

# Namespace HackRFDotnet.Api

## Classes

[DigitalRadioDevice](#)

# Class DigitalRadioDevice

Namespace: [HackRFDotnet.Api](#)

Assembly: HackRFDotnet.dll

```
public class DigitalRadioDevice : IDisposable
```

## Inheritance

[object](#) ← DigitalRadioDevice

## Implements

[IDisposable](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Extension Methods

[RfDeviceExtensions.AttenuateAmplification\(DigitalRadioDevice\)](#)

## Fields

### DevicePtr

```
public readonly HackRFDevice* DevicePtr
```

### Field Value

[HackRFDevice](#)\*

## Properties

### Bandwidth

```
public RadioBand Bandwidth { get; set; }
```

Property Value

[RadioBand](#)

## DeviceSamplingRate

```
public SampleRate DeviceSamplingRate { get; set; }
```

Property Value

[SampleRate](#)

## Frequency

```
public RadioBand Frequency { get; set; }
```

Property Value

[RadioBand](#)

## IsConnected

```
public bool IsConnected { get; }
```

Property Value

[bool](#) 

## Methods

### Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

## SetAmplifications(uint, uint, bool)

```
public void SetAmplifications(uint lna, uint vga, bool internalAmp)
```

### Parameters

lna [uint](#)

vga [uint](#)

internalAmp [bool](#)

## SetFrequency(RadioBand)

```
public bool SetFrequency(RadioBand radioFrequency)
```

### Parameters

radioFrequency [RadioBand](#)

### Returns

[bool](#)

## SetFrequency(RadioBand, RadioBand)

```
public bool SetFrequency(RadioBand radioFrequency, RadioBand bandwidth)
```

### Parameters

radioFrequency [RadioBand](#)

bandwidth [RadioBand](#)

Returns

[bool](#)

## SetSampleRate(SampleRate)

```
public void SetSampleRate(SampleRate sampleRate)
```

Parameters

sampleRate [SampleRate](#)

## StartRx(HackRFSampleBlockCallback)

```
public bool StartRx(HackRFSampleBlockCallback rxCallback)
```

Parameters

rxCallback [HackRFSampleBlockCallback](#)

Returns

[bool](#)

## StopRx()

```
public bool StopRx()
```

Returns

[bool](#)

# Namespace HackRFDotnet.Api.Extensions

## Classes

[RfDeviceExtensions](#)

# Class RfDeviceExtensions

Namespace: [HackRFDotnet.Api.Extensions](#)

Assembly: HackRFDotnet.dll

```
public static class RfDeviceExtensions
```

## Inheritance

[object](#) ← RfDeviceExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

### AttenuateAmplification(DigitalRadioDevice)

```
public static void AttenuateAmplification(this DigitalRadioDevice rfDevice)
```

#### Parameters

rfDevice [DigitalRadioDevice](#)

# Namespace HackRFDotnet.Api.Services

## Classes

[AnaloguePlayer](#)

[DigitalPlayer](#)

[RfDeviceControllerService](#)

# Class AnaloguePlayer

Namespace: [HackRFDotnet.Api.Services](#)

Assembly: HackRFDotnet.dll

```
public class AnaloguePlayer : IDisposable
```

## Inheritance

[object](#) ← AnaloguePlayer

## Implements

[IDisposable](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

### AnaloguePlayer(WaveSignalStream)

```
public AnaloguePlayer(WaveSignalStream signalStream)
```

## Parameters

signalStream [WaveSignalStream](#)

## Methods

### Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

## PlayStreamAsync(RadioBand, RadioBand, int)

```
public virtual void PlayStreamAsync(RadioBand centerOffset, RadioBand bandwidth,  
int audioRate)
```

### Parameters

centerOffset [RadioBand](#)

bandwidth [RadioBand](#)

audioRate [int](#)

# Class DigitalPlayer

Namespace: [HackRFDotnet.Api.Services](#)

