single channel. The first column specifies the unique ID of the channel. This ID can by any number that is unique among analog AND digital channels. The second (Name) column specifies the name of the channel as a string. Any string can be used here.

The third (Receive) column specifies the RX frequency of the channel in MHz. The fourth (Transmit) column specifies the TX frequency in MHz or alternatively, an offset relative to the receive frequency in MHz by prefixing "+" or "-".

The 5th (Power) column specifies the transmit power. This must be either the "High" or "Low" keyword. The 6th (Scan) column specifies the scan-list ID for this channel or "-" if there is no scan-list assigned to the channel. A scan-list (see below) is just a collection of channels that gets scanned whenever scanning is started on a particular channel.

The 7th (TOT) column specifies the transmit time-out in seconds or "-" if disabled. The 8th (RO) column specifies whether this channel is receive-only with either "-" meaning disabled and "+" enabled. If enabled, it is impossible to transmit on that channel.

The 9th column specifies the admit criterion on that channel. This must be either "-" meaning that there is no restriction when to send on that channel, the keyword "Free" meaning that the channel

must be free to transmit or the keyword "Tone" meaning that the channel must be free and the RxTone must match.

The 10th (Squelch) column specifies the squelch level for the channel. This must be a number between [0-10]. The larger the value, the stronger the signal must be to open the squelch. The value 0 disables the squelch.

The 11th (RxTone) specifies the receive CTCSS tone frequency in Hz. The squelch will then only open, if the signal is strong enough (see previous column) and the specified tone is received. If set "-" the RX tone is disabled and the squelch will open if the signal is strong enough. The 12th (TxTone) column specifies the CTCSS tone to transmit in Hz or "-" if disabled. This feature is used by some repeaters to open their squelch and to start repeating to avoid conflicts between repeaters operating on the same frequency (e.g., in case of DX conditions).

The 13th (Width) column specifies the bandwidth of the channel in kHz. This can be 12.5kHz narrow-band or 25kHz wide-band. Finally, the 14th column specifies the APRS system ID to use or "-" for *APRS disabled*.

Zone lists

Zones are just collections of channels. Typical radios can hold thousands of channels. To keep large numbers of channels manageable, they can be organized into zones. Usually, these zones represent a geographical area and all repeaters in that area are then grouped into zones. Of cause, a single channel can be added to multiple zones. Please note that for many radios, channels can only be accessed via a zone. That means, a channel that is not a member of any zone may not be accessible.

The zone table is defined within the configuration file as

```
Zone    Name            VFO Channels
1       "KW"            A   1,9,11,12,14,8,55,15,4,5,6,20,21,22,19,48
1       "KW"            B   1,3,2,81,82,84,85,86,87,88,89,90,91
2       "Berlin DMR"    A   10,9,11,12,34,35,31,32,33,27,28,29,30,38,39
2       "Berlin DMR"    B   1,3,2,81,82,84,85,86,87,88,89,90,91
3       "Berlin FM"     A   20,21,19,18,22,23,24,25
4       "Potsdam"       A   42,43,44,45,46,47,40,41
7       "Leipzig"       A   75,76,72,71,73,70,74,77,78,80,79,69
8       "Simplex"       A   1,3,2,81,82,84,85,86,87,88,89,90,91
```

The zone table starts with the keyword "Zone" and ends with an empty line. The remaining keywords (Name and Channels) are ignored but are part of the self-documentation of the configuration file. The first column specifies an unique identifier for each zone. This can be any integer as log as it is unique. The second (Name) column specifies the name of the zone as a string. Any string is valid here. The third column specifies the VFO (either A or B) for that zone. This allows to specify different channels for the two VFOs of the radio. For example, it allows to specify a list of repeater channels for VFO A and some simplex and calling frequencies on VFO B. The fourth column contains the comma-separated list of channel IDs for the zone anc VFO. A reference to any channel-type can be used here, analog and digital.

Scan lists

A scan list is list of channels, that are scanned whenever scanning is started on a channel, the scan list is associated with. A single scan list might be associated with several channels. For example, all channels within that scan list.

The list of scan lists has the following form

```
Scanlist Name            PCh1 PCh2 TxCh Channels
1        "KW"            1    84   Sel  1,84,2,9,11,8,14,4,5,20,21,19,6
2        "DMR Simplex"   84   -    Sel  84,85,86,87,88,89,90,91
```

The list of scan lists starts with the "Scanlist" keyword and ends with an empty line. The remaining keywords (Name, PCh1, PCh2 & Channels) are ignored but part of the self-documentation of the

configuration file format. A scan list is defined with every other line. The first column specifies the unique identifier of the scan list. The second (Name) column specifies the name of the scan list as a string. Any string will do. The third and fourth columns specify the first and second priority channels for the scan list respectively. These priority channels are visited more frequently during the scan. That is, the first priority channel is visited 50% of the time while the second is visited 25% of the time. These channels might also be set to "-" indicating that there is no priority channel. The 5th column specifies the transmit channel during the scan. Possible options are "Last", "Sel" and any valid channel index. The "Sel" keyword implies that the radio will transmit on the selected channel when the scan started. The "Last" keyword implies that the radio will transmit on the channel at which the scan stopped on, while specifying any channel index implies, that the radio will transmit on that channel. Finally the 6th column specifies the comma-separated list of channels that form the scan list.

GPS Systems

The GPS system list just specifies the contact to which some positional information is sent to (which usually gets forwarded to the APRS system) and at which period this information is sent.

```
GPS  Name               Dest Period Revert
1    "BM APRS"          20   300    -
```

The first column specifies the ID of the GPS system. This can be any number >0. The second column (Name) specifies the name of the GPS system. The third column specifies the destination contact ID (see Contacts above), the position information is sent to. The fourth column (Period) specifies the update period in seconds. The fifth column (Revert) specifies the revert channel. In amateur radio, this can be left blank ("-").

APRS Systems

The APRS system list specifies the various information for transmitting your position using analog APRS. As digital channels may use either DMR or analog APRS for position reporting, this list shares a namespace with the GPS system list. That is, the ID must be unique across both lists.

```
APRS Name               Channel Period Source      Destination Path                Icon        Message
2    "APRS APAT81"      103     300    DM3MAT-7    APAT81-0    "WIDE1-1WIDE2-1"    "Jogger"    "Y07"
```

The first column specifies the ID of the APRS positioning system. This must be unique across APRS and DMR position reporting systems. The second column specifies the name of the system as a string. the third column specifies the revert channel. That is, the analog channel the APRS information is transmitted on. The 4th column specifies the period with which the position gets reported. The 5th and 6th columns specify the source and destination calls and SSIDs respectively. The 7th column specifies the path string. This is list of calls and SSIDs stored as a string without any separators. The 8th column specifies the map icon name [http://www.aprs.org/symbols/symbols-new.txt]. The name does not need to match the official icon name exactly. The icon is identified as the closes matching icon name with respect to the Levenshtein distance [https://en.wikipedia.org/wiki/Levenshtein_distance] between the given and all icon names. That is, *jogger* and *jogging* will select the same icon. Finally, the 9th column specifies a freely selectable text to be sent with the position report.

