Universal Radio Hacker

Investigate Wireless Protocols like a Boss

User Guide / Tutorial

Guide version:	1.1
Refers to URH version:	2.0

Contact: Johannes.Pohl90@gmail.com
Andreas.Noack@hochschule-stralsund.de

Contents

1 Introduction

1.1 Motivation

The Universal Radio Hacker (URH) is a tool for analyzing unknown wireless protocols. With the rise of Internet of Things (IoT) such protocols often appear in the wild. Many IoT devices operate on frequencies like 433.92 MHz or 868.3 MHz and use proprietary protocols for communication. Reverse-engineering such protocols can be fascinating (»What does my fridge talks about?«) and reveal serious security leaks e.g. when bypassing smart alarm systems and door locks.

So how can we join this game? **Software Defined Radios** (SDR) are the answer for this. Such devices allow sending and receiving on nearly arbitrary frequencies. Figure 1 shows two examples. Both devices cost about 200 euro.

(a) HackRF One can send and receive on frequencies from 1 MHz to 6 GHz.

(b) SDRplay RSP2pro can receive on frequencies from 1 kHz to 2 GHz.

Figure 1: Two examples for Software Defined Radios.

Like the name suggests SDRs need software to be properly operated. This is where the **Universal Radio Hacker** comes into play: It takes the samples from the SDR and transforms them into binary information (bits). But this is only the beginning: URH is designed to help you throughout the entire process of attacking the wireless communication of IoT devices. In the upcoming sections you will learn how to use URH and reverse engineer wireless protocols in minutes.

1.2 Installation

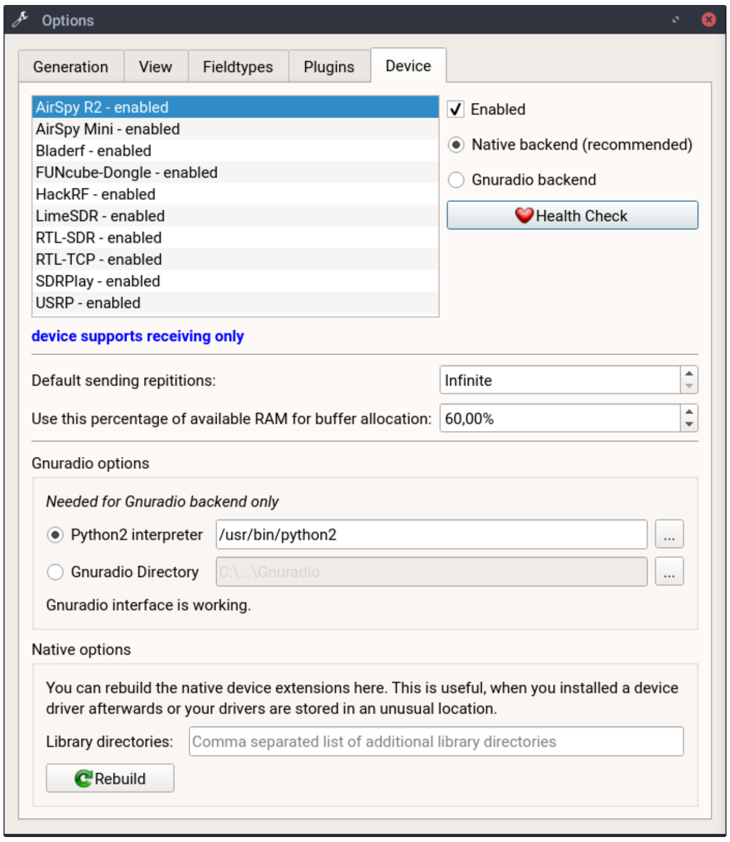
URH runs on Linux, OSX and Windows and is easy to install. You can find the latest installation instructions here: https://github.com/jopohl/urh#installation

2 Device interaction

To get started with investigating a wireless signal we need to capture it with a Software Defined Radio. URH offers support for a wide range of common SDRs. In this section you will learn how to configure your SDR and use it inside URH.

2.1 Configuring Devices

First, lets ensure that your device is enabled in Edit → Options → Device. You will see a list of supported devices like shown in fig. 2a.



AirSpy R2 - enabled
AirSpy Mini - enabled
Bladerf - enabled
FUNcube-Dongle - enabled
HackRF - enabled
LimeSDR - enabled
RTL-SDR - enabled
RTL-TCP - enabled
SDRPlay - enabled
USRP - enabled

[✓] Enabled
[●] Native backend (recommended)
[ ] Gnuradio backend

device supports receiving only

Default sending repititions: Infinite
Use this percentage of available RAM for buffer allocation: 60,00%

Gnuradio options

Needed for Gnuradio backend only

[●] Python2 interpreter /usr/bin/python2
[ ] Gnuradio Directory C:\...\Gnuradio

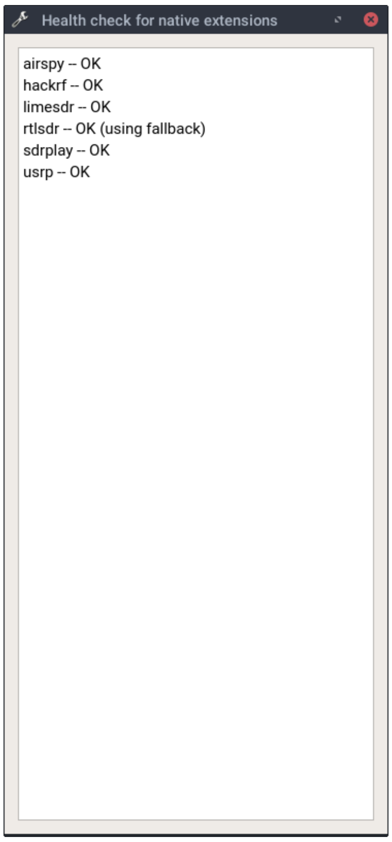
Gnuradio interface is working.

Native options

You can rebuild the native device extensions here. This is useful, when you installed a device driver afterwards or your drivers are stored in an unusual location.

Library directories: Comma separated list of additional library directories

(a) Device options tab.



```
airspy -- OK
hackrf -- OK
limesdr -- OK
rtlsdr -- OK (using fallback)
sdrplay -- OK
usrp -- OK
```

(b) Health check native backend.

Figure 2: Configuring devices

If you select a device in the list you can either **enable/disable** it via the Enabled checkbox. Disabling a device will prevent it from showing up in the device selection in the main program. Furthermore, you can **choose** between **native** and **GNU Radio backend**. More on this in the following sections.

2.1.1 Native Backend

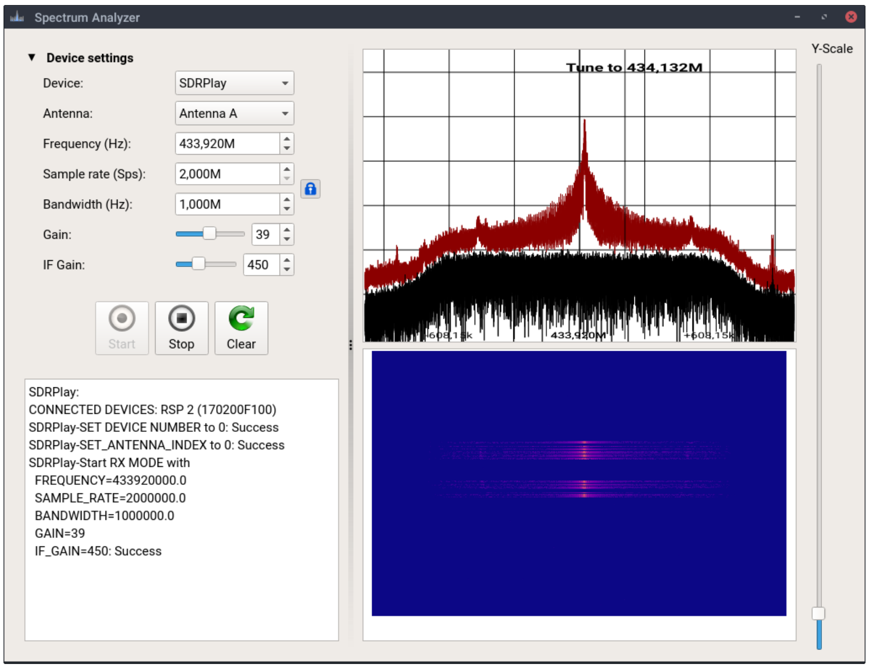
The native device backend should be used whenever possible. If native backend is greyed out for a device, there may be a library missing. You can check this by clicking the Health Check button. A dialog like shown in fig. 2b will pop up and give you more information about the status of native device backend. If you indeed miss a library you can install it on your system. For example, install HackRF library on Ubuntu with `sudo apt-get install libhackrf-dev`. After that, hit the Rebuild button at the bottom of the dialog.

2.1.2 GNU Radio Backend

As a fallback to the native backend, you can use GNU Radio to access your device. In order to do that configure either the path to your Python 2 interpreter or (on Windows) the GNU Radio installation directory. Then, you get notified that GNU Radio backend is available and can choose it after selecting your desired device from the device list in fig. 2a.

2.2 Scanning the spectrum

Having configured our SDR let's start by scanning the wireless spectrum and find the center frequency of the device we want to attack. To open the spectrum analyzer use File → Spectrum Analyzer, this will give you a dialog as shown in fig. 3.



Device: SDRPlay
Antenna: Antenna A
Frequency (Hz): 433,920M
Sample rate (Sps): 2,000M
Bandwidth (Hz): 1,000M
Gain: 39
IF Gain: 450

```
SDRPlay:
CONNECTED DEVICES: RSP 2 (170200F100)
SDRPlay-SET DEVICE NUMBER to 0: Success
SDRPlay-SET_ANTENNA_INDEX to 0: Success
SDRPlay-Start RX MODE with
 FREQUENCY=433920000.0
 SAMPLE_RATE=2000000.0
 BANDWIDTH=1000000.0
 GAIN=39
 IF_GAIN=450: Success
```

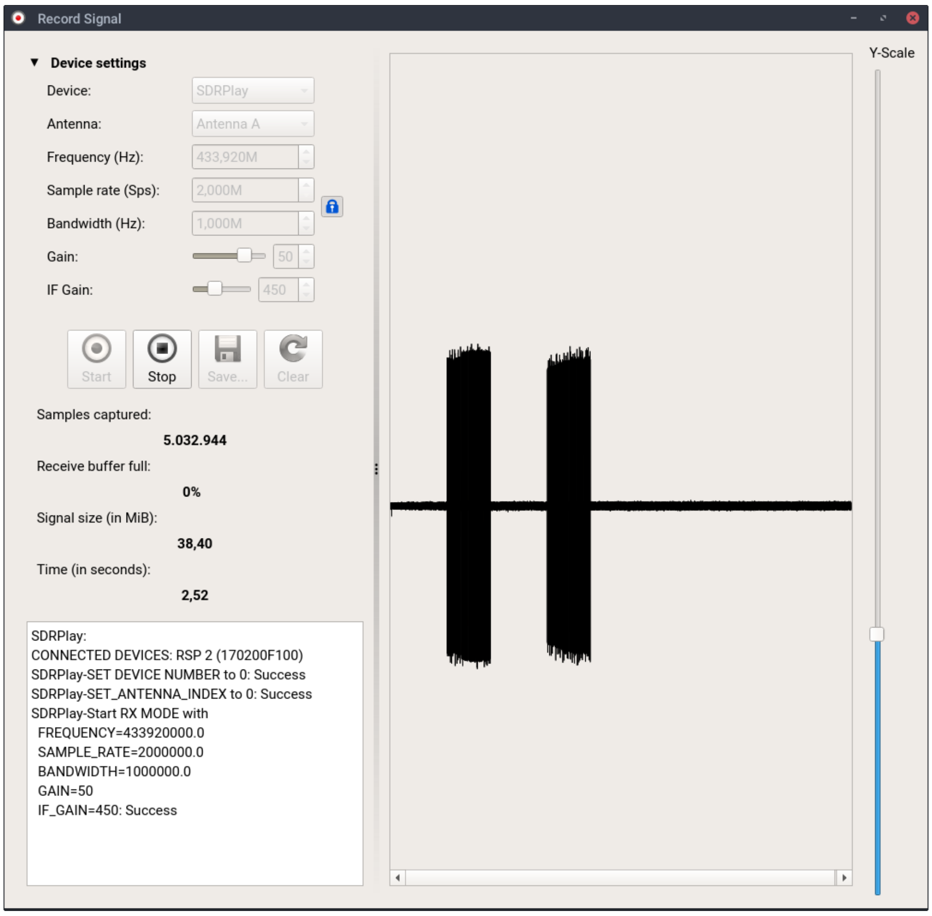
Figure 3: Spectrum Analyzer dialog.

Device settings can be made on the left. After starting the spectrum analyzer, you can watch how URH tunes your device to the desired parameters in the bottom left.

When spectrum analysis starts, you see the spectrum at the top right and a waterfall plot of the spectrum in the bottom right. Both views together give you a good impression of the spectrum and allow you to find the target frequency easily. To change the current frequency you can either use the spinbox on the left or click on the desired point in the spectrum. This way, you can navigate through the spectrum and visually find the desired frequency.

2.3 Recording a signal

Having found the target frequency it is time to record a signal for later analysis. Open up the recording dialog via File → Record Signal and you will see that it saved your parameters from Spectrum Analyzer, so you can simply start the recording with the record button on the middle left. In fig. 4 recording is already running and a wireless remote control was pressed two times. You can clearly see the two signals in the preview on the right.



Device: SDRPlay
Antenna: Antenna A
Frequency (Hz): 433,920M
Sample rate (Sps): 2,000M
Bandwidth (Hz): 1,000M
Gain: 50
IF Gain: 450

Samples captured: 5.032.944
Receive buffer full: 0%
Signal size (in MiB): 38,40
Time (in seconds): 2,52

```
SDRPlay:
CONNECTED DEVICES: RSP 2 (170200F100)
SDRPlay-SET DEVICE NUMBER to 0: Success
SDRPlay-SET_ANTENNA_INDEX to 0: Success
SDRPlay-Start RX MODE with
 FREQUENCY=433920000.0
 SAMPLE_RATE=2000000.0
 BANDWIDTH=1000000.0
 GAIN=50
 IF_GAIN=450: Success
```

Figure 4: Record signals in this dialog-

Stop the recording with the stop button. After this you can save the signal and make another recording or simply close the dialog. You will be asked if you want to save your current recording. Congratulations, you successfully recorded your first signal!

