

[Home \(https://myriadr.org/\)](https://myriadr.org/) / [Blog \(https://myriadr.org/blog/\)](https://myriadr.org/blog/) / LimeSDR Made Simple Part 8: C Examples and SoapySDR API

✧ BLOG CATEGORIES ✧

LimeSDR Made Simple Part 8: C Examples and SoapySDR API

By Karl Woodward (<https://myriadr.org/blog/author/karlwoodward/>)

16th November, 2017

[limesdr \(/blog/tag/limesdr/\)](/blog/tag/limesdr/)

[limesuite \(/blog/tag/limesuite/\)](/blog/tag/limesuite/)



[\(https://myriadr.org/blog/limesdr-made-simple-part-8-c-examples-soapysdr-api/\)](https://myriadr.org/blog/limesdr-made-simple-part-8-c-examples-soapysdr-api/)

Welcome to the eighth instalment of the LimeSDR Made Simple series. Very early on in episode 1, we promised to work from SDR novice to API programming examples in small, bite sized chunks. Having done many of those steps it's now time to take the final one and start programming. For those who have read the whole series we have touched on programming before with GNU Octave, and in a way Pothos and GNU radio – so moving to direct API access should be a tiny step.

Programming in Linux (Ubuntu)

First we need a compiler, for this we will use gcc and given it's a staple of Linux it's likely you already have gcc installed. If not “sudo apt-get install gcc” should do the trick and get you the latest version.

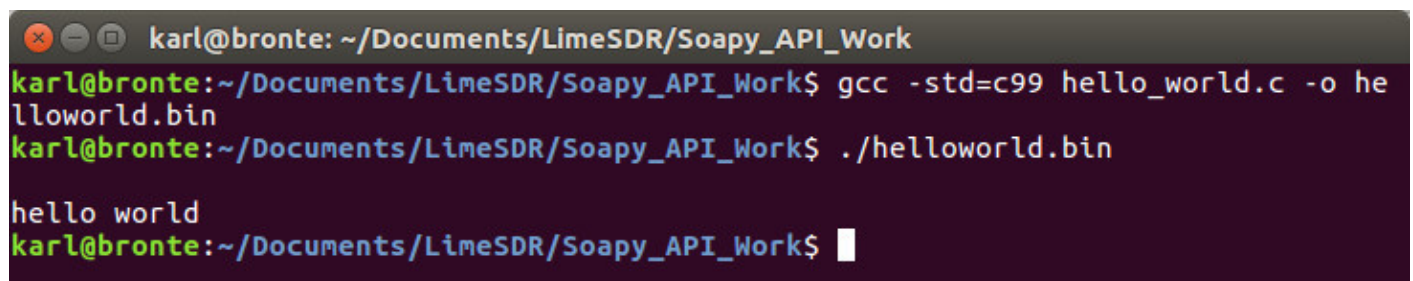
After this let's test our setup by creating a “hello world” application.

Create a file named “hello_world.c” by opening this file in your favourite editor and then add the following code:

```
#include // so we can use printf
int main()
{
printf("\nhello world\n"); // "\n" means new line
}
```

Save and close. Next we need to compile this so we use GCC via the command: `$ gcc -std=c99 hello_world.c -o helloworld.bin` This will create the file helloworld.bin, which is an executable program. If you do not specify a -o it will be named “a.out”. The -std=c99 specifies the “C” standard we are going to be using, which is C99 for the soapy examples.

Running helloworld.bin gives the output:



```
karl@bronte: ~/Documents/LimeSDR/Soapy_API_Work
karl@bronte:~/Documents/LimeSDR/Soapy_API_Work$ gcc -std=c99 hello_world.c -o helloworld.bin
karl@bronte:~/Documents/LimeSDR/Soapy_API_Work$ ./helloworld.bin

hello world
karl@bronte:~/Documents/LimeSDR/Soapy_API_Work$
```

(https://myriardf.org/app/uploads/2017/11/Soapy_API_Work_033_helloworld.jpg)

SoapySDR Application Programming Interface (API)

For the purposes of expedience we can look at the SoapySDR API as a black box. Much like the “printf” we just used, if we prod it with the right data it will do what we need. There are C++ and Python Variants of the API too if C is not your language of choice.