Roaming Zones

Roaming zones allow to stay in contact with a particular talk group when moving round and the current repeater gets out of range. In this case, the radio will search for the strongest repeater in a list (the so-called *roaming zone*) and switch to this repeater.

```
Roaming Name               Channels
1       "Berlin/Brand"     3,7,13,17,19,23,25,29,33,37,41,49
```

Therefore, a roaming zone is a simple channel list. The first column specifies the ID of the zone. This ID can be used in the digital channel table to associate a channel with a specific roaming zone. The

second column specifies the name of the zone and the third column holds the comma-separated list of channel in each zone.

Chapter 7. The dmrconf command line tool

Beside the graphical user interface provided by qdmr (see Chapter 2, *The Graphical User Interface*), there is also a command line tool allowing to read/write codeplugs from and to the radios. It is based on the same library called libdmrconf [https://dm3mat.darc.de/qdmr/libdmrconf] and thus provides the same features like qdmr.

This chapter will briefly describe the command line tool and how it can be used to handle codeplugs from the command line.

The command line tool might be helpful in cases, where the codeplug file (see Chapter 3, *Extensible Codeplug File Format*) gets assembled by a script. Then the same script may upload the codeplug to the radio using the command line tool.

Additionally to the the features of the GUI (see Chapter 2, *The Graphical User Interface*), the command line tool provides some features to analyze the memory representation of the binary codeplugs as well as debugging their implementation.

Reading and writing codeplugs

The major feature of the command line tool is certainly the ability to read and write codeplugs from and to the device. The majority of the action happens automatically. Like the detection of the radio. If something goes wrong, an error message will be written to `stderr`. A more detailed logging can be enabled by passing the `--verbose` flag.

Reading a codeplug

To read a codeplug, the **read** command is used. The codeplug can be stored in several formats. The human readable extensible codeplug format (YAML) and as a binary memory dump of the codeplug memory on the device. **dmrconf** detects the format based on the file extension or by means of an additional flag. The latter is particularly important if the read codeplug should be written to `stdout` for piping it to another program for further processing.

```
dmrconf read codeplug.yaml
```

Will simply read the codeplug from the detected device and stores it in the extensible codeplug format in the file `codeplug.yaml`. The format was detected by the file extension `yaml` which refers to the extensible codeplug format using YAML.

To store the memory dump of the codeplug memory of the radio, the file extension should be `dfu`.

As mentioned above, it is also possible to dump the decoded codeplug to `stdout` allowing to pipe the codeplug into another program for processing. This can be done by omitting the output filename. Then, however, the output format is not specified anymore. In this case, one the explicit format flags `--yaml` or `--bin` must be used to specify in which format the codeplug should be written. These flags can also be used to store a codeplug in a particular format in arbitrarily named files.

```
dmrconf read --yaml | python my_script.py
```

This example reads the codeplug from the connected device and decodes it. The decoded codeplug is then piped to the python script `my_script.py`.

Decoding binary codeplugs

It is also possible decode binary codeplugs that has been read from the device earlier and stored as a memory dump (i.e., in a `dfu` file). This step is actually the second step automatically performed during reading. When reading a codeplug, in a first step the memory dump of the codeplug is read from the device. In a second step, the read binary codeplug is then decoded and dumped in a human readable format.

The **decode** command performs that second step. To do that, it needs two additional information: The radio type, from which this codeplug was read and the format to write the decoded format to. Like for the **read** command, the latter can be passed by the output filename extension or via an additional flag.

```
dmrconf read codeplug.dfu
dmrconf decode --radio=uv390 codeplug.dfu codeplug.yaml
```

This example performs the same actions like a simple **read** command (assuming a TyT MD-UV390 is connected). It first downloads the binary codeplug. This time, the memory dump is stored in a binary form (`dfu` file). The second command then decodes the binary codeplug into the extensible codeplug format (`yaml` file), assuming that the binary codeplug stems from a TyT MD-UV390.

Since version 0.8.1, it is now also possible to decode some manufacturer binary codeplug files as they are produced by the manufacturer CPS. To signal the decode command to treat the file as a manufacturer CPS file, you need to pass the `-m` or `--manufacturer` and the `--radio` option. The latter tells the **decode** command the format of the file. That is, the call

```
dmrconf decode -m --radio=uv390 manufacturer_cps_file.rdt codeplug.yaml
```

Will decode the manufacturer CPS file `manufacturer_cps_file.rdt` assuming it is a file generated by the CPS for the TyT MD-UV390. Like for the “normal” decoding the output format must be specified either by file extension or flag.

Debugging the codplug decoding

Under normal circumstances, it makes no sense to first read the binary codeplug from the device and then decoding it in a separate step as the read command will do that for you.

However, if there is a bug in **dmrconf** that gets triggered by your codeplug on the device, the binary codeplug is an invaluable resource for debugging the application. Consider filing an issue at the bug tracker [https://github.com/hmatuschek/qdmr/issues] and include the binary codeplug as an attachment.

If you like, you can also send me your codeplug directly. I'll keep it confidentially.

Writing a codeplug

To write a codeplug into the device, the **write** command is used. The codeplug can be read from several formats. The extensible codeplug format (`yaml` file) as well as the old table based format (`conf` file). It is not possible to write binary codeplugs without decoding them first. Like for the **read** command, **dmrconf** will detect the format based on the file extension or by passed flags.

```
dmrconf write codeplug.yaml
```

This example will write the codeplug stored in the extensible codeplug format in `codeplug.yaml` to the connected device. Before writing the codeplug to the device, the connected device gets detected and the codeplug gets verified.

If the verification step fails, one or more error messages are written to `stderr` describing the issue with the codeplug. One verification step is the check whether all channel frequencies are within the frequency limits specified by the manufacturer. The latter check can be disabled using the `--ignore-frequency-limits` flag.

There are also some flags controlling the assembly of the binary codeplug. When the `--init-codeplug` flag is set, the codeplug will be generated from scratch using default values for all options not explicitly specified in the codeplug file. This might be used to initialize a brand new radio. However, any changes made to the radio are lost.

When this option is not set, the codeplug gets encoded and written in a two-step process. First the current binary codeplug is downloaded from the radio. Then the codeplug file is used to update the binary codeplug. The result is then written back to the device. This ensures that all settings made in the radio are kept, unless they are explicitly set in the codeplug file.

The `--auto-enable-gps` and `--auto-enable-roaming` flags will tell **dmrconf** to enable the GPS or roaming feature whenever any of the programmed channels use the GPS or a roaming zone. (This depends also on the ability of the radio.)

Verify a codeplug

The aforementioned verification of the codeplug file can also be performed separately using the **verify**. This command also needs to know against which radio the codeplug should be verified. The radio must be specified using the `--radio` option.

```
dmrconf verify --radio=d878uv codeplug.yaml
```

This command will verify the codeplug stored in `codeplug.yaml` in the extensible codeplug format against an AnyTone AT-D878UV. Like for the **write**, any issues are written to `stderr`.

Like for the **write** command, the verification can be altered using the `--ignore-frequency-limits` flag.

Encoding codeplugs

Is is also possible to perform the encoding step of the codeplug separately. This can be done with the **encode** command. Like for the **verify** command, the **encode** command also needs the radio for which the codeplug should be encoded. The input format of the codeplug is again specified by either the file extension of the codeplug file or by flags.

```
dmrconf encode --radio=opengd77 codeplug.yaml codeplug.dfu
```

This call will encode the codeplug `codeplug.yaml` specified in the extensible codeplug format for a radio running the OpenGD77 firmware and stores the resulting binary codeplug in `codeplug.dfu`.