3 Interpretation

Having recorded a signal, URH adds it automatically to the Interpretation Tab (fig. 5). Before we explore this tab, let's have a look at other ways of opening and importing a signal.

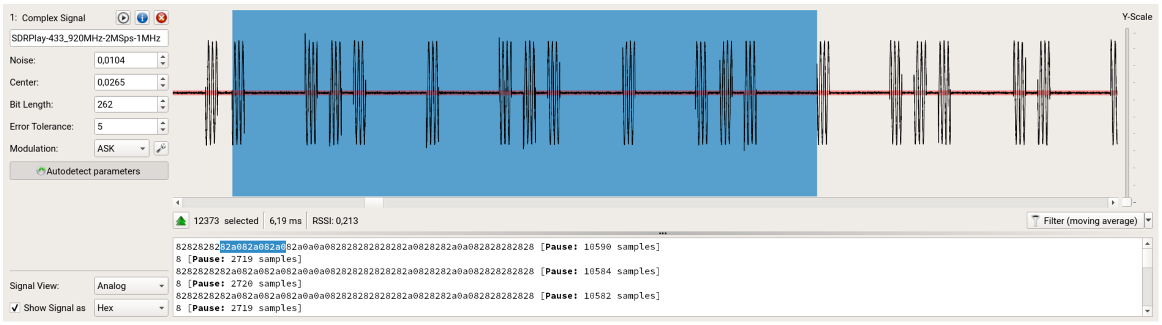
3.1 Importing a signal

Apart from recording, a signal can be added to Interpretation tab via File → Open. File ending determines how URH handles the signal. URH understands these file endings:

- `.complex` files with complex64 samples (32 Bit float for I and Q, respectively). This is the default signal file format and will also be used in case the file has no ending at all.
- `.complex16u` using two unsigned 8 Bit integers for I and Q
- `.complex16s` using two signed 8 Bit integers for I and Q
- `.wav` files can be imported, but must not be compressed, i.e., they should be PCM.
- `.csv` e.g. from USB oscilloscope can be imported using a CSV wizard available with File → Import → IQ samples from csv.

3.2 Signal editing

Having loaded a signal into Interpretation it can be zoomed using the mouse wheel or context menu. Make a selection with the mouse and navigate via drag by holding the **shift key**. This can be changed with the checkbox Edit → Options → Hold shift to drag.



SDRPlay-433_920MHz-2MSps-1MHz

Noise: 0,0104
Center: 0,0265
Bit Length: 262
Error Tolerance: 5
Modulation: ASK
Signal View: Analog
✓ Show Signal as Hex

12373 selected | 6,19 ms | RSSI: 0,213

```
8282828282a082a082a082a0a0a08282828282828282a0828282a0a082828282828 [Pause: 10590 samples]
8 [Pause: 2719 samples]
8282828282a082a082a082a0a0a08282828282828282a0828282a0a082828282828 [Pause: 10584 samples]
8 [Pause: 2720 samples]
8282828282a082a082a082a0a0a08282828282828282a0828282a0a082828282828 [Pause: 10582 samples]
8 [Pause: 2719 samples]
```

Figure 5: Recorded signal gets automatically added to Interpretation tab.

Apart from these basic functions URH has a powerful signal editor. After making a selection use the context menu to:

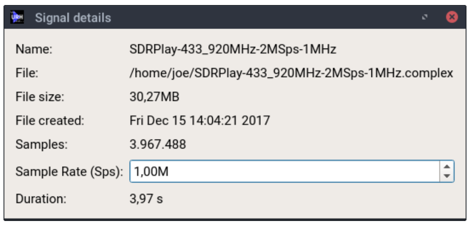
- **Copy/paste** parts of the signal
- **Delete** the selection
- **Crop** to current selection, i.e., remove everything that is not selected.

- **Mute** the selection, i.e., set the selected part to zero.
- **Create** a new signal from the selection
- **Assign a participant** more on this in section 4.5.
- **Set noise level from selection**

Using these features you can isolate interesting signal parts and fix noisy recordings.

3.3 Replay signal and view signal details

With the ▶ button in the top of fig. 5 you can **replay** the captured signal. The ℹ button opens signal detail dialog (fig. 6) where you can also edit the sample rate. This influences the displayed time in Interpretation and Analysis.


Signal details

Name:	SDRPlay-433_920MHz-2MSps-1MHz
File:	/home/joe/SDRPlay-433_920MHz-2MSps-1MHz.complex
File size:	30,27MB
File created:	Fri Dec 15 14:04:21 2017
Samples:	3.967.488
Sample Rate (Sps):	1,00M
Duration:	3,97 s

Figure 6: Signal detail dialog.

3.4 Demodulation

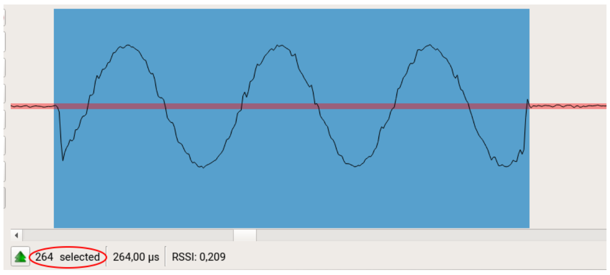
Demodulation is the process of converting the recorded sine waves into bits. URH does this automatically for you whenever you add a signal to Interpretation tab. You see the resulting bits right below the signal.

You can change the modulation type using the combobox on the left. Whenever you change the modulation type the demodulation parameters are newly detected by URH. You can disable this by clicking the Autodetect parameters button. Of course, you can fine tune the parameters by using the spinboxes.

The demodulation parameters are

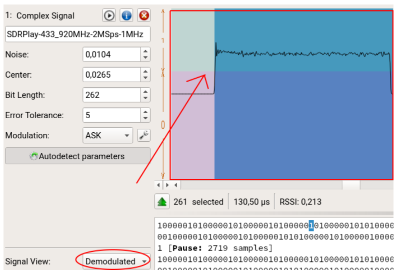
- **Noise**: Define which power must the surpassed to be not counted as noise.
- **Bit Length**: The sample length of a bit. You can find this out by selecting a pulse with minimum length and looking at the number of selected samples as shown in fig. 7a.
- **Center**: The center in demodulated view separating the ones from the zeros. To tune this value either use the spinbox or change the Signal View from Analog to Demodulated (see fig. 7b) and move the center between the areas with the mouse.
- **Error Tolerance**: Fine tune how tolerant (in samples) the demodulation routine shall be against errors.

Note that these parameters are found automatically and only need to be changed manually in rare cases e.g. bad signal recordings.



264 selected | 264,00 µs | RSSI: 0,209

(a) Finding the bit length manually.



1: Complex Signal

SDRPlay-433_920MHz-2MSps-1MHz

Noise: 0,0104

Center: 0,0265

Bit Length: 262

Error Tolerance: 5

Modulation: ASK

Signal View: Demodulated

261 selected | 130,50 µs | RSSI: 0,213

```
10000010100000101000001010000010100000101010000
0010000010100000101000001010100000101000001000
1 [Pause: 2719 samples]
1000001010000010100000101000001010000010101000
```

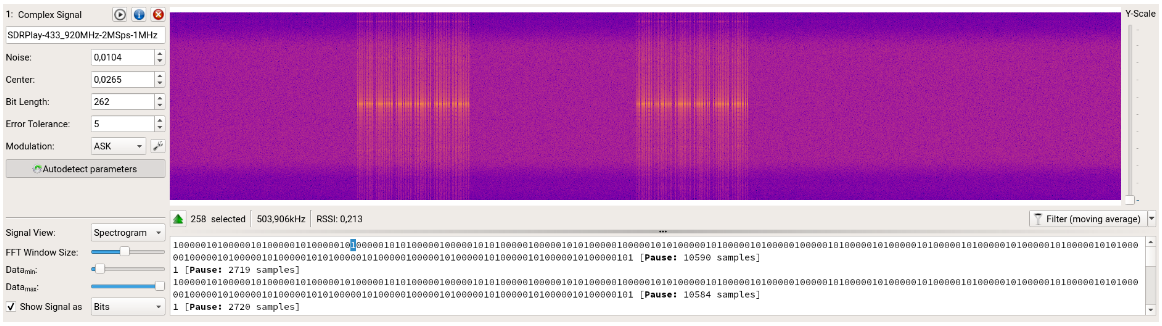
(b) Setting the center for demodulation.

Figure 7: Finding bit length and center visually.

In case of ASK demodulation you can make advanced settings using the button next to the demodulation combobox.

3.5 Spectrogram

With the spectrogram view (fig. 8) you can view the frequency spectrum of your signal during time. Simply switch to this view via Signal View → Spectrogram.



1: Complex Signal

SDRPlay-433_920MHz-2MSps-1MHz

Noise: 0,0104

Center: 0,0265

Bit Length: 262

Error Tolerance: 5

Modulation: ASK

Signal View: Spectrogram

FFT Window Size:

Data_{min}:

Data_{max}:

✓ Show Signal as: Bits

258 selected | 503,906kHz | RSSI: 0,213

```
1000001010000010100000101000001010000010101000001000001010100000100000101010000010000010101000001000001010100000101000001010000010000010100000101000001010000010100000101000001010000010101000
00100000101000001010000010101000001010000010000010100000101000001010000010100000101000001010000010100000101 [Pause: 10590 samples]
1 [Pause: 2719 samples]
1000001010000010100000101000001010000010101000001000001010100000100000101010000010000010101000001000001010100000101000001010000010000010100000101000001010000010100000101000001010000010101000
00100000101000001010000010101000001010000010000010100000101000001010000010100000101000001010000010100000101 [Pause: 10584 samples]
1 [Pause: 2720 samples]
```

Figure 8: Spectrogram view for a signal.

The spectrogram view has three parameters:

1. **FFT Window Size** is the used STFT (Short-Time-Fourier-Transform) window size and determines the time resolution of the spectrogram.
2. **Data**$_{\mathrm{min}}$ is the minimal frequency magnitude for the spectrogram. All magnitudes below **Data**$_{\mathrm{min}}$ will be set to this value.
3. **Data**$_{\mathrm{max}}$ is the maximal frequency magnitude for the spectrogramm. All magnitudes above **Data**$_{\mathrm{max}}$ will be set to this value.

You can change the spectrogram appearance using Edit → Options → View.

3.6 Filters (advanced feature)

URH supports filters to get the most of your signals. There are two types of filters available: bandpass filter and moving average filter. We will see both filter types in the next two sections. Note, this is an advanced feature and not needed in most cases.

3.6.1 Bandpass filter

You can apply a bandpass filter in the spectrogram view (section 3.5) to a defined selection and seperate channels from each other or correct misaligned signals. To do this, make a selection around your desired frequency band like in fig. 9.

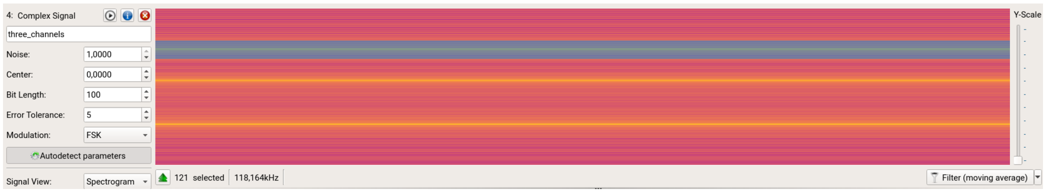
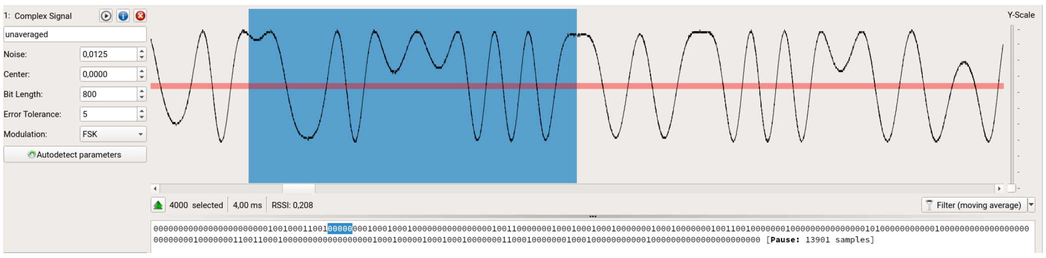

4: Complex Signal
three_channels
Noise: 1,0000
Center: 0,0000
Bit Length: 100
Error Tolerance: 5
Modulation: FSK
Signal View: Spectrogram
121 selected | 118,164kHz

Figure 9: Select your desired frequency range for filtering.

After that, right click on the spectrogram and select Create signal from frequency selection. A filtered signal will be created and added under the original signal. If you are not pleased with the result, try tuning the filter bandwidth using context menu and click Configure filter bandwidth... to make a configuration dialog appear. Here you can e.g. increase accuracy of the filter by choosing a lower bandwidth.

3.6.2 Moving average filter

Have a look at the signal in fig. 10. The demodulation does not look good.