The API documentation is in the [Device.h](#)

(<https://github.com/pothosware/SoapySDR/blob/master/include/SoapySDR/Device.h>) header files

There are also examples on the SoapySDR website, the C example we will use as reference is [here](#) (https://github.com/pothosware/SoapySDR/wiki/C_API_Example).

To use these APIs SoapySDR must be installed. Linux is preferable, but Windows is possible too, although setting up compilers gets more complicated. If you have been following the series this should already be good to go as we have used SoapySDR many times before. If not the install instructions are [here](https://github.com/pothosware/PothosCore/wiki/Ubuntu) (<https://github.com/pothosware/PothosCore/wiki/Ubuntu>).

Getting started, download the C example and place it in a file “example.c”.

Compile it just like helloworld.c, but this time with the following gcc command :

```
$ gcc -std=c99 example.c -lSoapySDR -lm -o example.bin
```

We found that we needed to add the linker argument -lm for certain math functions later, but best we add this all the time to avoid confusion.

Running this will give the error: “SoapySDRDevice_make fail: RTL-SDR device not found.”

Oh no, this example is for an RTL-SDR!

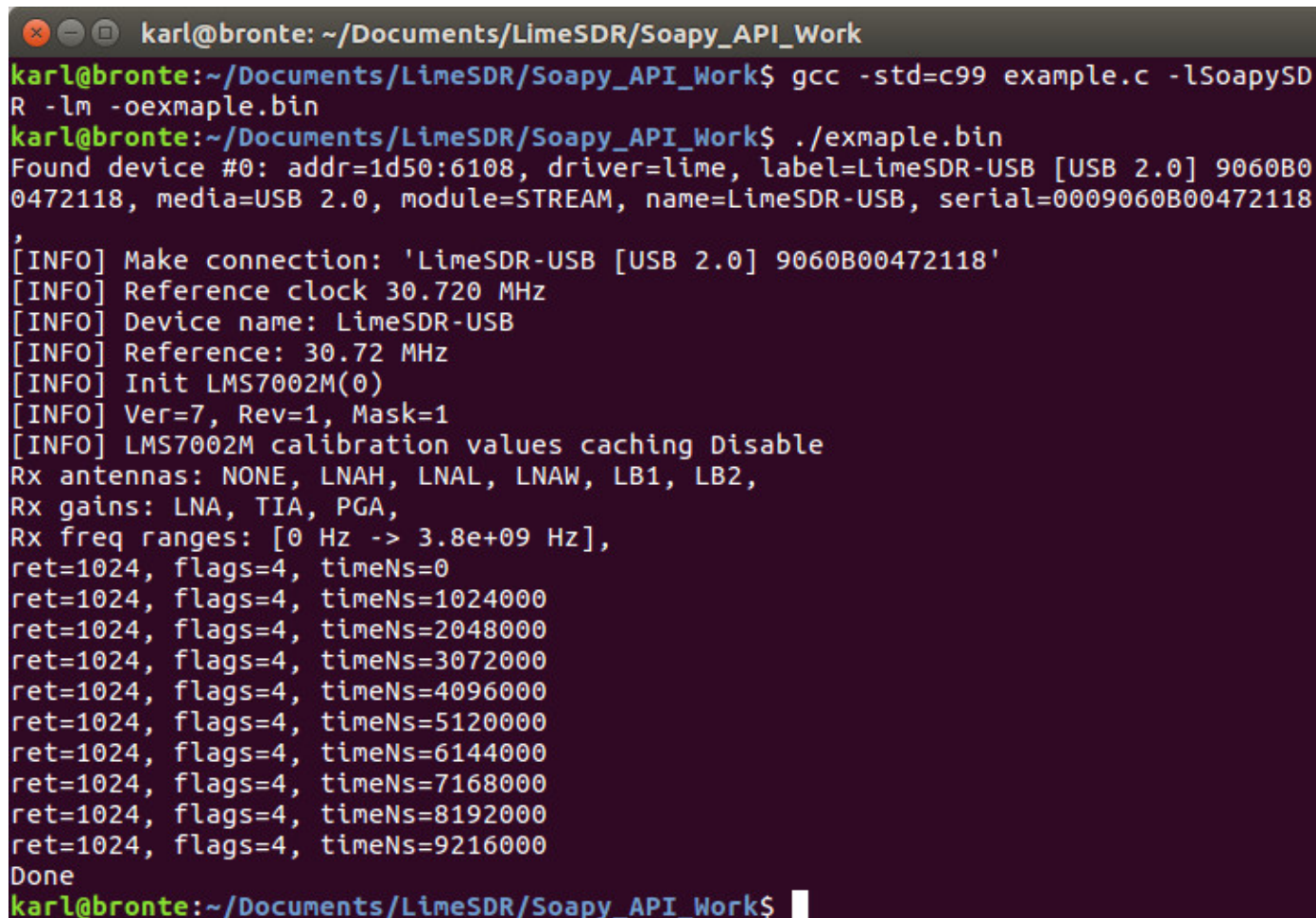
Do not fear as it’s a simple task to make this a LimeSDR example and just edit the following line:

```
“SoapySDRKwargs_set(&args, “driver”, “rtlsdr”);”
```