Like for the **write** command, the encoding can be controlled using the `--auto-enable-gps` and `--auto-enable-roaming` flags.

Writing the call-sign DB

The command line interface also allows to write a call-sign DB (also known as *Talker Alias*) to the radio if the radio supports it. This can be done with the **write-db** command. This command behaves similar to the call-sign DB upload in qdmr. That is, it tries to select the call-signs being written automatically.

Although many radios provide a huge amount of memory for the call-sign DB, they cannot hold the entire list of assigned DMR IDs. Therefore, a selection of relevant call-signs must be done. qdmr and **dmrconf** do that based on the DMR ID of the radio. DMR IDs are not random. They follow a pattern similar to land-line phone numbers. DMR IDs within a certain continent, country and region share a common prefix. This way, it is possible to select DMR IDs that are *close* by selecting IDs that share a common prefix.

qdmr and **dmrconf** use the default DMR ID of the radio to select the call-signs to be written. This ID is obtained by qdmr from the codeplug. **dmrconf** does not require a codeplug to be present for the upload. Consequently, it requires the explicit specification of the ID to base the selection on. The ID can be specified using the `--id` or `-I` options. It is also possible to just specify the prefix instead of an ID.

```
dmrconf write-db --id=2621
```

This example writes a call-sign DB to the connected radio using the DMR prefix 2621 (for the region Berlin/Brandenburg in Germany). The radio will fill the available space with as many call-signs as possible. So for the previous example, **dmrconf** will not only program all IDs with the prefix 2621 but as many as possible starting with those *close* to that prefix. It is possible to limit the number of call-signs encoded using the `--limit` or `-n` option.

Unfortunately, there is always an exception to a rule: Some countries have several prefixes or you may program the call-signs of several countries that are not necessarily *close* in terms of their prefixes. For these cases, it is possible to specify a list of prefixes to the `--id` option.

```
dmrconf write-db --id=262,263 --limit=10000
```

This example will select and write up to 10000 (given the radio can hold that amount) call-signs staring with those *closest* to the prefixes 262 and 263 (both prefixes for Germany).

Specifying own databases

With version 0.11.3, it is possible, to specify a user-crafted JSON database. Then, this file will be used for selecting the call signs for writing the call-sign DB.

```
dmrconf write-db --id=262 --limit=10000 --database=my_db.json
```

Here, up to 10000 call signs are written from the `my_db.json` JSON file starting with those IDs closest to the prefix 262. Specifying a user-curated call-sign DB, simply replaces the public one and thus all methods described above still work.

Encoding a call-sign DB

Like for the binary codeplug, it is also possible to generate and store the binary representation of the call-sign DB using the **encode-db** command. This command is only useful for debugging purposes as the binary representation of the call-sign DB cannot be written to the device.

The **encode-db** takes the same arguments as the **write-db** command but additionally needs a filename to store the encoded DB into as well as the radio to encode for using the `--radio` option.

```
dmrconf encode-db --radio=d878uv --id=262,263 --limit=10000 callsigns.dfu
```

Like the previous example, this one encodes up to 10000 call-signs starting with those closest to the prefixes 262 and 263 for a AnyTone AT-D878UV radio and stores the result into `callsigns.dfu`.

Various features of dmrconf

Beside reading and writing codeplugs or writing the call-sign DB, there are some more commands and features that mainly concern the debugging of the codeplug and call-sign DB encoding and decoding. If you are interested in the codeplug internals, you may use these commands to study them.

Getting help

As usual for command line tools, a brief help text about the commands and options gets written to `stdout` when the `--help` or `-h` option is passed. No other commands passed are executed then.

Similar to the `--help` option, it is possible to print the version number of **dmrconf** using the `--version` or `-v` option. Like for the `--help`, no other commands passed get executed.

To get a list of keys identifying radio models when specified using the `--radio` option, `--list-radios` can be passed. This will print a small table to `stdout` that lists the keys for each known radio as well as the model and manufacturer name.

Detecting the radio type

The command **detect** solely detects the radio. No data is written or read from the device (except of the radio model information). This command can be used to check whether a radio is detected correctly.

```
dmrconf detect
```

This command will try to detect the connected radio. If a known radio is found, the model and manufacturer name is written to `stdout`. If no radio is detected or if the model is unknown or unsupported, an error message is written to `stderr`.

Inspecting binary codeplugs

The **encode**, **encode-db** and **read** commands can store the codeplug and call-sign DB in binary form in a DFU file. The generated file is a valid DFU (device firmware update) file, that can be handled with other DFU tools.

A DFU file may contains several so-called images. Each image may contains several so-called elements. The latter represents a segment of memory with an associated memory address.

The **info** command produces a hex-dump of the DFU file that is written to `stdout`. It can then be inspected using **more** or **less**.

```
dmrconf info codeplug.dfu | less
```

This example will generate a hex dump of the encoded codeplug in the specified DFU file `codeplug.dfu`. The result is piped to **less** for easy reading. The hex dump also prints some information about the file structure as well as memory addresses. It also collapses repetitive memory sections (similar to **hexdump -C**). To this end, this command is a helpful tool for debugging the encoding of codeplugs and call-sign DBs.

Danger zone

Warning

You are about to enter the land of pain. Continue on your own risk.

Some radios are actually identical to others. They also identify themselves as a different radio. An example for such a radio is the Retevis RT3S, this radio is simply a relabeled TyT MD-UV390. The RT3S actually identifies itself as a MD-UV390. From the perspective of the CPS, these two radios are indistinguishable. Consequently, qdmr and **dmrconf** will always identify the RT3S as a MD-UV390.

There are, however, virtually identical radios. These are radios that actually identify themselves as different models but the firmware, communication protocol and codeplug is basically identical to another model. An example for such virtually identical models are the AnyTone AT-D868UV and the BTech DMR-6X2. Each model identifies itself correspondingly and thus is distinguished by the CPS.

Some of these relationships between virtually identical models are known to qdmr and **dmrconf**. In these cases, the CPS will treat these radios as identical.

Some of these close relationships between models are not known to **dmrconf**. In these cases, **dmrconf** will stop with an error that a radio is unknown although it actually supported as a different model. In these rare cases, it is possible to override the radio detection using the `--radio` option.

This option is usually used to specify the type whenever the radio model is not detected. This option also overrides the model detection and thus allows to handle virtually identical radios. For example, if the relation ship between the AT-D868UV and the DMR-6X2 would have not been known to **dmrconf**, a codeplug could read anyway from the device by calling

```
dmrconf read --radio=d868uv codeplug_6x2.yaml
```

Here the radio detection (resulting the detection of a DMR-6X2) gets overridden and the radio is handled as a AT-D868UV.

If you know of such virtually identical radios that **dmrconf** does not recognize, consider filing an issue at the bug tracker [https://github.com/hmatuschek/qdmr/issues].

Warning

Of cause, handling a radio differently as it identifies itself may cause permanent damage to the radio. So you should be very sure that the radios are actually identical when overriding the radio detection routines.