1: Complex Signal
unaveraged
Noise: 0,0125
Center: 0,0000
Bit Length: 800
Error Tolerance: 5
Modulation: FSK
4000 selected | 4,00 ms | RSSI: 0,208

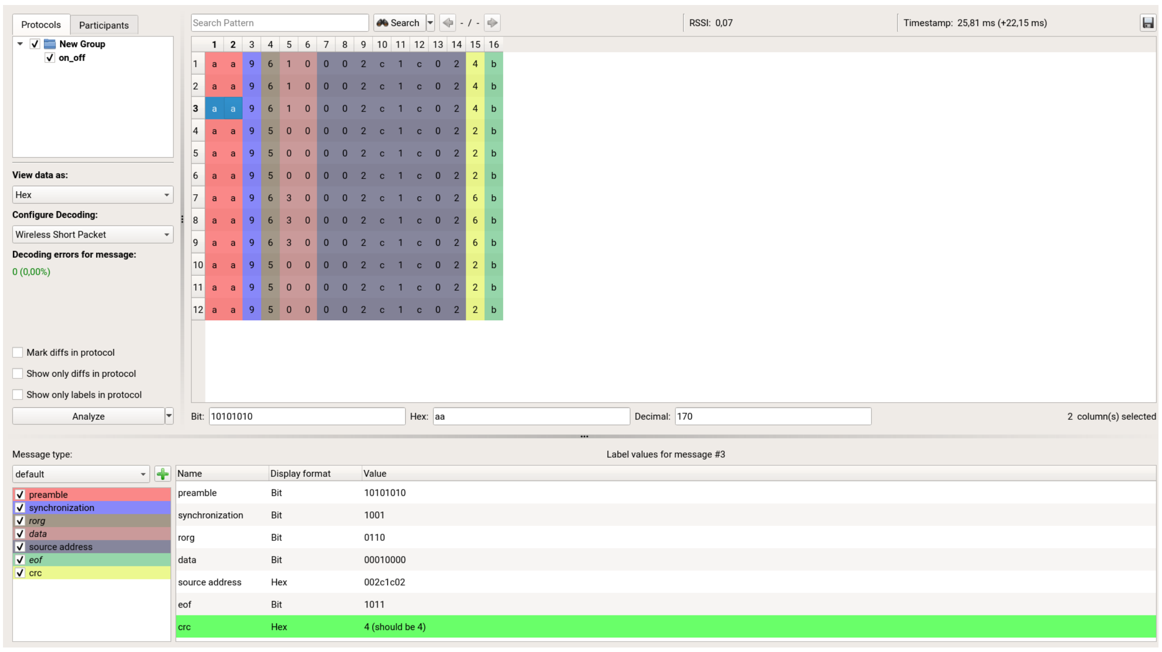
```
0000000000000000000000001001000110010000000010001000100000000000000000100110000000100010001000100000001000100000000100110010000000100000000000000010100000000000010000000000000000000
0000000010000000110011000100000000000000000000100010000010001000100000001100010000000100010000000000001000000000000000000000000 [Pause: 13901 samples]
```

Figure 10: Signal with quantization errors.

The reason for this is, that there are quantization errors in the signal. This can either be fixed with better recording hardware or by clicking the filter button below the signal. This will improve demodulation results by applying a moving average filter to the signal. The filter can be fine tuned and customized using the little arrow on the right of the button and choosing Configure filter.... This opens the filter configuration dialog.

4 Analysis

In Interpretation phase we demodulated a signal and transformed the sine waves into bits. But what do these bits mean? To find out we need to perform protocol reverse engineering. URH supports this process with the Analysis tab (fig. 11). We will break down the various features of this tab in this section.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
2	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
3	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
4	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
5	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
6	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
7	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
8	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
9	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
10	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
11	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
12	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b

Bit: 10101010 Hex: aa Decimal: 170

✓ preamble
✓ synchronization
✓ *rorg*
✓ *data*
✓ source address
✓ *eof*
✓ crc

Label values for message #3

Name	Display format	Value
preamble	Bit	10101010
synchronization	Bit	1001
rorg	Bit	0110
data	Bit	00010000
source address	Hex	002c1c02
eof	Bit	1011
crc	Hex	4 (should be 4)

Figure 11: Analysis tab.

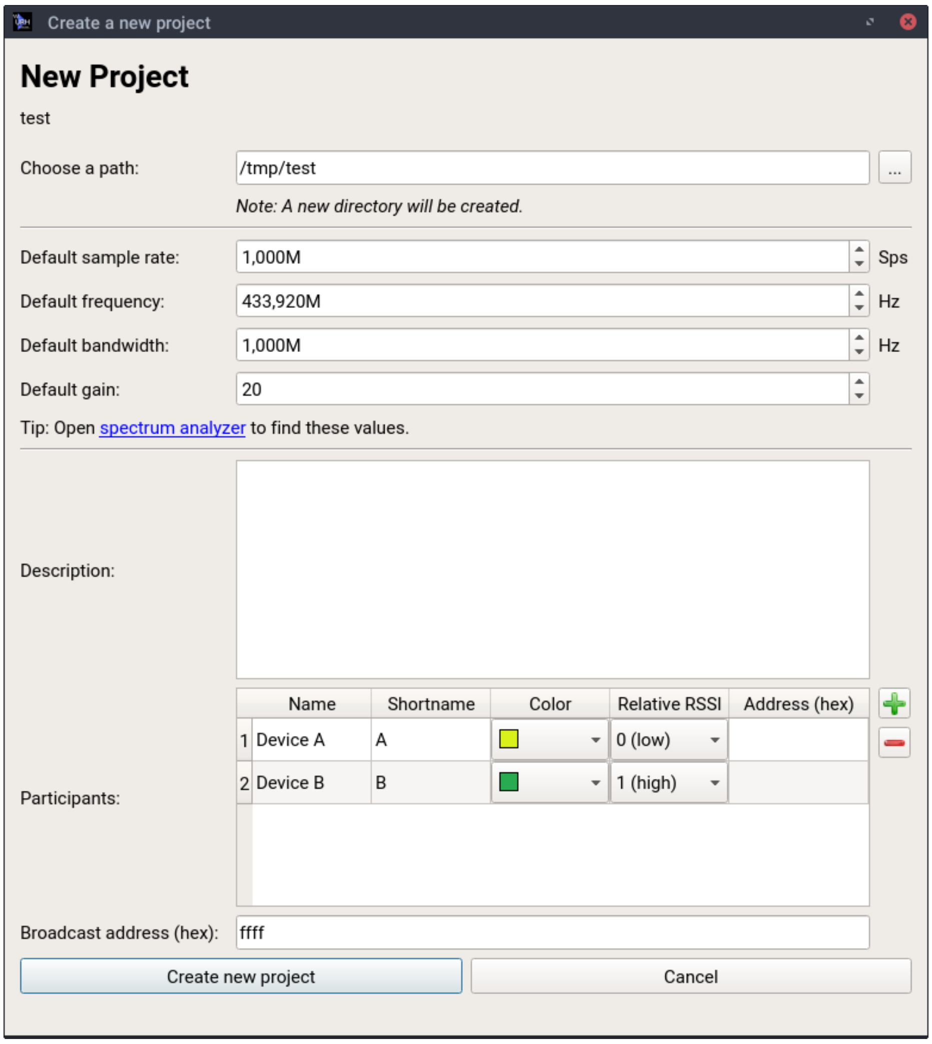
4.1 Getting data into analysis

Before we explore this tab in more detail let's see how you can get data into the Analysis tab. There are three ways:

1. **From Interpretation tab**: This is the default way. The bits from each signal you open in Interpretation tab will automatically be added to Analysis tab.
2. **Load a `.proto` file**: With the save button in the top right of the Analysis tab you can save your protocol and load it again via File → Open.
3. **Load plain bits from `.txt` file**: Got some bits lying around you want to make a quick analysis for? Just save your bits into a file that ends with `.txt` and open it via File → Open. Make sure your file only contains characters 1 and 0. This is also useful when your bits come from an external application.

4.2 Project files

A word of caution here: You already made some adjustments in the Interpretation tab and will probably do a lot more in Analysis tab. Your work will be lost when you close the program unless you **create a project**. You can convert your current work to a project any time via File → New Project. This will give you a dialog (fig. 12) for your new project.



New Project

test

Choose a path:	/tmp/test
	Note: A new directory will be created.
Default sample rate:	1,000M Sps
Default frequency:	433,920M Hz
Default bandwidth:	1,000M Hz
Default gain:	20

Tip: Open spectrum analyzer to find these values.

Description:

Participants:

	Name	Shortname	Color	Relative RSSI	Address (hex)
1	Device A	A	🟨	0 (low)	
2	Device B	B	🟩	1 (high)	

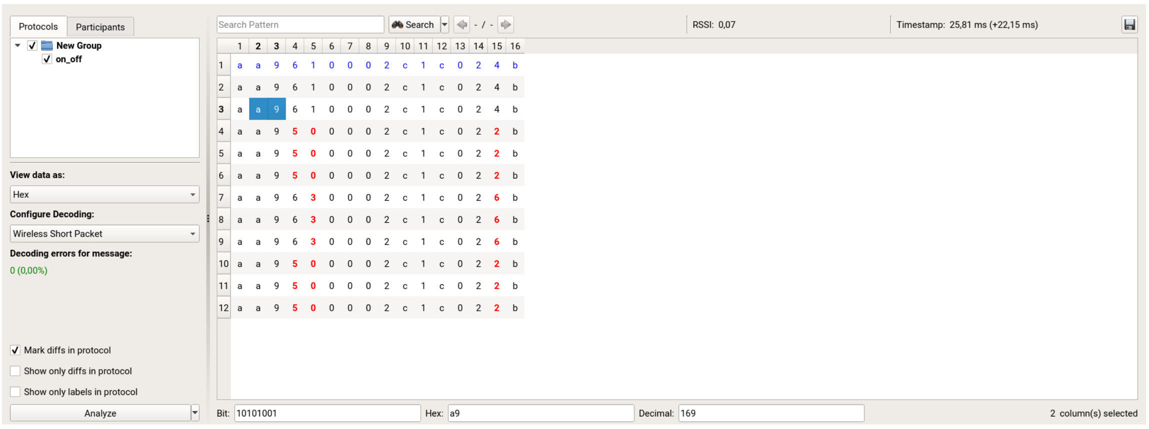
Broadcast address (hex): ffff

Figure 12: Dialog for creating a new project. All your changes from a non-project will be transferred to the new project.

All you need to do is to choose a directory where your project shall be saved to. Furthermore, you can make some default settings for your project and add a description. You can also configure the participants (see section 4.5) such as a remote control or a smart home central that are investigated in this project. Although you create a new project, your current work will be transferred, given that you are not already working on a project. If you want to make a fresh start, just close everything before creating a new project via File → Close all.

4.3 View data from different perspectives

Let’s start exploring the Analysis tab by looking at the top of it first as shown in fig. 13.



View data as: Hex

Configure Decoding: Wireless Short Packet

Decoding errors for message: 0 (0,00%)

✓ Mark diffs in protocol

Show only diffs in protocol

Show only labels in protocol

Analyze

RSSI: 0,07

Timestamp: 25,81 ms (+22,15 ms)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
2	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
3	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
4	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
5	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
6	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
7	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
8	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
9	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
10	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
11	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
12	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b

Bit: 10101001 Hex: a9 Decimal: 169 2 column(s) selected

Figure 13: Top part of the Analysis tab.

All messages are aligned under each other in the center table for easy comparison. For convenience, you can mark differences using the checkbox at the bottom left of fig. 13.

The top row above the table allows to:

- **Search** for patterns in the data.
- View the Received Signal Strength Indicator **RSSI** of the selected message.
- View the **absolute and relative timestamp** for the selected message.
- **Save** the current protocol with the button on the right.

Right below the table you see current selection in **Bit**, **Hex** and **Decimal** representation.

Now, let’s have a look at the left part of fig. 13. In the Protocols tab you can define which protocols you want to see in the table. Simply uncheck a protocol to hide it. For a better structure you can also create groups using the context menu and move your protocols around these groups using drag and drop or using the context menu. In the participants tab you can hide messages from certain participants (see section 4.5).

With the View data as combobox you can choose how the data in the table should be presented: Bit, Hex and ASCII view are supported. When opening the program, the view will default to the value set in Edit → Options → View → Default View.

With the Configure decoding combobox you can assign a decoding to the currently selected messages. You will learn more about decodings in section 4.4. Below this combobox you see how many errors occurred during decodind the selected message.

The Analyze button at the bottom left of fig. 13 automates many Analysis steps. It can

- **Assign participants** based on the RSSI of each message.
- **Assign decodings** based on the decoding errors for each message.

- **Assign messagetypes** based on the configured rules described in section 4.7.
- **Assign labels** based on heuristics for protocol labels.

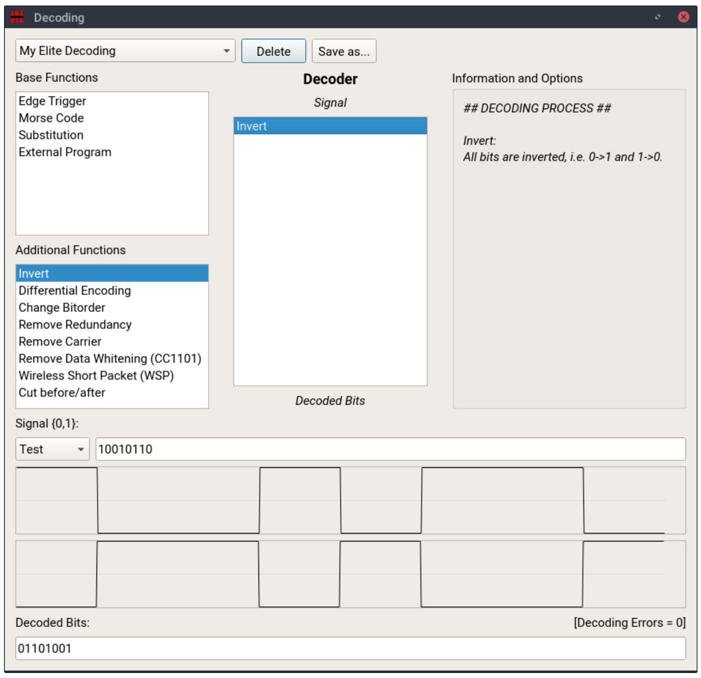
You can configure Analysis button behavior using the arrow on it's right. Especially when assigning different participants to many messages this button becomes useful.

4.4 Decoding

Wireless protocols may use sophisticated encodings to prevent transmission errors or increase their energy efficiency. While this is a great idea for designing good IoT protocols it somehow hinders you investigating the (true) transmitted data.

4.4.1 Configure Decodings

Therefore, URH comes with a powerful decoding component. You can access it via Edit → Decoding from the main menu or by selecting ... as decoding in the Analysis tab. This will open the decoding dialog shown in fig. 14.



## DECODING PROCESS ##

Invert:
All bits are inverted, i.e. 0->1 and 1->0.