Assembly: HackRFDotnet.dll

```
public class DigitalPlayer : IDisposable
```

## Inheritance

[object](#) ← DigitalPlayer

## Implements

[IDisposable](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

### DigitalPlayer(HdRadioSignalStream)

```
public DigitalPlayer(HdRadioSignalStream sampleDeModulator)
```

## Parameters

sampleDeModulator [HdRadioSignalStream](#)

## Methods

### Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

## PlayStreamAsync(RadioBand, RadioBand, int)

```
public virtual void PlayStreamAsync(RadioBand centerOffset, RadioBand bandwidth,  
int audioRate)
```

### Parameters

centerOffset [RadioBand](#)

bandwidth [RadioBand](#)

audioRate [int](#)

# Class RfDeviceControllerService

Namespace: [HackRFDotnet.Api.Services](#)

Assembly: HackRFDotnet.dll

```
public class RfDeviceControllerService
```

## Inheritance

[object](#) ← RfDeviceControllerService

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

### RfDeviceControllerService()

```
public RfDeviceControllerService()
```

## Fields

### RfDevices

```
public readonly List<DigitalRadioDevice> RfDevices
```

Field Value

[List](#)<[DigitalRadioDevice](#)>

## Methods

### ConnectToFirstDevice()

```
public DigitalRadioDevice? ConnectToFirstDevice()
```

Returns

[DigitalRadioDevice](#)

## FindDevices()

```
public HackRFDeviceList FindDevices()
```

Returns

[HackRFDeviceList](#)

# Namespace HackRFDotnet.Api.Streams

## Classes

[SweepingIQStream](#)

## Structs

[IQ](#)

[InterleavedSample](#)

# Struct IQ

Namespace: [HackRFDotnet.Api.Streams](#)

Assembly: HackRFDotnet.dll

```
public struct IQ : IEquatable<IQ>, IFormattable
```

Implements

[IEquatable](#)<[IQ](#)>, [IFormattable](#)

Inherited Members

[object.Equals\(object, object\)](#), [object.GetType\(\)](#), [object.ReferenceEquals\(object, object\)](#)

## Constructors

### [IQ\(InterleavedSample\)](#)

```
public IQ(InterleavedSample interleavedSample)
```

Parameters

interleavedSample [InterleavedSample](#)

### [IQ\(float, float\)](#)

```
public IQ(float real, float imaginary)
```

Parameters

real [float](#)

imaginary [float](#)

## Fields

## ImaginaryOne

```
public static readonly IQ ImaginaryOne
```

Field Value

[IQ](#)

## Infinity

```
public static readonly IQ Infinity
```

Field Value

[IQ](#)

## NaN

```
public static readonly IQ NaN
```

Field Value

[IQ](#)

## One

```
public static readonly IQ One
```

Field Value

[IQ](#)

## Zero

```
public static readonly IQ Zero
```

Field Value

[IQ](#)

## Properties

|

Real

```
public float I { get; set; }
```

Property Value

[float](#) ↗

## Magnitude

```
public float Magnitude { get; }
```

Property Value

[float](#) ↗

## Phase

```
public float Phase { get; }
```

Property Value

[float](#) ↗

Q

Imaginary

```
public float Q { get; set; }
```

Property Value

[float](#)

## Methods

Abs(IQ)

```
public static float Abs(IQ value)
```

Parameters

[value](#) [IQ](#)

Returns

[float](#)

Acos(IQ)

```
public static IQ Acos(IQ value)
```

Parameters

[value](#) [IQ](#)

Returns

[IQ](#)

## Add(IQ, IQ)

```
public static IQ Add(IQ left, IQ right)
```

Parameters

**left** [IQ](#)

**right** [IQ](#)

Returns

[IQ](#)

## Add(IQ, float)

```
public static IQ Add(IQ left, float right)
```

Parameters

**left** [IQ](#)

**right** [float](#)

Returns

[IQ](#)

## Add(float, IQ)

```
public static IQ Add(float left, IQ right)
```

Parameters

**left** [float](#)

**right** [IQ](#)

Returns

[IQ](#)

## Asin(IQ)

```
public static IQ Asin(IQ value)
```

Parameters

value [IQ](#)