Name

dmrconf — Command-line tool for programming DMR radios.

Synopsis

`dmrconf` [`OPTIONS`] [`COMMAND`] [file]

Description

dmrconf is a command-line tool to program DMR radios. That is, generating and uploading codeplugs to these radios. To this end, **dmrconf** uses a common human-readable text format to describe the codeplug for all supported radios (see below). This allows one to share codeplugs between different radios.

Additionally, **dmrconf** also allows one to download codeplugs from the radio and to store it in the human-readable text format.

Commands

detect Detects a connected radios. You may specify a specific device using the `-D` or `--device` option.

read Reads a codeplug from the radio and stores it into the given file. This command may need the `-y` or `-b` options if the file type cannot be inferred from the filename.

write Writes the specified codeplug to the radio. This command may need the `-c`, `-y` or `-b` options if the file type cannot be inferred from the filename.

write-db Writes the call-sign database to the device. This command may need the `--id` option to select call-signs if the complete database does not fit into the device. If specified, all call-signs closest to the specified ID are used.

verify Verifies the codeplug with the connected radio or the specified radio passed with the `--radio` option. This command may also need the `-y` or `-b` options if the file type cannot be inferred from the filename.

encode Encodes a YAML codeplug as a binary one for the connected or specified radio using the `--radio` option.

encode-db Encodes the call-sign database as a binary one for the connected or specified radio using the `--radio` option. This command may need the `--id` option to select call-signs if the complete database does not fit into the device. If specified, all call-signs closest to the specified ID are used.

decode Decodes a binary codeplug and stores the result in human-readable form. The radio must be specified using the `--radio` option.

info Prints some information about the given file.

Options

`-c` or `--csv` Specifies the file format for the input file for the **verify**, **encode** and **write** commands. This option is not needed if the filetype can be inferred from the filename. That is, if the file ends on `.conf` or `.csv`.

`-y` or `--yaml` Specifies the file format for the input or output file for the **verify**, **read** and **write** commands. This option is not needed if the

	filetype can be inferred from the filename. That is, if the file ends on `.yaml`.
`-b` or `--bin`	Specifies the file format for the input or output file for the **verify**, **read** and **write** commands. This option is not needed if the filetype can be inferred from the filename. That is, if the file ends on `.bin` or `.dfu`.
`-m` or `--manufacturer`	Specifies the file format for the input file for the **decode** command to be the manufacturer binary codeplug format. Not all manufacturer formats are implemented.
`-D` or `--device=DEVICE`	Specifies the device to use. Either a USB `BUS:DEVICE` number combination or the name of a serial interface. The device must be specified if the automatic radio detection fails or if more than one radio is connected to the host.
`-R` or `--radio=`NAME	Specifies the radio for the **verify**, **encode** or **decode** commands. This option can also be used to override the automatic radio detection for the **read** and **write** commands. Be careful using this option when writing to the device. An incompatible codeplug might be written.
`-I` or `--id=`DMR_ID	Specifies the DMR ID or a comma separated list of DMR ID prefixes for the **write-db** or **encode-db** commands. More than one ID may be specified using a comma-separator.
`-n` or `--limit=`N	Limits several amounts, depending on the context. When encoding or writing the call-sign db, this option specifies the maximum number of call-signs to encode.
`-B` or `--database=`JSON_FILE	Specifies the call-sign database to use for writing a user-db to the device.
`--init-codeplug`	Initializes the code-plug from scratch. If omitted (default) the codeplug on the device gets updated. This maintains all settings made earlier via the manufacturer CPS or on the radio itself.
`--auto-enable-gps`	Automatically enables GPS/APRS if at least one GPS/APRS system is defined and used by any channel.
`--auto-enable-roaming`	Automatically enables roaming if at least one roaming zone is defined and used by any channel.
`--ignore-limits`	Disables the enforcement of limits. Warnings are still shown.
`-h` or `--help`	Displays a short help message.
`--list-radios`	Lists all supported radios.
`-v` or `--version`	Displays the version number.
`-V` or `--verbose`	Enables debug messages.

Supported Radios

The following list contains all supported radios and their names for the `--radio` option.

`opengd77`	All radios running the Open GD77 firmware.
`d868uve`, `dmr6x2`	Anytone AT-D868UVE or Baofeng DMR-6X2.

`d878uv`	Anytone AT-D878UV.
`d878uv2`	Anytone AT-D878UVII.
`d578uv`	Anytone AT-D578UV.
`md390`, `rt8`	TYT MD-390 or Retevis RT8.
`uv390`, `rt3s`	TYT MD-UV390 or Retevis RT3S.
`md2017`, `rt82`	TYT MD-2017 or Retevis RT82.
`gd77`	Retevis GD-77.
`rd5r`	Baofeng/Radioddity RD-5R.
`dm1701`, `rt84`	Baofeng DM1701 or Retevis RT84.

Bugs

This program is still under development and may contain bugs that may cause harm to the radios and may even destroy them. Hence you may use this software on your own risk. If you want to have guaranties, consider using the CPS (code-plug programming software) supplied with your radio.

Name

qdmr — Graphical tool for programming DMR radios.

Synopsis

`qdmr` [`OPTIONS`] [`CODEPLUG_FILE`]

Description

qdmr is a graphical tool to program DMR radios. That is, reading, editing, generating and uploading codeplugs to these radios.

The configuration of these radios is usually stored in a highly vendor and device specific binary codeplug. **qdmr** stores this configuration in a human-readable and device independent format, thus allowing for sharing codeplugs across devices.

Options

`CODEPLUG_FILE`
: If the optional codeplug file is passed to qdmr, it gets loaded on start-up. Otherwise, the application starts with an empty codeplug.

`-style=STYLE`
: This option allows to set a Qt style for the application. This can be used to alter the style of the widgets. So call **`qdmr -style=windows`** if you fancy.

  However, this option is of real use if you want to use a dark theme. By installing a dark Qt theme like kvantum-dark and starting qdmr by passing this theme to the style option. That is

  ```
  qdmr -style=kvantum-dark
  ```

  You may set the Qt style for all Qt applications using the `QT_STYLE_OVERRIDE` environment variable.

`-stylesheet=FILENAME`
: This option allows for specifying a style sheet to alter the appearance of some or all widgets of qdmr.

Logging

Qdmr writes a lot of debug and error messages to `stderr`. This can be used to inspect some issues during reading/writing the a codeplug to the device. There is also a log file containing these messages usually at `~/.local/share/DM3MAT/qdmr/qdmr.log`.

Bugs

This program is still under development and may contain bugs that may cause harm to the radios and may even destroy them. Hence you may use this software on your own risk. If you want to have guaranties, consider using the CPS (code-plug programming software) supplied with your radio.

There are some issues with the so-called dark mode under Linux. The Qt library is not able to detect that the desktop has a dark mode theme. To this end, qdmr cannot react to it too. To enable a dark theme for qdmr, see the `-style` option above.

Writing a single application supporting several radios of different manufacturers is a hard task. To this end, there are plenty of bugs to be expected. If you stumble across one of them, consider opening an issue at https://github.com/hmatuschek/qdmr/issues.