Signal {0,1}: Test 10010110

Decoded Bits: [Decoding Errors = 0]

01101001

Figure 14: In the decoding dialog custom decodings can be crafted.

A decoding consists of (at least one) primitives that are chained together and processed based on their position in the list from top to bottom. You can add primitives by drag&drop.

You can preview the effect of your decoding with the fields in the lower area of the dialog. This way you can make experiments without leaving the dialog.

4.4.2 External Decodings

Have a very tough encoding and can't find a way to build it with the given primitives? No worries! You can program your decoding in your favored language and use it right in URH!

The interface for your external program is simple:

- Read the input as string (e.g. `11000`) as command line argument
- Write the result to STDOUT

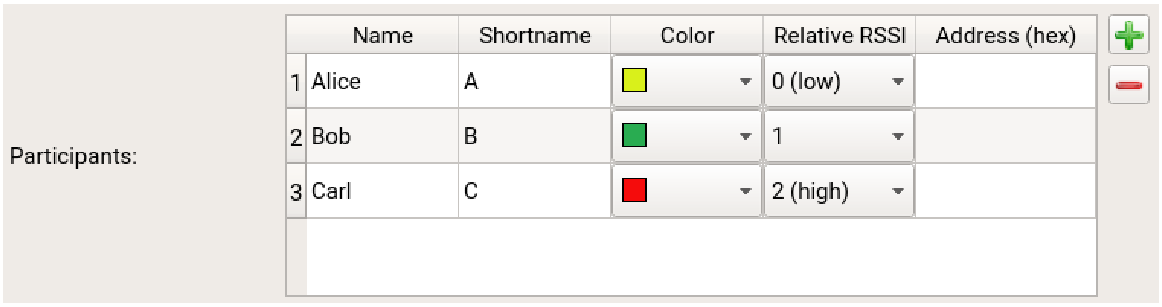
You learn more about external decodings in our GitHub wiki:
https://github.com/jopohl/urh/wiki/Decodings#use-external-program.

4.4.3 Choose decoding

Save your new decoding with the Save as... button and it will be added to the list of available decodings in the Analysis tab. To apply your crafted decoding to your data, just select the messages you want to set it for (Ctrl + A for all) and use the Configure Decoding combobox in the Analysis tab.

4.5 Participants

Protocol reverse engineering gets really complicated when multiple participants are involved. For example, a wireless socket may be switched from a smart home central which is triggered by a remote control. To keep an overview, URH offers configurable participants. To use this feature, you need to create or load **a project** (section 4.2) and configure your participants via File → Project settings. Here, you will find the table from fig. 15.



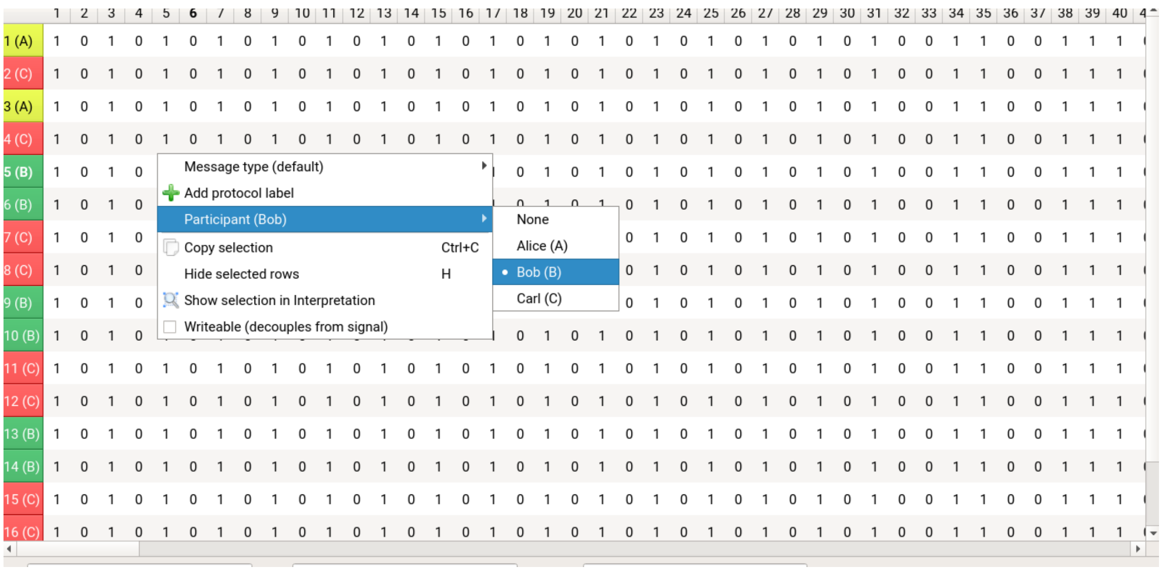
Participants:

	Name	Shortname	Color	Relative RSSI	Address (hex)
1	Alice	A	🟨	0 (low)	
2	Bob	B	🟩	1	
3	Carl	C	🟥	2 (high)	

Figure 15: Use this table in project settings to configure participants.

Use the ➕ and ➖ buttons right to the table to add or remove participants, respectively. You can configure name, shortname and color of a participant to adapt it to your project. The relative RSSI will be used from URH to assign participants automatically when pressing the Analyze button (section 4.3). If you know a address of a participant, you can enter it here to help URH during automatic finding of labels when pressing the Analyze button.

To **assign** participants, you can either use Analyze button to do this automatically or use the context menu of the message table as shown in fig. 16

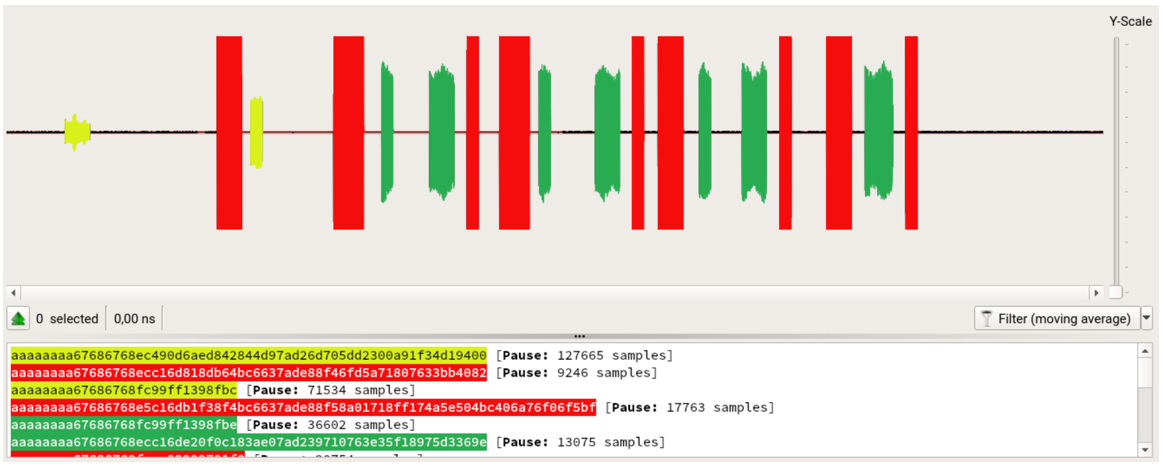


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1 (A)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
2 (C)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
3 (A)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
4 (C)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
5 (B)	1	0	1	0														0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
6 (B)	1	0	1	0														0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
7 (C)	1	0	1	0																		0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
8 (C)	1	0	1	0																		0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
9 (B)	1	0	1	0																		0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
10 (B)	1	0	1	0														0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
11 (C)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
12 (C)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
13 (B)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
14 (B)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
15 (C)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
16 (C)	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1

Figure 16: Manually assign a participant to selected messages with context menu.

As shown in fig. 16, each row header gets the background color of the assigned participant of the respective message. Furthermore, the participants shortname is added behind the row number.

With the context menu you can also show the current selection in Interpretation. This tight integration with Interpretation allows to find bit errors easily. Furthermore, the messages in Interpretation are also colored according to the assigned participants as shown in fig. 17.



```
aaaaaaaa67686768ec490d6aed842844d97ad26d705dd2300a91f34d19400 [Pause: 127665 samples]
aaaaaaaa67686768ecc16d818db64bc6637ade88f46fd5a71807633bb4082 [Pause: 9246 samples]
aaaaaaaa67686768fc99ff1398fbc [Pause: 71534 samples]
aaaaaaaa67686768e5c16db1f38f4bc6637ade88f58a01718ff174a5e504bc406a76f06f5bf [Pause: 17763 samples]
aaaaaaaa67686768fc99ff1398fbe [Pause: 36602 samples]
aaaaaaaa67686768ecc16de20f0c183ae07ad239710763e35f18975d3369e [Pause: 13075 samples]
```

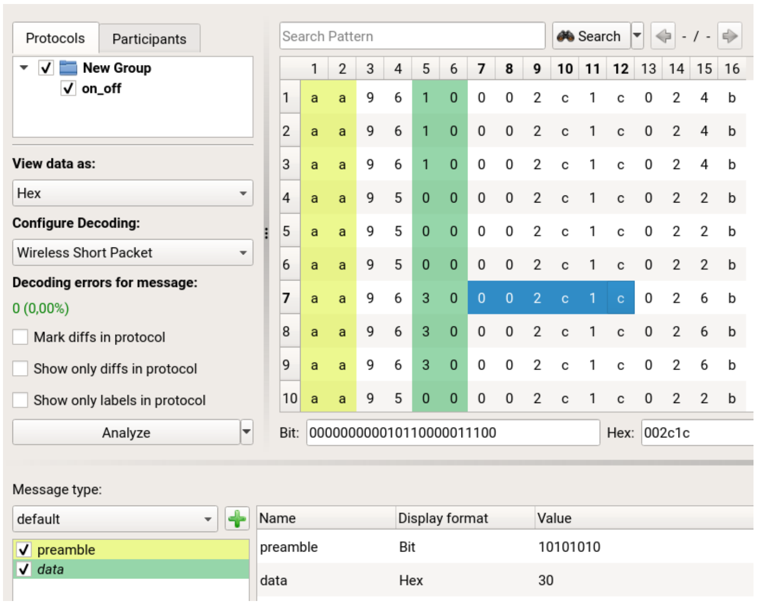
Figure 17: Participants are also visualized in Interpretation.

4.6 Labels

Messages of a protocol contain fields like `Length`, `Address` or `Synchronization`. During Analysis, you will reveal protocol fields one by one. To help you with this task, URH supports assigning **labels** to arbitrary parts of messages. These labels represent your current hypothesis for a field. You will learn how to master labels in this section.

4.6.1 Assigning and editing labels

To assign a label simply select your desired range in message table and use context menu (right click) → Add protocol label. After adding a label, you can enter a name for it in the list view on the list view in bottom left (see fig. 18) or choose a predefined name based on the configured field types (section 4.6.2).



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
2	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
3	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
4	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
5	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
6	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
7	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
8	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
9	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
10	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b

Bit: 000000000010110000011100 Hex: 002c1c

✓ preamble
✓ *data*

Name	Display format	Value
preamble	Bit	10101010
data	Hex	30

Figure 18: Assigned labels will be added to label list view and Wireshark like preview.

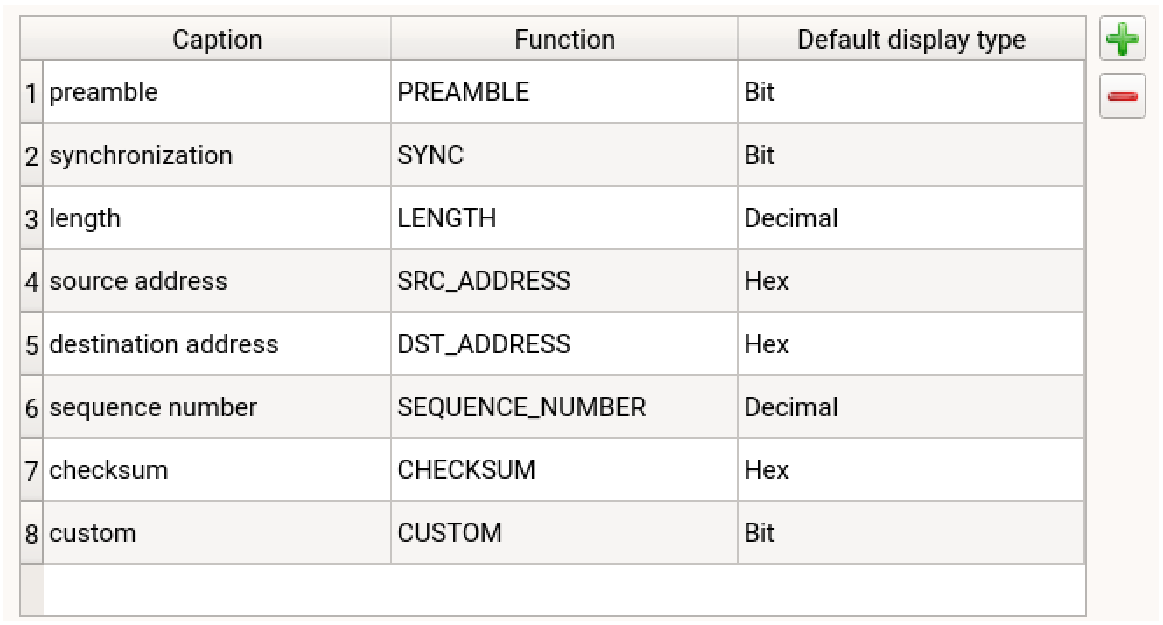
Once the label is added, you notice three changes on Analysis tab:

- The according data in message table is colored in label color.
- The label is added to the label list on bottom left. You can also uncheck a label here to hide all ranges with this label from message table.
- When moving the selection in message table you see the values of all labels for the current message in a Wireshark like table view below the message table. You can configure the Display format here using the combobox in second column. Possible values are Bit, Hex, ASCII and Decimal.

In order to **edit** a label simply right click on it either in message table or label list. This will open a new dialog, where you can edit name, start, end and color of each label.

4.6.2 Field Types

Field types can speed up the assigning of labels even further by defining default values for label names. Field type configuration dialog can be opened either via context menu of label list view or Edit → Options → Fieldtypes. By default, the field types from fig. 19 are available.