Returns

[IQ](#)

## Atan(IQ)

```
public static IQ Atan(IQ value)
```

Parameters

value [IQ](#)

Returns

[IQ](#)

## Conjugate(IQ)

```
public static IQ Conjugate(IQ value)
```

Parameters

value [IQ](#)

Returns

[IQ](#)

## Cos(IQ)

```
public static IQ Cos(IQ value)
```

Parameters

value [IQ](#)

Returns

[IQ](#)

## Cosh(IQ)

```
public static IQ Cosh(IQ value)
```

Parameters

value [IQ](#)

Returns

[IQ](#)

## Divide(IQ, IQ)

```
public static IQ Divide(IQ dividend, IQ divisor)
```

Parameters

dividend [IQ](#)

`divisor` [IQ](#)

Returns

[IQ](#)

## Divide(IQ, float)

```
public static IQ Divide(IQ dividend, float divisor)
```

Parameters

`dividend` [IQ](#)

`divisor` [float](#)

Returns

[IQ](#)

## Divide(float, IQ)

```
public static IQ Divide(float dividend, IQ divisor)
```

Parameters

`dividend` [float](#)

`divisor` [IQ](#)

Returns

[IQ](#)

## Equals(IQ)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(IQ value)
```

Parameters

**value** [IQ](#)

Returns

[bool](#)

[true](#) if the current object is equal to the **other** parameter; otherwise, [false](#).

## Equals(object?)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object? obj)
```

Parameters

**obj** [object](#)

The object to compare with the current instance.

Returns

[bool](#)

[true](#) if **obj** and this instance are the same type and represent the same value; otherwise, [false](#).

## Exp(IQ)

```
public static IQ Exp(IQ value)
```

Parameters

**value** [IQ](#)

Returns

[IQ](#)

## FromPolarCoordinates(float, float)

```
public static IQ FromPolarCoordinates(float magnitude, float phase)
```

Parameters

**magnitude** [float](#)

**phase** [float](#)

Returns

[IQ](#)

## GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## IsFinite(IQ)

```
public static bool IsFinite(IQ value)
```

Parameters

`value` [IQ](#)

Returns

[bool](#) ↗

## IsInfinity(IQ)

```
public static bool IsInfinity(IQ value)
```

Parameters

`value` [IQ](#)

Returns

[bool](#) ↗

## IsNaN(IQ)

```
public static bool IsNaN(IQ value)
```

Parameters

`value` [IQ](#)

Returns

[bool](#) ↗

## Log(IQ)

```
public static IQ Log(IQ value)
```

Parameters

`value` [IQ](#)

Returns

[IQ](#)

## Log(IQ, float)

```
public static IQ Log(IQ value, float baseValue)
```

Parameters

`value` [IQ](#)

`baseValue` [float](#)

Returns

[IQ](#)

## Log10(IQ)

```
public static IQ Log10(IQ value)
```

Parameters

`value` [IQ](#)

Returns

[IQ](#)

## Multiply(IQ, IQ)

```
public static IQ Multiply(IQ left, IQ right)
```

Parameters

**left** [IQ](#)

**right** [IQ](#)

Returns

[IQ](#)

## Multiply(IQ, float)

```
public static IQ Multiply(IQ left, float right)
```

Parameters

**left** [IQ](#)

**right** [float](#)↗

Returns

[IQ](#)

## Multiply(float, IQ)

```
public static IQ Multiply(float left, IQ right)
```

Parameters

**left** [float](#)↗

**right** [IQ](#)

Returns

[IQ](#)

## Negate(IQ)

```
public static IQ Negate(IQ value)
```

Parameters

value [IQ](#)

Returns

[IQ](#)

## Pow(IQ, IQ)

```
public static IQ Pow(IQ value, IQ power)
```

Parameters

value [IQ](#)

power [IQ](#)

Returns

[IQ](#)

## Pow(IQ, float)

```
public static IQ Pow(IQ value, float power)
```

Parameters

value [IQ](#)

power [float](#)

Returns

[IQ](#)