Chapter 8. Reverse engineering

The majority of the development time needed for qdmr, consists of reverse engineering the code plug and communication protocols of the radios. Fortunately, many radios share the same communication protocol, in particular those from the same manufacturer. Sometimes these protocols are even used by other manufacturers. Even if the protocol is already implemented in qdmr, you may need to reverse engineer the code plug format. This step has to be performed always, even if an already implemented code plug is reused, you will need to verify that format. Any mistake here, may brick your device.

Note

Before you attempt to reverse engineer anything, consider to invest some significant time in research. Reverse engineering is an cumbersome and frequently frustrating task, there is no need to waste your valuable time on something, that someone else has already done. Also, consider documenting and publishing your results, so others can find it, even if it is incomplete. Someone else might pickup your work and complete it.

In the following sections, I attempt to describe, how I approach reverse engineering the communication protocols and code plugs. To this end, I hope it might help you in your reverse engineering work. And if you like, you may contribute your work to qdmr.

Reverse engineering communication protocols

There is no one ultimate way to successfully reverse engineer a communication protocol. But some methods are very common. First, one needs to get some captures of the communication between the manufacturer codeplug programming software (CPS) and the radio of interest. The majority of these radios will communicate via USB. Hence, some means are needed to capture USB traffic.

There is an application, called Wireshark (https://www.wireshark.org/). This application is usually used to capture and analyze network traffic. It can, however, also capture traffic from USB. How this is done, depends a bit on the operating system you are using. If you want to capture and analyze the communication under Windows™, read the instructions at https://wiki.wireshark.org/CaptureSetup/USB. This makes sense, as you need a windows installation, to run the manufacturer CPS.

I personally prefer to work under Linux, to this end, I've installed Windows in a virtual machine, but capture the USB traffic under the host Linux system. To do that, the **usbmon** kernel module must be loaded. If loaded, Wireshark will show USB as a capture source.

Once, capturing USB traffic works, connect the radio to the host. You may need to allow the guest system to have access to the USB device. Then fire up the manufacturer CPS, start capturing in Wireshark and start a codeplug read. Stop capturing once the read is complete. Save the captured data and restart the capturing. Then write the codeplug back to the device, stop the capture and save it in another file.

These files now contain all packets send and received by the PC over USB. In a next step, the packets must be filtered and inspected. To do that, I personally prefer to write short Python scripts. The pyshark package provides means to read packages from the capture file and access their content. This allows to filter those packages to and from the device, that contain the actual payload to and from the device.

Please note, that may radios may misuse some already existing protocols over USB. Some Radioddity and Baofeng devices use the HID specification to transfer data to and from the device. Others may use the DFU specification (TyT, Retevis), mass storage (CSi) or simply serial-over-USB (AnyTone).

Irrespective of the underlying specification used to send data to the device, the payload of these packages must be extracted.

To do that, use Wireshark to inspect these packages by hand. Search for a packet, that was definitely send to or by the device and note the USB (URB) device address. This is the best way to filter all packets to and from the device.

Example 8.1. Some simple example to filter packages by device address and destination/source.

```
import pyshark

packages = pyshark.FileCapture(FILENAME, include_raw=True, use_json=True)
device_address = '9' # just an example

for package in packages:
  if device_address != package.usb.device_address:
    continue
  if "host" == package.usb.src:
    # From host to device.
  else if "host" == package.usb.dst:
    # From device to host.
```

This example shows how to filter only those packets to and from a specific device and also dispatch depending on the destination and source of the packet. That is, if the packet was send by the host or the device.

In a next step, the captured payload must be inspected. If the packet contains any additional payload data, the `usb.data_length` field will be set. Then, there is a field called `usb.capdata_raw`. Unfortunately, that field name contains a dot, so you need to access it differently. That is

```
  # [...]
  if int(package.usb.data_length):
    data = package["usb.capdata_raw"].value
  # [...]
```

The extracted data is hex string. Now comes the difficult part. Staring at these packets and making sense of them. Usually, the communication is performed in a strict command-response structure. That is, the host sends a request packet and the device responses with a single packet. This eases the analysis a lot, as the communication is always initiated by the host and the association between the request and the response packets are trivial.

Identifying the communication structure

A typical communication with the device is performed in three steps: First, the radio is identified and brought into a programming mode. Several packets might be needed to perform this task. In that state, the radio usually displays some message on the screen or blinks an LED.

The second step consists of the actual codeplug transfer. There, a large amount of packets are send. This step is usually easy to find within the captured packets, as a large number of equally sized packets are exchanged. Sometimes, a relatively large amount of data is read or written at once. This can be spotted easily, as basic control commands of the first step are usually pretty short.

Frequently, these packets contain an address field or sequence number used to specify where the codeplug data chunk is written to or read from. These fields are of upmost importance and need to be identified. They are, however, easy to spot, as this field is likely constantly increasing, with a fixed increment from packet to packet.

The final step usually is to reboot the radio or leaving the programming mode. This is commonly performed by a single packet. Sometimes, the device reboots immediately and no response to the command is received by the host.

Once the protocol structure is identified, that is, the purpose of the single packets is known, the actual work of identifying the packet structure can start.

Identifying the packet format

The actual packet format will be much harder to reverse engineer. I cannot give any general suggestions on how to figure out the meaning of each byte in a packet. However, I can give you some examples. This might help in reverse engineering new protocols as the structure of the packets will be different, but the concepts remain the same. For example, read and write commands must somehow specify an address to read from and write to as well as the amount of data to read or write. This might be helpful

Let us inspect some packets send and received from an AnyTone device.

Example 8.2. Capture of a codeplug read from an AnyTone AT-D578UV

```
> 50 52 4f 47 52 41 4d                            | PROGRAM
< 51 58 06                                        | QX.

> 02                                              | .
< 49 44 35 37 38 55 56 00 12 56 31 31 30 00 00 06 | ID578UV..V110...

> 52 02 64 00 00 10                               | R.d...
< 57 02 64 00 00 10 fe ff ff ff ff ff ff ff ff ff | W.d.............
| ff ff ff ff ff ff 65 06                         | ......e.

> 52 02 64 00 10 10                               | R.d...
< 57 02 64 00 10 10 ff ff ff ff ff ff ff ff ff ff | W.d.............
| ff ff ff ff ff ff 76 06                         | ......v.

> 52 02 64 00 20 10                               | R.d. .
< 57 02 64 00 20 10 ff ff ff ff ff ff ff ff ff ff | W.d. ...........
| ff ff ff ff ff ff 86 06                         | ........

...

> 52 01 64 08 80 10                               | R.d...
< 57 01 64 08 80 10 00 ff ff ff ff ff ff ff ff ff | W.d.............
| ff ff ff ff ff ff ee 06                         | ........

> 52 02 48 02 00 10                               | R.H...
< 57 02 48 02 00 10 01 08 00 00 ff ff ff ff ff ff | W.H.............
| ff ff ff ff ff ff 59 06                         | ......Y.

...
```

The Example 8.2, “Capture of a codeplug read from an AnyTone AT-D578UV” shows a capture of the data exchanged between the host and an AnyTone AT-D578UV during a codeplug read. Everything starting with `>` is sent from the host to the device, while everything starting with `<` is sent by the device to the host.