	Caption	Function	Default display type
1	preamble	PREAMBLE	Bit
2	synchronization	SYNC	Bit
3	length	LENGTH	Decimal
4	source address	SRC_ADDRESS	Hex
5	destination address	DST_ADDRESS	Hex
6	sequence number	SEQUENCE_NUMBER	Decimal
7	checksum	CHECKSUM	Hex
8	custom	CUSTOM	Bit

Figure 19: Configure the field types you want to use.

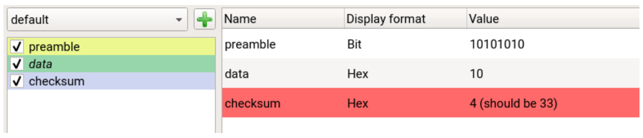
You can add or remove field types with the buttons right to the table. All Captions will be available when adding new labels in Analysis and also considered in **autocompletion** when entering a label name. This way, you can spare yourself from typing the same label names again and again for each new project.

The Default display type is the initial option for the Display format of the label preview table (bottom of fig. 18) when a new label with this field type is added.

The Function of a field type is mostly used internally by URH when it assigns labels automatically. The `CHECKSUM` function, however, is an exception, because it unlocks advanced functionality. Let's explore this feature in the next section.

4.6.3 Checksum labels

If your protocol has checksums, URH can verify these for you. Simply assign a label with a field type that has `CHECKSUM` function configured and you will get instant feedback if message's checksum matches the expectation in label preview table as shown in fig. 20.



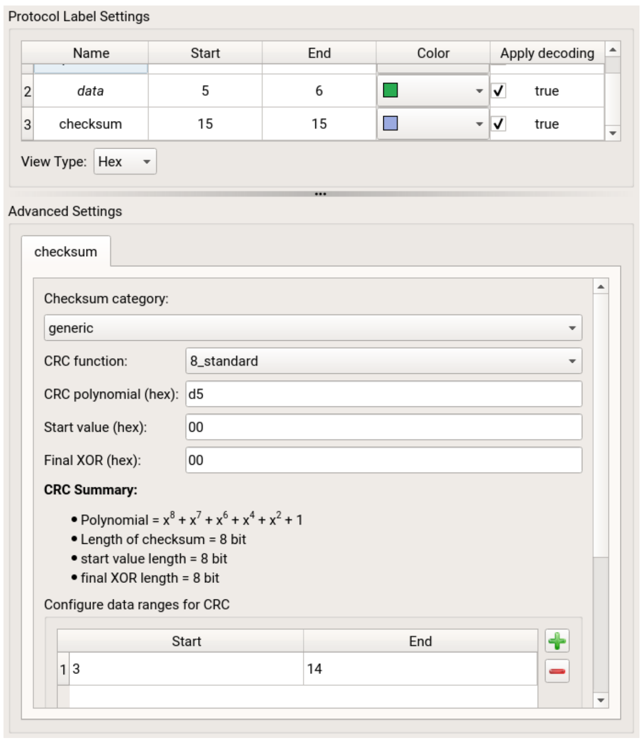
default

- ✓ preamble
- ✓ *data*
- ✓ checksum

Name	Display format	Value
preamble	Bit	10101010
data	Hex	10
checksum	Hex	4 (should be 33)

Figure 20: Feedback about checksum.

You can configure the type of the checksum in the label detail dialog. This dialog can be opening using context menu of the label in label list view or message table. Whenever a checksum label is present in the set of labels the checksum configuration as shown in fig. 21 will appear. Here you can define a CRC polynomial (generic category) or choose the Wireless Short Packet (WSP) checksum which is included by default.



Protocol Label Settings

	Name	Start	End	Color	Apply decoding
2	*data*	5	6	🟩	✓ true
3	checksum	15	15	🟦	✓ true

View Type: Hex

Advanced Settings

checksum

Checksum category: generic

CRC function: 8_standard

CRC polynomial (hex): d5

Start value (hex): 00

Final XOR (hex): 00

CRC Summary:

- Polynomial = $x^8 + x^7 + x^6 + x^4 + x^2 + 1$
- Length of checksum = 8 bit
- start value length = 8 bit
- final XOR length = 8 bit

Configure data ranges for CRC

	Start	End
1	3	14

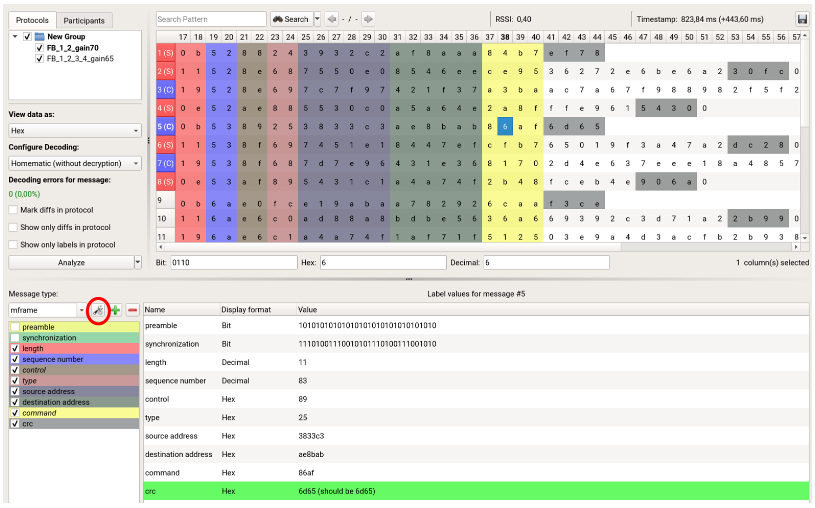
Figure 21: Checksum configuration.

In the generic category you will always see a preview of your configured CRC so you have instant feedback when editing it. In the table at the bottom you define for which ranges of the message the checksum shall be calculated. By default, these ranges exclude labels of type `PREAMBLE` and `SYNC`. Note, you can also add multiple ranges using the ✚ button when you have a more complicated checksum that excludes certain ranges.

4.7 Message Types

Message types take the label concept to the next level. Complex protocols tend to have different message types like ACK and DATA. By default, URH uses only one message type and applies configured labels to all messages based on their position only. With message types, you gain another level of control as you can configure labels per message type.

Message types can be added with the ✚ button next to the message type combobox right above the label list view. New message types will be bootstrapped with the labels from the default message type. You can assign message types to messages by selecting them in message table and use the context menu. This way, you can handle complex protocols like the one shown in fig. 11.



View data as: Hex

Configure Decoding: Homematic (without decryption)

Decoding errors for message: 0 (0,00%)

Mark diffs in protocol

Show only diffs in protocol

Show only labels in protocol

RSSI: 0,40

Timestamp: 823,84 ms (+443,60 ms)

	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57
1 (S)	0	b	5	2	8	8	2	4	3	9	3	2	c	2	a	f	8	a	a	a	8	4	b	7	e	f	7	8													
2 (S)	1	1	5	2	8	e	6	8	7	5	5	0	e	0	8	5	4	6	e	e	c	e	9	5	3	6	2	7	2	e	6	b	e	6	a	2	3	0	f	c	0
3 (C)	1	9	5	2	8	e	6	9	7	c	7	f	9	7	4	2	1	f	3	7	a	3	b	a	a	c	7	a	6	7	f	9	8	8	9	8	2	f	5	f	2
4 (S)	0	e	5	2	a	e	8	8	5	5	3	0	c	0	a	5	a	6	4	e	2	a	8	f	f	f	e	9	6	1	5	4	3	0	0						
5 (C)	0	b	5	3	8	9	2	5	3	8	3	3	c	3	a	e	8	b	a	b	8	6	a	f	6	d	6	5													
6 (S)	1	1	5	3	8	f	6	9	7	4	5	1	e	1	8	4	4	7	e	f	c	f	b	7	6	5	0	1	9	f	3	a	4	7	a	2	d	c	2	8	0
7 (C)	1	9	5	3	8	f	6	8	7	d	7	e	9	6	4	3	1	e	3	6	8	1	7	0	2	d	4	e	6	3	7	e	e	e	1	8	a	4	8	5	7
8 (S)	0	e	5	3	a	f	8	9	5	4	3	1	c	1	a	4	a	7	4	f	2	b	4	8	f	c	e	b	4	e	9	0	6	a	0						
9	0	b	6	a	e	0	f	c	e	1	9	a	b	a	a	7	8	2	9	2	6	c	a	a	f	3	c	e													
10	1	1	6	a	e	6	c	0	a	d	8	8	a	8	b	d	b	e	5	6	3	6	a	6	6	9	3	9	2	c	3	d	7	1	a	2	2	b	9	9	0
11	1	9	6	a	e	6	c	1	a	4	a	7	4	f	1	a	f	7	1	f	5	1	2	5	0	3	e	9	a	4	d	3	a	c	f	b	2	b	9	3	8

Bit: 0110 Hex: 6 Decimal: 6 1 column(s) selected

Message type: mframe

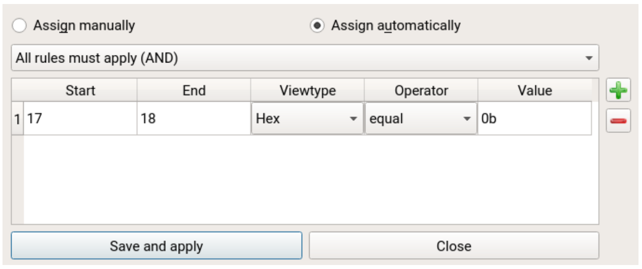
☐	preamble
☐	synchronization
✓	length
✓	sequence number
✓	*control*
✓	*type*
✓	source address
✓	destination address
✓	*command*
✓	crc

Label values for message #5

Name	Display format	Value
preamble	Bit	10101010101010101010101010101010
synchronization	Bit	11101001110010101110100111001010
length	Decimal	11
sequence number	Decimal	83
control	Hex	89
type	Hex	25
source address	Hex	3833c3
destination address	Hex	ae8bab
command	Hex	86af
crc	Hex	6d65 (should be 6d65)

Figure 22: A more complex protocol with various message types.

Assigning message types manually can be tedious. Therefore, URH has a feature to **assign message types automatically** based on rules. You can configure these rules for a message type by clicking the highlighted button in fig. 22. This will pop up the rule configuration dialog. Configuring rules is straightforward using the shown table.



Assign manually [● Assign automatically]

All rules must apply (AND)

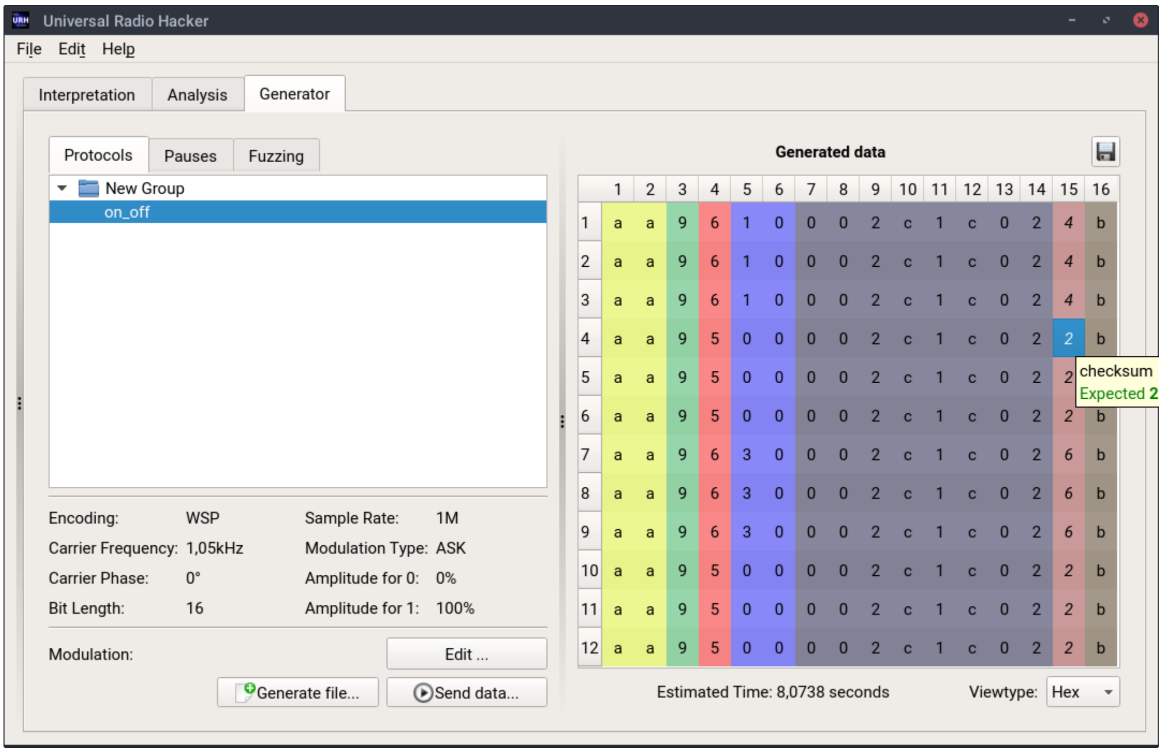
	Start	End	Viewtype	Operator	Value
1	17	18	Hex	equal	0b

Figure 23: Rules for automatic message type assignment can be configured in this dialog.

When hitting the Save and apply button URH will close the dialog and apply the message types based on the rules. You can also enforce the update of automatically assigned message types using the context menu of the label list view.

5 Generation

So far we have demodulated wireless signals in Interpretation and reverse engineered protocol logic in Analyis. It is time to get the reward for this work and finally break a wireless protocol. This is exactly what the Generator tab (fig. 24) is for.



Encoding: WSP
Sample Rate: 1M
Carrier Frequency: 1,05kHz
Modulation Type: ASK
Carrier Phase: 0°
Amplitude for 0: 0%
Bit Length: 16
Amplitude for 1: 100%

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
2	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
3	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4	b
4	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
5	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	[illegible]
6	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
7	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
8	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
9	a	a	9	6	3	0	0	0	2	c	1	c	0	2	6	b
10	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
11	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b
12	a	a	9	5	0	0	0	0	2	c	1	c	0	2	2	b

Estimated Time: 8,0738 seconds
Viewtype: Hex

Figure 24: Generator tab.