change to :

```
SoapySDRKwargs_set(&args, “driver”, “lime”);
```

Compile, run again and this time you should get the following:



```
karl@bronte: ~/Documents/LimeSDR/Soapy_API_Work
karl@bronte:~/Documents/LimeSDR/Soapy_API_Work$ gcc -std=c99 example.c -lSoapySDR -lm -oexample.bin
karl@bronte:~/Documents/LimeSDR/Soapy_API_Work$ ./example.bin
Found device #0: addr=1d50:6108, driver=lime, label=LimeSDR-USB [USB 2.0] 9060B00472118, media=USB 2.0, module=STREAM, name=LimeSDR-USB, serial=0009060B00472118
[INFO] Make connection: 'LimeSDR-USB [USB 2.0] 9060B00472118'
[INFO] Reference clock 30.720 MHz
[INFO] Device name: LimeSDR-USB
[INFO] Reference: 30.72 MHz
[INFO] Init LMS7002M(0)
[INFO] Ver=7, Rev=1, Mask=1
[INFO] LMS7002M calibration values caching Disable
Rx antennas: NONE, LNAH, LNAL, LNAW, LB1, LB2,
Rx gains: LNA, TIA, PGA,
Rx freq ranges: [0 Hz -> 3.8e+09 Hz],
ret=1024, flags=4, timeNs=0
ret=1024, flags=4, timeNs=1024000
ret=1024, flags=4, timeNs=2048000
ret=1024, flags=4, timeNs=3072000
ret=1024, flags=4, timeNs=4096000
ret=1024, flags=4, timeNs=5120000
ret=1024, flags=4, timeNs=6144000
ret=1024, flags=4, timeNs=7168000
ret=1024, flags=4, timeNs=8192000
ret=1024, flags=4, timeNs=9216000
Done
karl@bronte:~/Documents/LimeSDR/Soapy_API_Work$
```

(https://myriadr.org/app/uploads/2017/11/Soapy_API_Work_033_example.jpg)

That was easy!

Now we have a known working example, which reads the settings and serial number from our LimeSDR, and in addition 10 sets of 1024 samples.

The API documentation is quite self-explanatory; for example if we wish to alter this and add some gain, the gain types are already listed in the code and this is done with the command:

```
SoapySDRDevice_listGains(sdr, SOAPY_SDR_RX, 0, &length);
```

if we look at the API:

```

* List available amplification elements.
* Elements should be in order RF to baseband.
* \param device a pointer to a device instance
* \param direction the channel direction RX or TX
* \param channel an available channel
* \param [out] length the number of gain names
* \return a list of gain string names
*/
SOAPY_SDR_API char **SoapySDRDevice_listGains(const SoapySDRDevice *device,
const int direction, const size_t channel, size_t *length);