The first packet send, is the ASCII string `PROGRAM`. This causes the device to enter the programming mode. A message will be shown on the screen of the device. This is then acknowledged by the device with a short message `QX` followed by a single `06h` byte. The last byte appears to be weird, but it will always be present on any response from the device. This might be seen as an end-of-packet byte.

The next command send by the host is pretty simple. It consists of a single `02h` byte. The device responds with a 15 bytes long ID string, identifying the device name and hardware version followed by the 06h end-of-packet byte.

After this, the actual codeplug read appears to start. The host sends a series of equally size commands, each starting with an `R` followed by 5 bytes of payload. As mentioned above, this payload must contain some sort of memory address and length field, to specify how much to read and from where. To figure that out, one may study subsequent read requests.

The payloads of the first tree read read requests are

```
02 64 00 00 10
02 64 00 10 10
02 64 00 20 10
```

The only difference of these requests is the fourth byte. Hence, one may assume, that this byte is part of the address field. However, we do not know which bytes are part of the address as well.

The responses to these requests also contain these bytes as well as 18 more bytes of payload. It is reasonable that the transfer size is a power of two. So the request likely reads 16 bytes from the device. 16 in hex is `10h`, hence one may assume that the last byte specifies the amount of data to read. Hence we identified the size field as the fifth byte of the read request payload data.

As the address increases by 10h from request to request, one may infer, that the read request specifies the address to read from, not the sequence or block number of a read from. As the entire Codeplug will not fit into 256 bytes, the address field must be larger. Even 64kb will be too small, to hold the entire codeplug, the address is likely a 32bit integer.

This does not need to be true. Some radios will implement special commands to select the memory bank beforehand. Then each 64kB bank can be accessed with a 16bit address.

To verify our assumption, we will study some read requests, that appear much later and check if the first bytes change as well. And indeed, much later, we observe read request payloads like

```
01 64 08 80 10
02 48 02 00 10
```

Obviously, these bytes change too. Consequently, we may assume that the address is stored as a 32bit unsigned integer in big-endian byte-order. Finally, we may summarize the read request format as

```
+---+---+---+---+---+---+
|'R'| Address       | N |
+---+---+---+---+---+---+
```

Where `N` is the number of bytes to read.

Now, it is time to study the structure of the read response. As we already reverse engineered the read request, the read response is then easy to understand. Each of these responses start with a `W` char, followed by the same address and length as send by the request. That is

```
> 52 02 64 00 00 10
< 57 02 64 00 00 10 fe ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 65 06
```

Obviously the 16 byte data read, follows the length byte. The last byte is the common 06h end-of-packet byte, we have already seen. So a single unknown 65h byte is still unknown.

This byte might be some sort of checksum over the payload. Checksums are notoriously difficult to figure out. Frequently, common techniques like CRC16 are used. Here, a single checksum byte is used, hence one of the common checksums is unlikely. Now starts some guesswork. We may compare some very similar read responses to get an idea, how this checksum may work. For example

```
57 02 64 00 10 10 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 76 06
57 02 64 00 20 10 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 86 06
```

The only difference between these two responses is one byte in the address, the remaining payload is identical. The address also only differs by 10h like the checksum. So one may assume, that the checksum is simply the sum over the payload bytes. Summing all bytes of the latter up to the checksum gets 10ddh. Of cause, this is not the checksum directly. The checksum is a single byte. But the least significant byte of the sum (ddh), however, does not match.

Maybe, the sum is not taken over the entire payload, but over the part that actually matters. That is, the address, length and data fields. This time, the sum is 1086h and this time, the least significant byte (86h) matches the checksum. Now, we can summarize the read response format as

```
+---+---+---+---+---+---+---+...+---+---+---+
|'W'| Address       | N | Data (N)  |CRC|06h|
+---+---+---+---+---+---+---+...+---+---+---+
```

where the CRC is the least significant byte of the sum over the address, length and data payload.

Reverse engineering of the code plug format

Once the communication protocol is known, it is time to focus on the code plug format. Depending on the code plug complexity, this can be a very cumbersome task. Moreover, depending on the device, it can be pretty hard to access the binary representation of the configuration (code plug) that is actually

written onto the device. In general, one cannot expect, that the binary file, created by the manufacturer, does relate to the binary code plug written onto the device at all.

To figure the relationship out, between the code plug file and the actual code plug written onto the device, the code plug must first be extracted from the Wireshark capture. As we already know the protocol, this can be done easily using Python and pyshark. When doing so, be careful with the addresses, data is written to or read from. The binary code plug might not be written sequentially and with gaps. A best practice is to extract the data read or written along with its addresses and sort it with respect to the address. Then dump everything as a hex-dump.

Once the code plug is extracted from the Wireshark capture, compare it to the corresponding code plug file created by the CPS. If you are lucky (usually for very cheap DMR radios), the CPS code plug file is simply the binary code plug written to the device (e.g., for GD77, RD5R). Then it is very easy to reverse engineer the code plug format, as you can study changes to the CPS code plug file.

If they are not identical, you will need to somehow extract the binary code plug written to the device from the manufacturer CPS. If the protocol is based on Serial-over-USB, you may be able to write an emulator for the device, knowing the communication protocol. This then allows to trick the CPS to talk to the emulator instead of the device and dump the code plug. If not, you are likely stuck with writing a lot of code plugs to the real device and extracting the code plug from the Wireshark captures. This can be considered the worst-case scenario.

Differential analysis

Irrespective on how you gain access to the binary code plug written onto the device, the analysis of this code plug is always the same. This is called differential analysis. The idea is, to change a single option in the CPS and compare the resulting binary code plugs bit by bit. Ideally, only a single place in the binary code plug will change too and this change then reflects the encoding of the particular setting touched. This is then repeated for all possible settings in the CPS until the entire code plug structure is known.

Note

There will be some bits and bytes in the code plug that will never changed. These are usually reserved bits/bytes. There may also be some undocumented and hidden options in the code plug, that cannot be set via the (normal) CPS.

Before you can start reverse engineering the code plug, you need to create a “base” code plug. That is, one small simple code plug, that touches every feature of the radio. It should not be overly complex with hundreds of channels, but should contain everything the radio provides. For example, it should contain the APRS settings, if the radio has GPS or a roaming zone if the radio supports roaming. You should also create two of every kind. That is, at least two channels, contacts, group lists, zones and scan lists. Otherwise, it will be hard to figure the index scheme out.

Once you created the base-code plug, get its binary representation (e.g., from file, emulation or Wireshark capture). Change something simple. For example the name of the first contact. Then get the binary representation of this modified code plug and compare the two binary code plugs.

As an example, consider the following difference between two hex-dumps of the binary code plugs for a BTech DMR-6X2UV Pro. Here, the name of the first contact was changed.

Example 8.3. Difference between two binary code plugs for a DMR-6X2UV Pro. Only the name of the first contact was changed.

```
< 02680000 : 01 43 6f 6e 74 61 63 74 20 31 00 00 00 00 00 00 | .Contact 1......
< 02680010 : 00 00 00 55 6e 6b 6e 6f 77 6e 00 00 00 00 00 00 | ...Unknown......
> 02680000 : 01 56 65 72 79 20 6c 6f 6e 67 20 6e 61 6d 65 2e | .Very long name.
> 02680010 : 2e 00 00 55 6e 6b 6e 6f 77 6e 00 00 00 00 00 00 | ...Unknown......
```

This difference already shows some very important information about the code plug encoding. First, the list of contacts appears to start at address 2680000h. This is very important for the coarse structure

of the binary code plug. We also learn about the structure of the encoded contacts, that the name starts at offset 01h. So, the first byte is still unknown. The length of the name is limited to 16 bytes and is terminated/padded with 0-bytes. However, we still don't know how large the encoded contact element in the code plug actually is.

