Welcome to HackRfDotnet

HackRf Dotnet is the only complete and cross platform hackrf api wrapper for C# Dotnet.

Getting Started With HackRf Dotnet

The most basic way we can use HackRfDotnet is by playing an FM audio stream.

```
    var controllerService = new RfDeviceControllerService();
    Console.WriteLine("looking for HackRf Device...");

    var deviceList = controllerService.FindDevices();
    Console.WriteLine($"Found {deviceList.devicecount} HackRf devices... Opening Rx");

    using var rfDevice = controllerService.ConnectToFirstDevice();
    if (rfDevice is null) {
        Console.WriteLine("Could not connect to Rf Device");
        return;
    }

    // Make an IQDeviceStream from the rf device.
    // this immutable stream will stem all of your signal processing pipelines with data.
    using var deviceStream = new IQDeviceStream(rfDevice);

    // Open Rx from the radio device at 20 Mega Samples Per Second.
    deviceStream.OpenRx(SampleRate.FromMSps(20));

    // Set the radio frequency channel amd bandwidth.
    rfDevice.SetFrequency(RadioBand.FromMHz(98.7f), RadioBand.FromKHz(200));

    // We must build an effects pipeline to clean up our received signal from the SDR.
    var effectsPipeline = new SignalProcessingBuilder()
        // DownSampleEffect decimates your signal down to it's bandwidth.
        // Since our signal has been frequency shifted by the SDR mixer
        // our target frequency has been shifted to Direct Current (DC).
        // Meaning we don't need any more sample rate than the band of the signal to
represent it in the time domain,
        // so we reduce it's extraneous information
        .AddSignalEffect(new DownSampleEffect(deviceStream.SampleRate,
            rfDevice.Bandwidth.NyquistSampleRate, out var reducedSampleRate, out
var producedChunkSize))

        // Fast Fourier Transform from the Time domain signal to the Frequency domain
        .AddSignalEffect(new FftEffect(true, producedChunkSize))

        // Low pass filter our band (Since we are mixed to DC, we only need to low pass
filter the signal it gets affected on + and -)
        .AddSignalEffect(new LowPassFilterEffect(reducedSampleRate, rfDevice.Bandwidth))
```

```
        // Inverse Fast Fourier Transform from the Frequency domain back to the Time domain.
        .AddSignalEffect(new FftEffect(false, producedChunkSize))

        // Compile our effect pipeline
        .BuildPipeline();

    // Create a signal stream and configure it with our effects pipeline,
    // it will allow us to read from it as a stream with pre-demodulated results, like
a StreamReader
    var fmSignalStream = new FmSignalStream(deviceStream, reducedSampleRate, stereo: true,
        processingPipeline: effectsPipeline, keepOpen: false);

    // And AnaloguePlayer let's us resample and pipe an audio out the speakers.
    var fmPlayer = new AnaloguePlayer(fmSignalStream);
    fmPlayer.PlayStreamAsync(rfDevice.Frequency, rfDevice.Bandwidth, 48000);
```

Signal Processing Pipelines

A Signal Processing Pipeline, is a method of applying a chain of effects onto a signal from the SignalStream. A SignalStream is a stream piped from the IQDeviceStream Without a Signal Processing Pipeline you will get the full capture data.