```

To adjust gain we search down and find “_setGain”. As can be seen from the API extract below this is almost identical to the listGain API, so adding a new command to set the gain is just as simple.

```

* Set the overall amplification in a chain.
*The gain will be distributed automatically across available element.
*\param device a pointer to a device instance
*\param direction the channel direction RX or TX
*\param channel an available channel on the device
*\param value the new amplification value in dB
*\return an error code or 0 for success
*/
SOAPY_SDR_API int SoapySDRDevice_setGain(SoapySDRDevice *device, const int
direction, const size_t channel, const double value);

```

Using this information we can set the API in the following way:

```
int success = SoapySDRDevice_setGain(sdr, SOAPY_SDR_RX, 0, 20)
```

This will add 20dB in the RX path 0. It really is that simple.

The whole API is documented in this way, so adding new commands is a cinch.

Getting organised

It's time to try and make a new tool and since we are starting small, a site survey tool is a good choice. For this we will read 1024 samples at each frequency and calculate an RMS value for these. We will be able to specify a start and stop frequency, and the number of Hz between tune attempts.

First things first before we get started: code is not the natural habitat of the hardware engineer, we find it hard to read whole walls of code so let's split this example up so we can have a smaller main loop and maintaining the code is easier. This requires the use of functions and pointers, so it may be a good time to brush up on your basic C skills (we have to admit we needed some practice too).

We'll make a few functions (these are the prototypes):

```
struct SoapySDRDevice *Setup(void);
void DeviceInfo();
void Read_1024_samples(struct SoapySDRDevice *sdr, double freq, complex
float *buffer);
void Close();
```

SoapySDRDevice

This is a relatively complex function, which returns both a SoapySDRDevice structure and a pointer to the SDR device created. This pointer is what all of the other functions use and how they interact with the LimeSDR.

It is called like: `struct SoapySDRDevice *sdr = Setup();`

```
struct SoapySDRDevice *Setup(void)
{
//enumerate devices
SoapySDRKwargs *results = SoapySDRDevice_enumerate(NULL, &length);
for (size_t i = 0; i < length; i++)
{
printf("Found device #%d: ", (int)i);
for (size_t j = 0; j < results[i].size; j++)
{
printf("%s=%s, ", results[i].keys[j], results[i].vals[j]);
}
printf("\n");
}
SoapySDRKwargsList_clear(results, length);
//create device instance
//args can be user defined or from the enumeration result
```

```

SoapySDRKwargs args = {};
SoapySDRKwargs_set(&args, "driver", "lime");
SoapySDRDevice *sdr = SoapySDRDevice_make(&args);
SoapySDRKwargs_clear(&args);
if(sdr == NULL)
{
printf("SoapySDRDevice_make fail: %s\n", SoapySDRDevice_lastError());
//return EXIT_FAILURE;
}
return sdr;
}

```

DeviceInfo

A simple function that returns the information data from the example, it is called by passing the SDR device into it: `DeviceInfo(sdr);`

```

void DeviceInfo(struct SoapySDRDevice *sdr)
{
//query device info
char** names = SoapySDRDevice_listAntennas(sdr, SOAPY_SDR_RX, 0, &length);
printf("Rx antennas: ");
for (size_t i = 0; i < length; i++) printf("%s, ", names[i]);
printf("\n");
SoapySDRStrings_clear(&names, length);
names = SoapySDRDevice_listGains(sdr, SOAPY_SDR_RX, 0, &length);
printf("Rx gains: ");
for (size_t i = 0; i < length; i++) printf("%s, ", names[i]);
printf("\n");
SoapySDRStrings_clear(&names, length);
SoapySDRRange *ranges = SoapySDRDevice_getFrequencyRange(sdr, SOAPY_SDR_RX,
0, &length);
printf("Rx freq ranges: ");
for(size_t i = 0; i < length; i++) printf("[%g Hz -> %g Hz], ",
ranges[i].minimum, ranges[i].maximum);
printf("\n");
free(ranges);
}

```


Read_1024_samples

At first this function looks terrifying. It is called by the following: `Read_1024_samples(sdr, frequency, buffn);`

`sdr` is the device made in `Setup()`, `frequency` is the tune frequency and finally `buffn` is the buffer to receive the samples read. These are passed back to the `buffn` variable via pointers.