To figure out the size of the contact element, one can use the fact, that all elements very likely have identical sizes. Hence the offset from one name to the name of the next contact will be the size of the contact element. We already know, that the contact list starts at the address 2680000h. Hence, we can look in the code plug directly at that address.

Example 8.4. Codeplug hex-dump at the contact-list address.

```
02680000 : 01 43 6f 6e 74 61 63 74 20 31 00 00 00 00 00 00 | .Contact 1......
02680010 : 00 00 00 55 6e 6b 6e 6f 77 6e 00 00 00 00 00 00 | ...Unknown......
02680020 : 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 | ................
02680030 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
02680040 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
02680050 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
02680060 : 00 00 00 00 01 43 6f 6e 74 61 63 74 20 32 00 00 | .....Contact 2..
02680070 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
02680080 : 00 00 00 00 00 00 00 00 00 00 02 00 00 00 00 00 | ................
02680090 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
026800A0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
026800B0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
026800C0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
```

The offset between the two names “Contact 1” and “Contact 2” is exactly 100 bytes (64h). With this, we know the exact size of the contact element. However, wo only know what 16 of these 100 bytes mean. So there is a lot to figure out. The differential analysis, however, makes it quiet easy. First we may change the call type of the first contact. From “group call” to “private call” and study the differences in the code plug.

Example 8.5.

```
< 02680000 : 01 43 6f 6e 74 61 63 74 20 31 00 00 00 00 00 00 | .Contact 1......
> 02680000 : 00 43 6f 6e 74 61 63 74 20 31 00 00 00 00 00 00 | .Contact 1......
```

We see, that only one byte has changed. The one at offset 0. It changed from 01h to 00h. We may conclude that this byte encodes the contact type, where 00h means private call and 01h means group call. Analogously, we find out that 03h means all-call.

The next step is to find the encoding and offset of the contact DMR ID. To find it, we reload the base code plug and change the DMR ID of the first contact from “1” to something more complex. E.g., to “1234567” and compare the binary .

Example 8.6.

```
< 02680020 : 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 | ................
> 02680020 : 00 00 00 01 23 45 67 00 00 00 00 00 00 00 00 00 | ....#Eg.........
```

Here we find some changes at the offset 23h. The content changed from 00000001h to 01234567h. This is somewhat weird, as the binary representation of 1234567 in hex is actually 12d687h. However, for DMR code plugs, it is quiet common to store integers in so-called BCD. Here each digit is stored in 4bits. This has the consequence, that the decimal digits are readable in hex. But this storage is inefficient, as the DMR ID is actually a 24bit number, that can be stored in 3 bytes. Its BCD encoding requires 4 bytes.

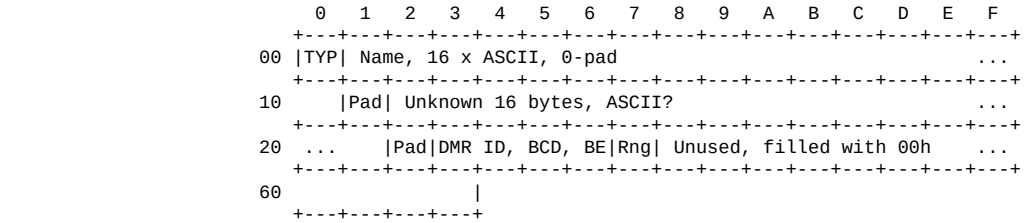
The last setting for contacts remaining in the CPS is the “call-alert”. This setting is only valid for private calls. Changing the contact 1 from group to private call, changing the call alert from “None” to “Ring” and comparing the code plugs, one gets:

Example 8.7.

```
< 02680000 : 01 43 6f 6e 74 61 63 74 20 31 00 00 00 00 00 00 | .Contact 1......
> 02680000 : 00 43 6f 6e 74 61 63 74 20 31 00 00 00 00 00 00 | .Contact 1......
< 02680020 : 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 | ................
> 02680020 : 00 00 00 00 00 00 01 01 00 00 00 00 00 00 00 00 | ................
```

Of cause, we find two changes. The change of the first byte, indicating the change from group to private call, as well as one byte changing at offset 27h from 00h to 01h. This is the only unexplained change and thus must reflect the alert-type setting. Consequently, the alert type “None” is represented by 00h and “Ring” by 01h. Analogously, we find that alert-type “Online” is encoded as 02h.

Obviously, we still don't know the majority of the 100 bytes of the contact element. Lets have a look at the structure of the contact element.



There are some additional Pad bytes (00h) that terminate ASCII strings. This is nothing, we found by the differential analysis, but it is some common scheme, found in many binary code plugs. Obviously, the firmware developer did not knew the `strnlen` C function.

The remaining bytes of the contact element appears to be empty, unused space. This is also quiet frequent in code plugs. They contain a lot of unused bytes that might be reserved for future uses. This allows the firmware developer to extend the code plug, while maintaining backward compatibility with older versions of the CPS.

Having reverse engineered the contacts, does not mean that you are done. There are a lot more elements to decipher. Did I mentioned that it is a lot of work? But the general scheme of things remain the same. Change a single setting in the CPS and compare the resulting binary code plugs. Document your findings. Step by step, you will figure out the entire code-plug structure.

Glossary

Admit Criterion

The admit criterion specifies a criterion under which transmissions on a channel are allowed. These criteria can be defined for both, analog and DMR channels. Irrespective of the channel type, the admit criteria “always” and “channel free” can be selected. The former simply always allows to transmit. The former requires the channel to be free.

For analog channels, usually there is also the criterion “tone”. This criterion would only allow transmissions if the matching CTCSS tone or DCS code has been received. For analog repeater operation, the criterion “channel free” should not be used as this would prevent any transmissions while the repeater is active.

For digital channels, usually there is also the criterion “color code”. This criterion would only allow transmissions if the matching color code has been received. To this end, this criterion is similar to the “tone” criterion for analog channels.

All Call

An all-call is a special type of an DMR call to a special number (*16777215*). All radios are supposed to receive this call irrespective of their particular configuration. This call type is usually used in emergency situations or on simplex channels.

Unfortunately, the local talk group (number 9) is frequently used on simplex channel although only those radios will receive calls to this talk group if it is in the group list assigned to the simplex channel. To be on the safe side, use the all-call for simplex calls.

Automatic Packet Reporting System

The Automatic Packet Reporting System is a protocol that uses single broadcast frequencies to transmit small amounts of information though a network of repeaters. Usually but not limited to position reports. This allows other services (e.g., aprs.fi [https://aprs.fi] to display that information on a map. The usual APRS frequencies are 144.390 MHz (North America), 144.800 MHz (Europe) and 145.175 MHz (Australia).

Codeplug

The term “codeplug” is rather loosely defined. Usually it refers to the binary representation of the configuration of a radio that is written to the radio to configure it. It contains all contacts, channels, zones, settings, etc that form the configuration of the radio.