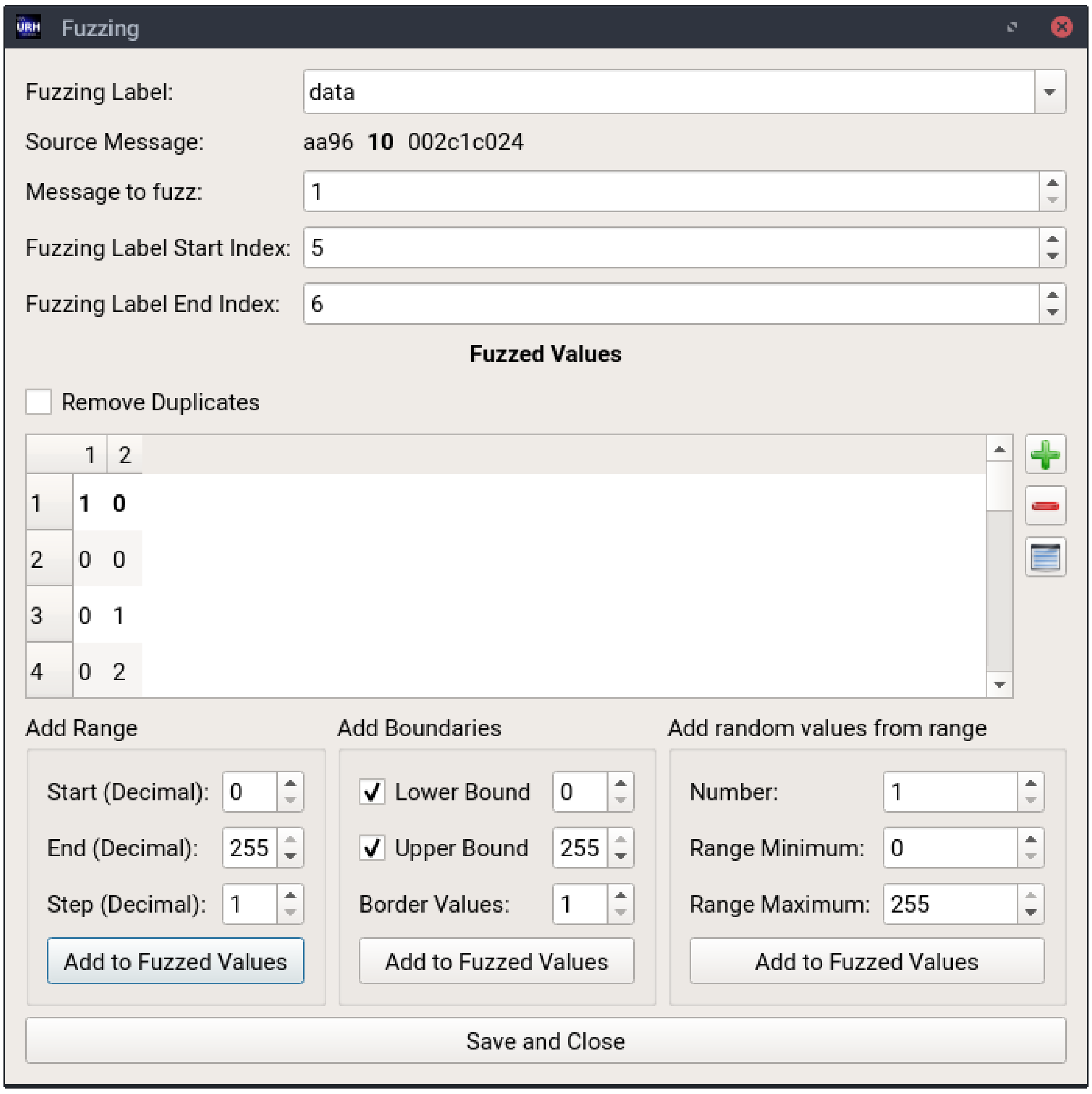
You can add a protocol via drag&drop from the tree view on the right. As you see, all your labels from Analysis are transferred when doing this. The table is **editable** so you can manipulate the data. Furthermore, you can add columns or empty messages using the context menu of the table.

The Generator has also a special feature for checksum labels: It will automatically **recalculate the checksum** whenever you change message data. This way, you do not need to worry about the checksums when manipulating data. Moreover, a checksum label's tooltip identifies whether the checksum is correct or not.

5.1 Fuzzing

While it is possible to edit messages by hand it is much faster to let URH do the dirty work for us. This is the purpose of the **fuzzing** component. You can enter a range of values for a label you want to fuzz and just hit a button while URH takes care of the rest. Let's see how this works.

Start by selecting your desired label in message table, then right click on it and choose Edit fuzzing label. This will open the fuzzing dialog from fig. 25


Fuzzing

Fuzzing Label: data

Source Message: aa96 **10** 002c1c024

Message to fuzz: 1

Fuzzing Label Start Index: 5

Fuzzing Label End Index: 6

Fuzzed Values

☐ Remove Duplicates

	1	2
1	**1**	**0**
2	0	0
3	0	1
4	0	2

Add Range

Start (Decimal): 0

End (Decimal): 255

Step (Decimal): 1

Add to Fuzzed Values

Add Boundaries

✓ Lower Bound 0

✓ Upper Bound 255

Border Values: 1

Add to Fuzzed Values

Add random values from range

Number: 1

Range Minimum: 0

Range Maximum: 255

Add to Fuzzed Values

Save and Close

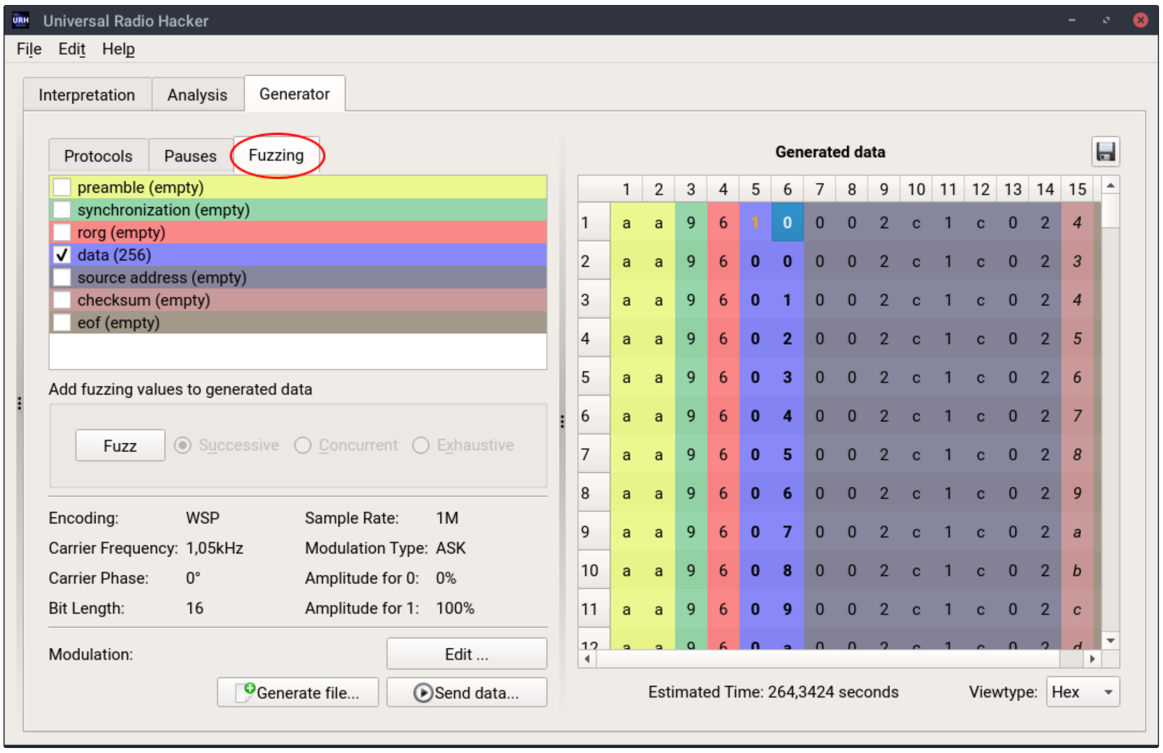
Figure 25: With the fuzzing dialog you can quickly add value ranges for labels.

In this example, we fuzz the label **data**. The bold **`10`** indicate the original value for this message. In the center table you define which values should be set for this label during fuzzing. With the buttons right to the table you can add values manually (➕), or remove (➖) and repeat (▤) selected values, respectively.

The controls at the bottom allow to automate the adding of values in the following ways:

- Add a **full range** of values from Start to End e.g. 0...255.
- Add **boundaries** only, e.g. $\{0, 1, 254, 255\}$ for Start $= 0$ End $= 255$, Border Values $= 2$.
- Add **random values** from specified range.

When you are done hit the Save and Close button and go to the Fuzzing tab in Generator as shown in fig. 26. Note, you configure the fuzzing settings **per message** so make sure you select the message you want to fuzz before making fuzzing settings. You see the number of fuzzing values behind the label name in this case 256. Make sure you check the label you want to fuzz to activate it.



- [ ] preamble (empty)
- [ ] synchronization (empty)
- [ ] rorg (empty)
- [✓] data (256)
- [ ] source address (empty)
- [ ] checksum (empty)
- [ ] eof (empty)

Add fuzzing values to generated data

Encoding:	WSP	Sample Rate:	1M
Carrier Frequency:	1,05kHz	Modulation Type:	ASK
Carrier Phase:	0°	Amplitude for 0:	0%
Bit Length:	16	Amplitude for 1:	100%

Modulation:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	a	a	9	6	1	0	0	0	2	c	1	c	0	2	4
2	a	a	9	6	0	0	0	0	2	c	1	c	0	2	3
3	a	a	9	6	0	1	0	0	2	c	1	c	0	2	4
4	a	a	9	6	0	2	0	0	2	c	1	c	0	2	5
5	a	a	9	6	0	3	0	0	2	c	1	c	0	2	6
6	a	a	9	6	0	4	0	0	2	c	1	c	0	2	7
7	a	a	9	6	0	5	0	0	2	c	1	c	0	2	8
8	a	a	9	6	0	6	0	0	2	c	1	c	0	2	9
9	a	a	9	6	0	7	0	0	2	c	1	c	0	2	a
10	a	a	9	6	0	8	0	0	2	c	1	c	0	2	b
11	a	a	9	6	0	9	0	0	2	c	1	c	0	2	c
12	a	a	9	6	0	a	0	0	2	c	1	c	0	2	d

Estimated Time: 264,3424 seconds

Viewtype: Hex

Figure 26: Fuzzing is as easy as clicking a button.

An active fuzzing label will be marked orange in the Generator table. Now everything is set up, just hit the Fuzz button and watch how URH generates all the messages. As can be seen from fig. 26 it also takes care of recalculating the checksum for each fuzzed message.

By default, URH adds a pause of one mega sample before each fuzzed message. You can change this via Options → Generation. Pauses between messages are itself configurable in the Pauses tab left to Fuzzing tab.

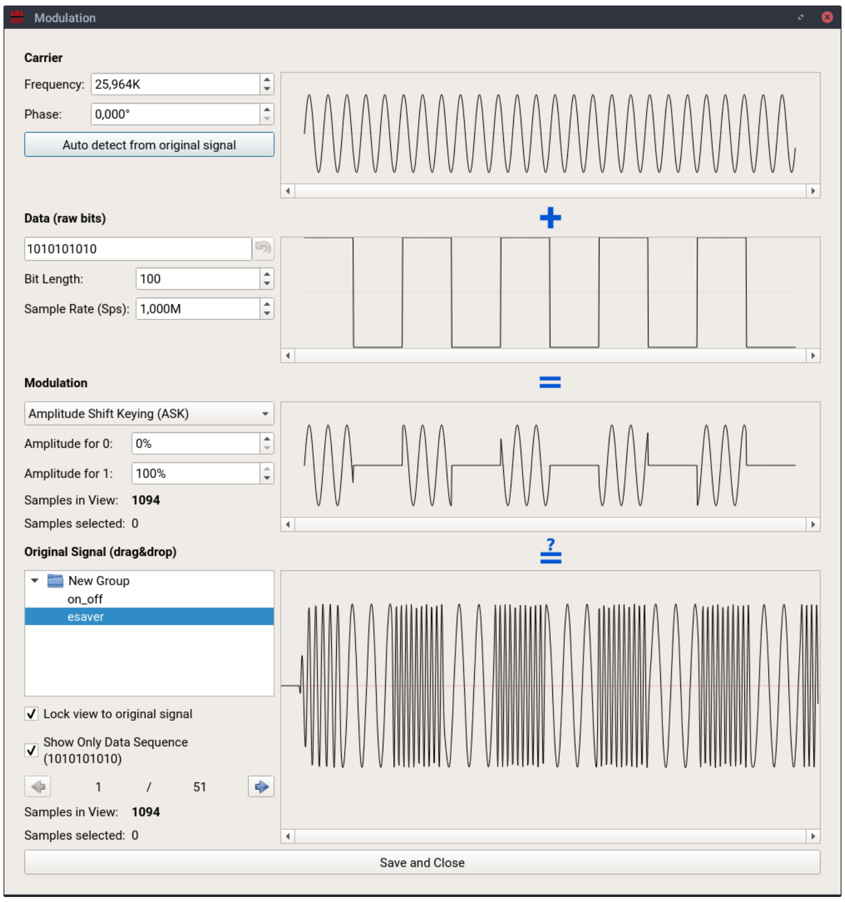
In case you want to fuzz different ranges within the same message, use the radio buttons next to the Fuzz button to select a mode to control how the values should be added:

- **Successive**: Fuzzed values are inserted **one-by-one**.
- **Concurrent**: Labels are fuzzed **at the same time**.
- **Exhaustive**: Fuzzed values are inserted in **all possible combinations**.

Note, you can **save** the current fuzzing profile with the save button at the top right corner.