We also took the liberty of adding some error checking in this function so we can avoid accidentally tuning out of range. In addition we added some gain and forced LNAL as the receive antenna.

```
void Read_1024_samples(struct SoapySDRDevice *sdr, double freq, complex
float *buffer)
{
// check freq is within range
if (freq <10e3)
{
freq = 10e3;
printf("Frequency out of range setting min: \n");
}
if (freq >3.8e9)
{
freq = 3.8e9;
printf("Frequency out of range setting max: \n");
}
//apply settings
if (SoapySDRDevice_setSampleRate(sdr, SOAPY_SDR_RX, 0, 1e6) != 0)
{
printf("setSampleRate fail: %s\n", SoapySDRDevice_lastError());
}
if (SoapySDRDevice_setAntenna(sdr, SOAPY_SDR_RX, 0, "LNAL") != 0)
{
printf("setAntenna fail: %s\n", SoapySDRDevice_lastError());
}
if (SoapySDRDevice_setGain(sdr, SOAPY_SDR_RX, 0, 20) != 0)
{
printf("setAntenna fail: %s\n", SoapySDRDevice_lastError());
}
if (SoapySDRDevice_setFrequency(sdr, SOAPY_SDR_RX, 0, freq, NULL) != 0)
```



```

{
printf("setFrequency fail: %s\n", SoapySDRDevice_lastError());
}
//setup a stream (complex floats)
SoapySDRStream *rxStream;
if (SoapySDRDevice_setupStream(sdr, &rxStream, SOAPY_SDR_RX, SOAPY_SDR_CF32,
NULL, 0, NULL) != 0)
{
printf("setupStream fail: %s\n", SoapySDRDevice_lastError());
}
SoapySDRDevice_activateStream(sdr, rxStream, 0, 0, 0); //start streaming
void *buffs[] = {buffer}; //array of buffers
int flags; //flags set by receive operation
long long timeNs; //timestamp for receive buffer
int ret = SoapySDRDevice_readStream(sdr, rxStream, buffs, 1024, &flags,
&timeNs, 100000);
//shutdown the stream
SoapySDRDevice_deactivateStream(sdr, rxStream, 0, 0); //stop streaming
SoapySDRDevice_closeStream(sdr, rxStream);
}

```

Close

Finally we need a way of closing the SDR device we opened in Setup. This is done by calling `close(sdr)`;

```

void Close(struct SoapySDRDevice *sdr2)
{
//cleanup device handle
SoapySDRDevice_unmake(sdr2);
printf("Done\n");
//return EXIT_SUCCESS;
}

```

A minimal application

Spring cleaning done, we now have 90% of our application and here would be a minimum example. We need the following includes:

```
#include <SoapySDR/Device.h>
#include <SoapySDR/Formats.h>
#include <stdio.h> //printf
#include <stdlib.h> //free
#include <complex.h>
```

There is a single global:

```
size_t length;
```

The Main contains only these 3 lines:

```
struct SoapySDRDevice *sdr = Setup();
DeviceInfo(sdr);
Close(sdr);
```

A site survey tool

At this point it should be a simple matter of adding a little code and we can create quite a few applications. Here is the site survey example:

In essence we get passed 3 arguments from the command line: Start freq, Stop freq and interval. The code then sets up the lime SDR, calculates the number of samples required. From this information the code reads each frequency for 1024 samples, does an RMS calculation on these. $RMS = \sqrt{(a^2 + b^2 + c^2) / \text{num samples}}$. This is then printed to the console. Note the Sqrt command is what the -lm linker argument during compile is for, this may not be needed on your system.

```
int main(int argc, char *argv[])
{
    int argvcount=0;
    if(argc!=4)
    {
        printf("%d",argc);
        printf("invalid input");
        exit(EXIT_FAILURE);
    }
    struct SoapySDRDevice *sdr = Setup();
    DeviceInfo(sdr);
    atexit(Cleanup);
```

```
// HERE WE CAN DO THINGS
// Lets do a site survey measuring every xxx Hz and displaying the power
double StartFreq = strtod(argv[1],NULL);
double EndFreq = strtod(argv[2],NULL);
double Interval =strtod(argv[3],NULL);
double NumberOfScanPoints= (EndFreq-StartFreq)/Interval;
printf("Starting RMS scan, Start is: %f, End is: %f, Interval is:
%f\n",StartFreq,EndFreq,Interval);
printf("This gives a total of: %f Sample points",NumberOfScanPoints);
while (NumberOfScanPoints !=0)
{
Get_Samples(sdr);
//create a re-usable buffer for rx samples
complex float buffn[1024];
double frequency = StartFreq + (Interval * ((EndFreq/Interval)-
NumberOfScanPoints));
Read_1024_samples(sdr, frequency,buffn);
//printf("ret=%d, flags=%d, timeNs=%lld\n", ret, flags, timeNs);
float realavg = 0;
for (int x = 0; x < 1024; x++)
{
printf("%f + i%f\n", creal(buffn[x]), cimag(buffn[x]));
realavg= realavg+ (creal(buffn[x])*creal(buffn[x]));
}
realavg = sqrt(realavg/1024);
printf("Freq is: %f, RMS is: %f \n",frequency,realavg);
NumberOfScanPoints--;
}
Close(sdr);
}
```

Having compiled and run this program the output looks like:

```

karl@bronte:~/Documents/LimeSDR/Soapy_API_Work$ gcc -std=c99 soapy_api.c -lSoapy
SDR -lm -osoapy_api.bin
karl@bronte:~/Documents/LimeSDR/Soapy_API_Work$ ./soapy_api.bin 1e6 2e6 100e3
Found device #0: addr=1d50:6108, driver=lime, label=LimeSDR-USB [USB 2.0] 9060B0
0472118, media=USB 2.0, module=STREAM, name=LimeSDR-USB, serial=0009060B00472118
,
[INFO] Make connection: 'LimeSDR-USB [USB 2.0] 9060B00472118'
[INFO] Reference clock 30.720 MHz
[INFO] Device name: LimeSDR-USB
[INFO] Reference: 30.72 MHz
[INFO] Init LMS7002M(0)
[INFO] Ver=7, Rev=1, Mask=1
[INFO] LMS7002M calibration values caching Disable
Rx antennas: NONE, LNAH, LNAL, LNAW, LB1, LB2,
Rx gains: LNA, TIA, PGA,
Rx freq ranges: [0 Hz -> 3.8e+09 Hz],
Starting RMS scan, Start is: 1000000.000000, End is: 2000000.000000, Interval is
: 100000.000000
This gives a total of: 10.000000 Sample pointsFreq is: 2000000.000000, RMS is: 0
.002227
Freq is: 2100000.000000, RMS is: 0.001122
Freq is: 2200000.000000, RMS is: 0.000579
Freq is: 2300000.000000, RMS is: 0.002567
Freq is: 2400000.000000, RMS is: 0.001817
Freq is: 2500000.000000, RMS is: 0.000418
Freq is: 2600000.000000, RMS is: 0.002236
Freq is: 2700000.000000, RMS is: 0.000648
Freq is: 2800000.000000, RMS is: 0.001009
Freq is: 2900000.000000, RMS is: 0.001249
Done
karl@bronte:~/Documents/LimeSDR/Soapy_API_Work$

```

(https://myriadr.org/app/uploads/2017/11/Soapy_api.c.jpg)

Adding a choice of LNA or gain settings at the command line is trivial matter. Saving the output to a file is simple in Linux just run with the command: `./soapy_api.bin 1e6 2e6 100e3 >output.txt`

Final words

C is a very powerful language and making a more powerful application at this point is a matter of code. Decoding, encoding, transmitting and receiving are all possible and just as easy to access as the example above.

We hope this short foray into the SoapySDR API will give inspiration and using it we are not limited to blocks in Pothos or GNU radio; we can create whole new applications using your language of choice (assuming you choose C, C++ or python).

Karl Woodward

Leave a Reply