Color Code

A color code is some additional information attached to each DMR call. It is used to prevent interference between repeaters with overlapping ranges that operate at the same frequencies. In amateur radio, this may only happen in very densely populated areas. It is much more frequent in commercial applications, where a single company may got only a few frequencies assigned but needs much more repeaters to cover a campus reliably.

The color code is a number between 0 and 16. A radio or repeater may only react to calls with matching color codes. Hence beside the actual input and output frequencies, the color code of a repeater must be known to be able to access it.

Continuous Tone Coded Subaudio Squelch

Means to control the squelch of a radio or to open a repeater, that does not rely on the strength of a carrier. Instead a tone is transmitted along with the audio, which is usually filtered out (e.g., less than 300Hz) by the receiver. There are a set of common CTCSS tone frequencies, but may radios allow for specifying arbitrary sub-tone frequencies.

Decentralized Amateur Paging Network	DAPNET, short for Decentralized Amateur Paging Network does exactly what it says. It is a network of transmitters that transmit pager messages in the UHF band, usually at 439.9875 MHz. As the old pager operated at a frequency near by, they can be modified to operate at that frequency and can therefore be used in amateur radio. The DAPNET is particularly popular with emergency operators as it allows for an convenient and reliable multicast notification.
DCDM	Stands for dual capacity direct mode. This allows for two independent connections on a single simplex channel by TDMA. Lacking a single central repeater, one of the participants must act as the so-called *leader*. This radio then defines the clock and thus time slot 1 and 2. All remaining participants must act as so-called *follower*. They must synchronize to the clock of the *leader*. This obviously requires that all participants can reach the *leader* directly.
Digital Coded Squelch	Similar to the CTCSS tones, the DCM allows to open the squelch or repeater by transmitting sub-tones along with the audio. These tones are usually filtered out by the receiver (e.g., below 300Hz). In contrast to CTCSS tones, DCS uses these tones to repeatedly transmit a digital code.
EchoLink	EchoLink is a network of analog FM repeaters that allows to link repeaters within this network temporarily. That is, two FM repeaters that are linked via EchoLink behave like a single repeater irrespective of their location. This network also allows to access the repeaters directly though the internet. This is particularly helpful if one cannot reach any repeater.
Group Call	A group call is simply a call to a talk group. That is, not to a single participant but rather to a group of participants. Every participant that has this talk group in their group list can receive this call. See also Group List and Talk Group.
Group List	A group list is a simple list of talk groups. A group list is then assigned to a channel to specify which group calls to receive on that channel. If a talk group is not listed in the group list of a channel, the radio will ignore calls to this talk group on that channel. The DMR network cannot know which talk groups you are interested in. You have to tell your radio using group lists.
Hang Time	The *hang time* specifies the time period, a DMR call remains “active” after it ended. For this time period, the call will replace the default transmit contact on that channel. This allows to directly answer a call received on a channel, even if that call is not the default transmit contact on the channel.
Maidenhead Locator	The Maidenhead locator, also known as “QTH locator” or “IARU locator” is a means for transferring a geo-location over radio. Using a so-called grid-square system.
Lone Worker	“Lone worker” refers to a feature of many DMR radios used in some commercial settings, where a participant in the network needs to report regularly to some other station. This feature is certainly not very useful for the HAM radio context.
Private Call	A DMR call to an individual participant. Usually, radios do not receive private calls not addressed to the radio. This therefore can be used to talk to a specific HAM over a repeater without disturbing other participants. The repeater however, will be blocked during that time.
Revert Channel	A revert channel is a designated channel the radio resorts to, to transfer some information or the radio switches to, when the PTT is pressed. This is usually used in the context of APRS (i.e., the channel to switch to send the position)

	or scan lists (i.e., the channel the radio will switch to, when the PTT is pressed during the scan).
Roaming Channel	Specifies which settings of the current channel needs to be overridden to stay in contact with a particular talk group. These settings are usually those, specific for a repeater. That is, receive and transmit frequency and color code. The time-slot might also be overridden.
Roaming Zone	A collection of *roaming channels* usually sharing the same transmit contact. When roaming is enabled (and supported by the radio) this list of channels is scanned for a reachable channel once the current repeater gets out-of-range. This way, one can stay in contact with a particular talk group when being mobile.
Secondary Station Identifier	AX.25 addresses (and thus APRS) consists of a call sign and a so-called SSID (0-15). This number can be used as a replacement for the lack of ports in AX.25 or to provide some additional information about the station.
Talkaround	
Talk Group	The term “Talk Group” refers to a DMR ID, which represents a virtual contact. DMR repeater may subscribe to one or more talk groups. Every call to these talk groups are then transmitted by those repeaters, that are subscribed to these talk groups. Usually, it is possible to subscribe a repeater to any talk group temporarily by simply starting a transmission to this talk group on a repeater channel. For some time period after the initial transmission, the repeater will transmit all calls to this now subscribed talk group. The subscription gets renewed with every further call the talk group. See Also Group Call.
Time-Division Multiple Access	See Time Slot.
Time Slot	Due to the lossy compression of audio data, it is possible to send more than twice the amount of audio data through a single 12.5kHz wide channel than needed. This allows to provide two independent audio streams within a single physical channel. There are basically two ways to separate these two audio streams. Either by frequency, effectively splitting the single 12.5kHz channel into two 6.25kHz wide ones. Alternatively, one can split the channel in “time”. That is, the band width remains 12.5kHz but each of the two simultaneous participants sends one after the other. One in time slot 1 and the other in time slot 2. This is called TDMA and DMR uses this technique to provide two independent audio and data streams within a single physical channel. For this TDMA to work, one participant needs to define the “clock” to specify when time slot 1 or 2 happen. This role is usually taken by the repeater but on direct simplex connections, one of the participants might provide that clock. This is then called DCDM (Dual-Capacity Direct Mode). This mode, however, makes little sense in amateur radio context.
Transmit Contact	A transmit or default contact can be specified for each DMR channel. Some radios actually require that each channel has an transmit contact. This contact can be any contact you want to call whenever the PTT is pressed on that channel. It is, however, possible to answer any received call directly by pressing the PTT button within the so-called hang time, although the call may not match the transmit contact. These hang times can be set within the radio settings, usually for private and group calls separately.

It is also possible to call any contact by selecting it from the contact list or by entering the contacts DMR ID into the keypad.

Transmit Timeout

Specifies the time of continuous transmission, after which the transmission is interrupted automatically. This prevents the accidental blocking of a repeater or talk group by transmitting continuously.

Scan List

A scan list is just a list of analog and digital channels that one wants to scan. A scan list can be associated with each channel. This scan list is then used whenever a scan is started on that channel.

Zone

As there are usually several channels defined per repeater (at least two), the number of defined channels grows rapidly. To organize this large number of channels, zones are used. These zones usually collect all relevant channels for a region. That is all (analog and digital) channels associated with the local repeaters. The relevant zone is then selected via the keypad on the radio and only those channels in that zone are then accessible.

Note

A defined channel that is not member of any zone is usually not accessible in the radio.

Some radios allow to specify a different zones for each VFO. Others set a zone globally. In the latter case, the zone consists of two channel lists. One for each VFO.

Index