5.2 Modulation

Now we have manipulated the data and can send it back to the air, right? Almost! It may be necessary to fine tune the modulation first. You can do this with the Edit button left to the generator table. Whenever you drop a protocol to Generator, URH will **automatically** determine reasonable modulation settings. Complicated modulations, however, may need some manual fine-tuning. The modulation dialog (fig. 27) which opens after clicking the Edit button helps you with this.



Frequency: 25,964K
Phase: 0,000°
1010101010
Bit Length: 100
Sample Rate (Sps): 1,000M
Modulation
Amplitude Shift Keying (ASK)
Amplitude for 0: 0%
Amplitude for 1: 100%
Samples in View: 1094
Samples selected: 0
✓ Lock view to original signal
✓ Show Only Data Sequence (1010101010)
1 / 51
Samples in View: 1094
Samples selected: 0

Figure 27: Fine-tune the modulation in the modulation dialog.

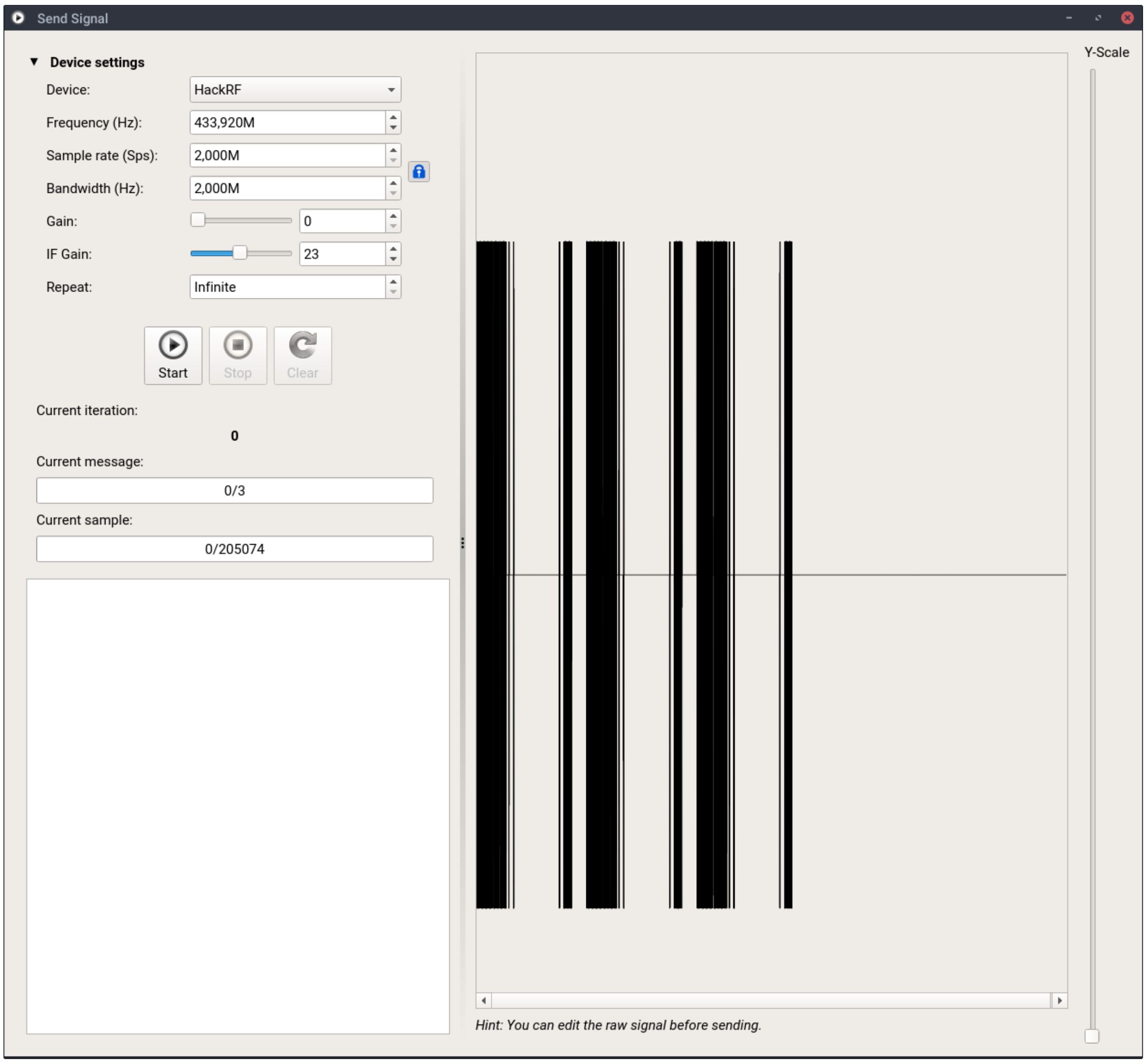
Simply verify all settings from top to bottom in this dialog. Each time you make a change the modulated preview will change. You can add an original signal via drag&drop from the tree view on the bottom and use the comboboxes to only show certain ranges that represent the customizable Data (raw bits). This way, you can visually verify your modulation settings by comparing the modulation result to the original signal.

You can review modulation settings in the area left of the generator table (see fig. 26) so you do not need to open modulation dialog for this.

Note, URH automatically applies the decoding chain (section 4.4) in reverted order before modulating the data, so you do not need to bother with encoding at this stage.

5.3 Sending it back

Having made all necessary configurations it is time to generate data. URH will take care of applying the encoding and modulating the bits. You can either use Generate file button to save this to a `.complex` file or hit the Send data button to open the send dialog (fig. 28).



Device: HackRF
Frequency (Hz): 433,920M
Sample rate (Sps): 2,000M
Bandwidth (Hz): 2,000M
Gain: 0
IF Gain: 23
Repeat: Infinite

Current iteration:
0
Current message:
0/3
Current sample:
0/205074

Hint: You can edit the raw signal before sending.

Figure 28: The send dialog will use the configured SDR to send out the data.

You see the modulated data on the right of the dialog. You can also edit the signal before sending using the context menu, if required. On the left side SDR related settings can be made. As soon as you hit the start button, send progress will be visualized in three ways. First, you see the current message in the progress bar on the right. Second, right below this the current sample is shown. Third, in the graphic view on the right an indicator will be drawn and updated during sending.

Could you trigger an action on the target device? Welcome to the circle of IoT hackers!

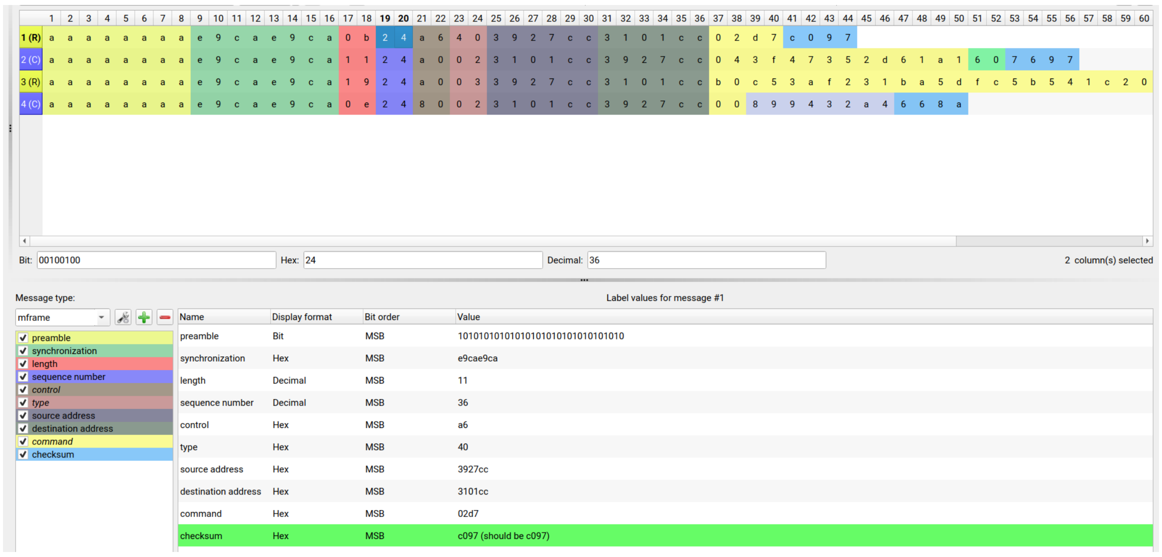
6 Simulation

There are protocols where it is not enough to fire a bunch of fuzzed packets against the device and hope to break them. More adavanced protocols have several states and use e.g. sequence numbers to refer to previous messages in a session. These data must be captured during interacting with the device and manipulated in a certain way e.g. increase a sequence number by one.

This is were the Simulator tab comes into play. Here you can configure label values to be learned during simulation time and even apply formulas to these values or get them from external programs. We will explore the features of the Simulator tab based on the challenge response procedure between a smart home central and a wireless remote control.

6.1 Getting started

First we start as usual by capturing messages with a Software Defined Radio and label the data in Analysis tab (see fig. 29). Of course, you can edit the labels later in the Simulator tab but the protocols from Analysis are useful for bootstrapping.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1 (R)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	0	b	2	4	a	6	4	0	3	9	2	7	c	c
2 (C)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	1	1	2	4	a	0	0	2	3	1	0	1	c	c
3 (R)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	1	9	2	4	a	0	0	3	3	9	2	7	c	c
4 (C)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	0	e	2	4	8	0	0	2	3	1	0	1	c	c

	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
1 (R)	3	1	0	1	c	c	0	2	d	7	c	0	9	7																
2 (C)	3	9	2	7	c	c	0	4	3	f	4	7	3	5	2	d	6	1	a	1	6	0	7	6	9	7				
3 (R)	3	1	0	1	c	c	b	0	c	5	3	a	f	2	3	1	b	a	5	d	f	c	5	b	5	4	1	c	2	0
4 (C)	3	9	2	7	c	c	0	0	8	9	9	4	3	2	a	4	6	6	8	a										

Bit: 00100100 Hex: 24 Decimal: 36 2 column(s) selected

Message type: mframe

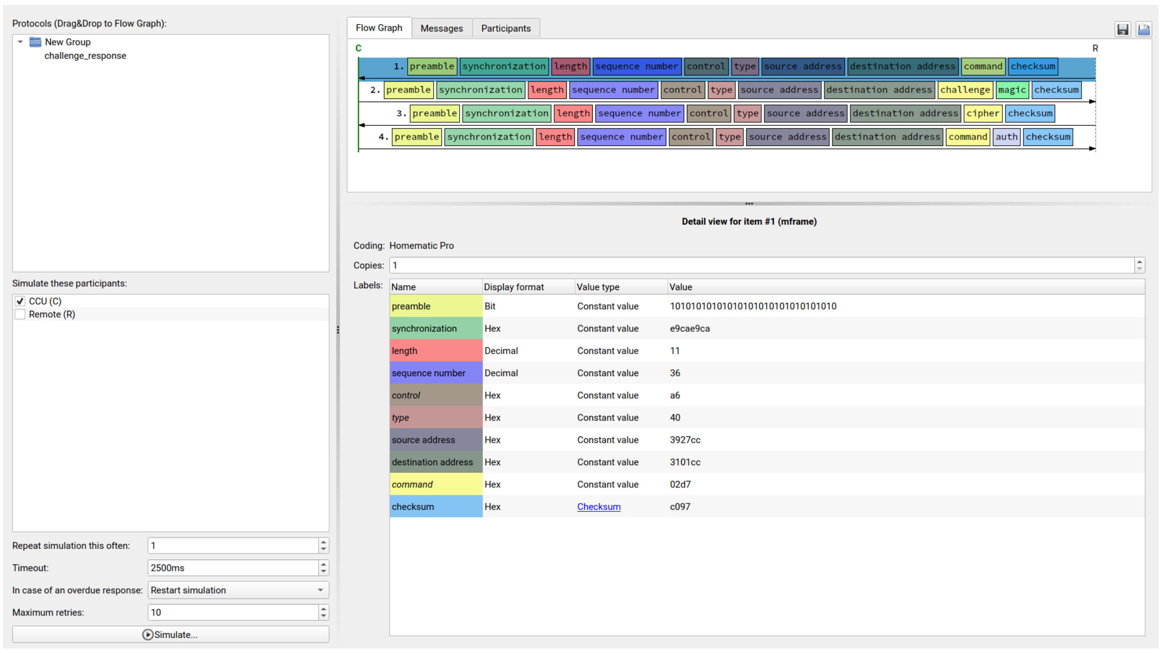
- ✓ preamble
- ✓ synchronization
- ✓ length
- ✓ sequence number
- ✓ *control*
- ✓ *type*
- ✓ source address
- ✓ destination address
- ✓ *command*
- ✓ checksum

Label values for message #1

Name	Display format	Bit order	Value
preamble	Bit	MSB	10101010101010101010101010101010
synchronization	Hex	MSB	e9cae9ca
length	Decimal	MSB	11
sequence number	Decimal	MSB	36
control	Hex	MSB	a6
type	Hex	MSB	40
source address	Hex	MSB	3927cc
destination address	Hex	MSB	3101cc
command	Hex	MSB	02d7
checksum	Hex	MSB	c097 (should be c097)

Figure 29: Labeled protocol in Analysis tab.

Now, move to the Simulator Tab and drop your protocol from the tree view on the left to the flow graph on the right. You will see the logical flow of your messages as shown in fig. 30.



1. preamble synchronization length sequence number control type source address destination address command checksum
2. preamble synchronization length sequence number control type source address destination address challenge magic checksum
3. preamble synchronization length sequence number control type source address destination address cipher checksum
4. preamble synchronization length sequence number control type source address destination address command auth checksum

Detail view for item #1 (mframe)

Coding: Homematic Pro

Copies: 1

Labels:

Name	Display format	Value type	Value
preamble	Bit	Constant value	10101010101010101010101010101010
synchronization	Hex	Constant value	e9cae9ca
length	Decimal	Constant value	11
sequence number	Decimal	Constant value	36
control	Hex	Constant value	a6
type	Hex	Constant value	40
source address	Hex	Constant value	3927cc
destination address	Hex	Constant value	3101cc
command	Hex	Constant value	02d7
checksum	Hex	Checksum	c097

Repeat simulation this often: 1

Timeout: 2500ms

In case of an overdue response: Restart simulation

Maximum retries: 10

Figure 30: Overview of Simulator tab.

6.2 Configuration options

6.2.1 Message direction (participants)

Next, we want to configure the flow graph to match our use case. Simply right click on a message in the flow graph to edit it's **source** and **destination**. You can choose any participant that you have defined in project settings. For convenience, you can also configure available participants in the **Participants** tab located on the most right of the **Flow Graph** tab.

6.2.2 Rules

Use the context menu of the flow graph to add **rules** (if then else). This allows you to model protocol state machines and e.g. send responses only when certain conditions are met.

6.2.3 Actions

Using the context menu (right click) of the flow graph you can add actions to arbitrary positions allowing you to deal with more complex scenarios:

- **Goto** enables to jump to arbitrary items in the flow graph. Simply choose the desired item from a dropdown menu.

- **Counter** actions increase their value by a (configurable) number each time they are hit during simulation. In combination with rules and Goto actions this enables stateful fuzzing.
- **Sleep** will pause the simulation for a selected time.
- **Trigger Command** enables you to call external programs so you can e.g. send an E-Mail as soon as simulation reaches a certain state.

6.2.4 Label configuration

Using the table on the bottom of the Simulator tab you can define how the values of labels shall be determined. You have five options for this:

- **Constant value**will match against the defined value of the label. If the value does not match during simulation the message will not be accepted. You can configure the value in the **Messages** tab on the right of the **Flow Graph** tab.
- **Live input** will insert the value of the label during simulation time. This is especially useful if a value is not known beforehand e.g. a sequence number.
- **Formula** allows you to calculate a value based on previous messages. You can write arbitrary python expressions here and also refer to previous label values, for example, with `item1.sequence_number + 1`. This will insert the value of the first message's sequence number (which may be a live input) incremented by one. This way to can refer to previous values and manipulate them as desired.
- **External program** is useful when you have an external program (e.g. AES encryption) that produces the desired value for a label. When called during simulation, your external program will get all previously received and sent messages for the certain participant. These messages will be passed to STDIN of your program.
- **Random value** will insert a random value from a customizable range.

6.3 Starting the simulation

Once everything is configured, you can click the Simulate... button on the bottom left of the Simulator tab. This will bring up the **simulation dialog** shown in fig. 31.

Before starting the simulation, you should configure a SDR for receiving and transmitting in the RX settings and TX settings tab, respectively.

During simulation, the currently captured signal will be shown under RX status so you can quickly debug RX issues. This capture is limited in size, if you want to capture everything simply check the Capture complete RX checkbox, but be aware that this will constantly use more and more RAM. With the save button on the right you can save the current capture to further investigate it in Interpretation.

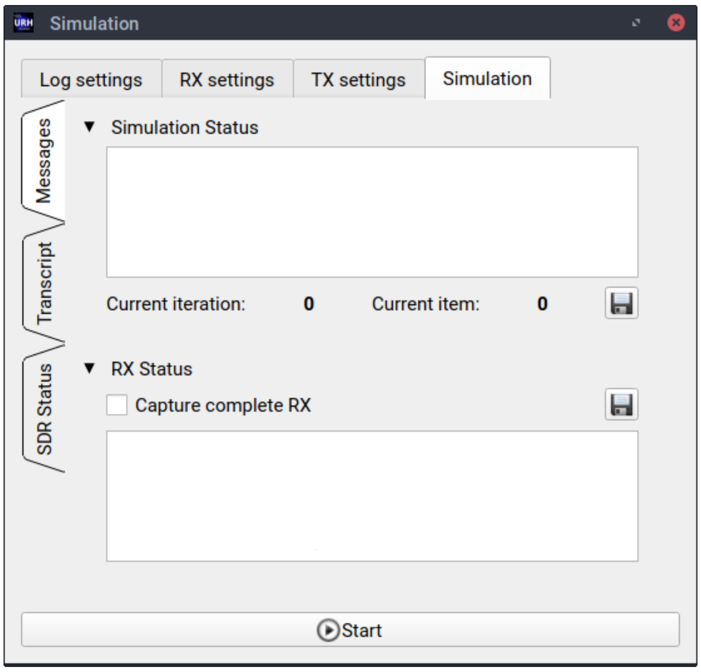
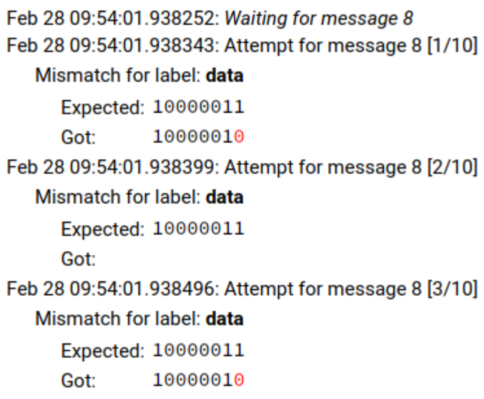


Figure 31: Simulation dialog.

To start the simulation use the Start button at the bottom of the dialog. During simulation, you will see a log what is currently happening and be informed e.g. about mismatching constant values like shown in fig. 32.



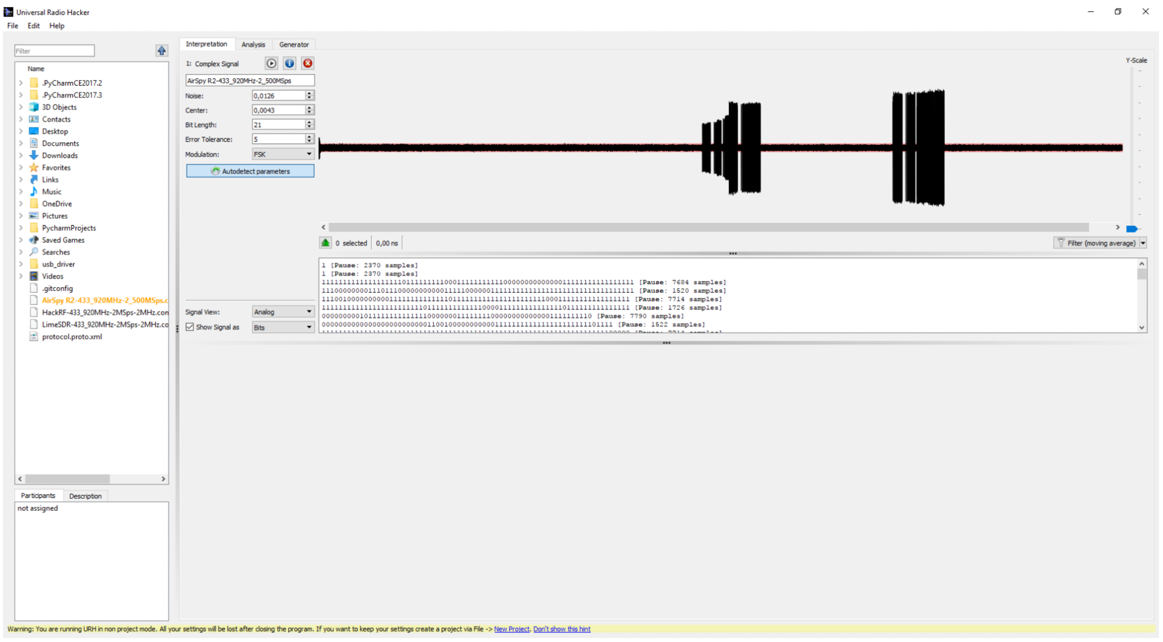
```
Feb 28 09:54:01.938252: Waiting for message 8
Feb 28 09:54:01.938343: Attempt for message 8 [1/10]
    Mismatch for label: data
        Expected: 10000011
        Got:      10000010
Feb 28 09:54:01.938399: Attempt for message 8 [2/10]
    Mismatch for label: data
        Expected: 10000011
        Got:
Feb 28 09:54:01.938496: Attempt for message 8 [3/10]
    Mismatch for label: data
        Expected: 10000011
        Got:      10000010
```

Figure 32: Example of simulator log.

After simulation, you will find the transcript in the Transcript tab located at the left of the Simulator tab. You can also save this transcript or open directly in Analysis for further investigations.

7 Configuring Look and Feel

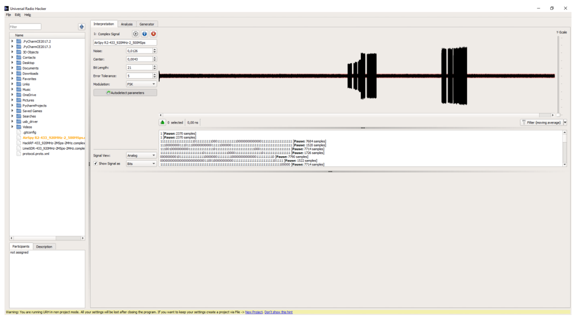
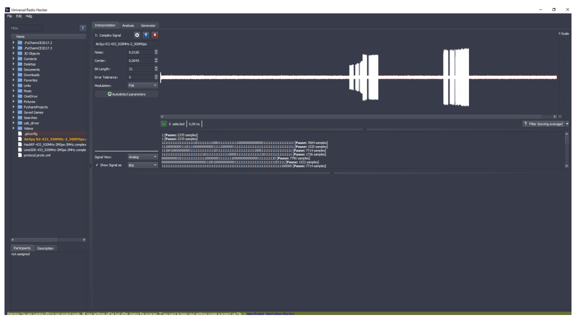
URH gives you a **native** look and feel regardless if you install it on Linux, Windows or OSX. However, if you would like to customize it there are two fallback themes available. Figure 33 gives you an impression how the three themes compare to each other on Windows.


Noise: 0,0126
Center: 0,0043
Bit Length: 21
Error Tolerance: 5
Modulation: FSK
Signal View: Analog
Show Signal as: Bits

```
1 [Pause: 2370 samples]
1 [Pause: 2370 samples]
11111111111111111110111111111100011111111111000000000000000111111111111111111 [Pause: 7684 samples]
1110000000011101110000000000001111100000011111111111111111111111111111111111 [Pause: 1520 samples]
11100100000000000111111111111101111111111111111111111110001111111111111111 [Pause: 7714 samples]
1111111111111111111111111011111111111111000011111111111111101111111111111111 [Pause: 1726 samples]
000000000101111111111111110000000111111110000000000000011111111110 [Pause: 7790 samples]
0000000000000000000000000110010000000000001111111111111111111111111101111 [Pause: 1522 samples]
```

Warning: You are running URH in non project mode. All your settings will be lost after closing the program. If you want to keep your settings create a project via File -> New Project. Don't show this hint

(a) Native theme on Windows.

(b) Light theme.

(c) Dark theme.

Figure 33: URH in the three different themes on Windows.

You can configure the theme using Edit → Options → View → Choose application theme.

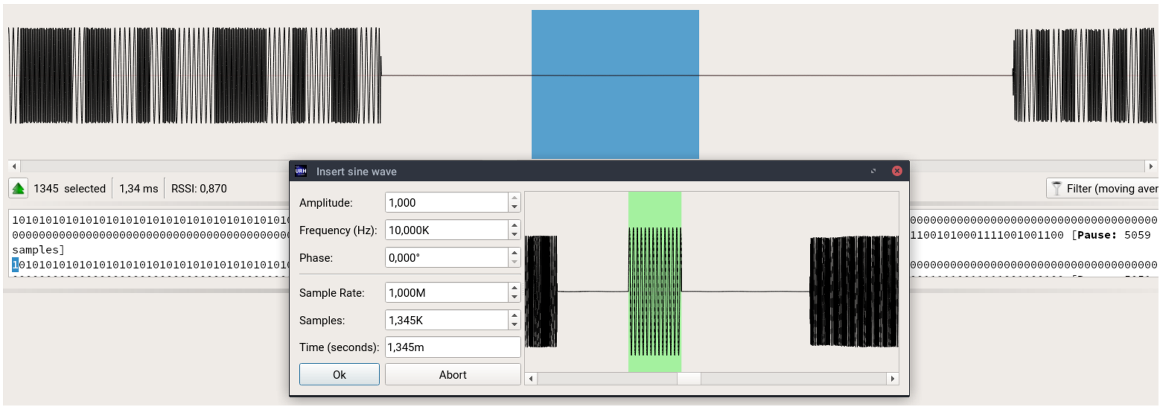
Linux users can also configure **icon theme**. This allows using system icons instead of the bundled icon theme that come with URH. You can choose this via Edit → Options → View → Choose icon theme.

Note to KDE users: If you experience icon issues in file dialogs it is necessary to switch the icon theme to native icon theme using Edit → Options → View → Choose icon theme.

8 Plugins

URH can be enhanced using plugins. To activate a plugin simply check it's checkbox using Edit → Options → Plugins. In this section you find an overview of available plugins.

- **ZeroHide**: This plugin allows you to entirely crop long sequences of zeros in your protocol and focus on relevant data. You can set a threshold below. All sequences of directly following zeros, which are longer than this threshold will be removed. This will give you a new entry in Analysis message table context menu.
- **RfCat**: With this plugin we support sending bytestreams via RfCat (e.g. using YARD Stick One). Therefore, a new button below generator table will be created.
- **InsertSine**: This plugin enables you to insert custom sine waves into your signal as shown in fig. 34. You will find a new context menu entry in Interpretation signal view. Transform URH into a full fledged signal editor!
- **NetworkSDRInterface**: With this plugin you can interface external applications using TCP. You can use your external application for performing protocol simulation on logical level or advanced modulation/decoding. If you activate this plugin, a new SDR will be selectable in device dialogs. Furthermore, a new button below generator table will be created.
- **MessageBreak**: This plugin enables you to break a protocol message on an arbitrary position. This is helpful when you have redundancy in your messages. This will give you a new entry in Analysis message table context menu.



Amplitude: 1,000
Frequency (Hz): 10,000K
Phase: 0,000°
Sample Rate: 1,000M
Samples: 1,345K
Time (seconds): 1,345m

```
101010101010101010101010101010101010101010
00000000000000000000000000000000000000000
samples]
101010101010101010101010101010101010101010
```

```
000000000000000000000000000000000000000
1100101000111100100110 0 [Pause: 5059
000000000000000000000000000000000000000
```

Figure 34: Insert arbitrary sine waves in Interpretation with the InsertSine plugin.