## Reciprocal(IQ)

```
public static IQ Reciprocal(IQ value)
```

Parameters

value [IQ](#)

Returns

[IQ](#)

## Sin(IQ)

```
public static IQ Sin(IQ value)
```

Parameters

value [IQ](#)

Returns

[IQ](#)

## Sinh(IQ)

```
public static IQ Sinh(IQ value)
```

Parameters

value [IQ](#)

Returns

[IQ](#)

## Sqrt(IQ)

```
public static IQ Sqrt(IQ value)
```

Parameters

value [IQ](#)

Returns

[IQ](#)

## Subtract(IQ, IQ)

```
public static IQ Subtract(IQ left, IQ right)
```

Parameters

left [IQ](#)

right [IQ](#)

Returns

[IQ](#)

## Subtract(IQ, float)

```
public static IQ Subtract(IQ left, float right)
```

Parameters

left [IQ](#)

`right float` ↗

Returns

[IQ](#)

## Subtract(float, IQ)

```
public static IQ Subtract(float left, IQ right)
```

Parameters

`left float` ↗

`right IQ`

Returns

[IQ](#)

## Tan(IQ)

```
public static IQ Tan(IQ value)
```

Parameters

`value IQ`

Returns

[IQ](#)

## Tanh(IQ)

```
public static IQ Tanh(IQ value)
```

Parameters

**value** [IQ](#)

Returns

[IQ](#)

## ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

[string](#)

The fully qualified type name.

## ToString(IFormatProvider?)

```
public string ToString(IFormatProvider? provider)
```

Parameters

**provider** [IFormatProvider](#)

Returns

[string](#)

## ToString(string?)

```
public string ToString(string? format)
```

Parameters

**format** [string](#)

Returns

[string](#)

## ToString(string?, IFormatProvider?)

Formats the value of the current instance using the specified format.

```
public string ToString(string? format, IFormatProvider? provider)
```

Parameters

**format** [string](#)

The format to use.

-or-

A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

**provider** [IFormatProvider](#)

Returns

[string](#)

The value of the current instance in the specified format.

## Operators

### operator +(IQ, IQ)

```
public static IQ operator +(IQ left, IQ right)
```

Parameters

**left** [IQ](#)

**right** [IQ](#)

Returns

[IQ](#)

**operator +(IQ, float)**

```
public static IQ operator +(IQ left, float right)
```

Parameters

**left** [IQ](#)

**right** [float](#)↗

Returns

[IQ](#)

**operator +(float, IQ)**

```
public static IQ operator +(float left, IQ right)
```

Parameters

**left** [float](#)↗

**right** [IQ](#)

Returns

[IQ](#)

## operator /(IQ, IQ)

```
public static IQ operator /(IQ left, IQ right)
```

Parameters

**left** [IQ](#)

**right** [IQ](#)

Returns

[IQ](#)

## operator /(IQ, float)

```
public static IQ operator /(IQ left, float right)
```

Parameters

**left** [IQ](#)

**right** [float](#) ↗

Returns

[IQ](#)

## operator /(float, IQ)

```
public static IQ operator /(float left, IQ right)
```

Parameters

**left** [float](#) ↗

**right** [IQ](#)

Returns

[IQ](#)

## operator ==(IQ, IQ)

```
public static bool operator ==(IQ left, IQ right)
```

Parameters

**left** [IQ](#)

**right** [IQ](#)

Returns

[bool](#)

## explicit operator IQ(decimal)

```
public static explicit operator IQ(decimal value)
```

Parameters

**value** [decimal](#)

Returns

[IQ](#)

## implicit operator Complex(IQ)

```
public static implicit operator Complex(IQ value)
```

Parameters

`value` [IQ](#)

Returns

[Complex](#)

## implicit operator IQ(byte)

```
public static implicit operator IQ(byte value)
```

Parameters

`value` [byte](#)

Returns

[IQ](#)

## implicit operator IQ(char)

```
public static implicit operator IQ(char value)
```

Parameters

`value` [char](#)

Returns

[IQ](#)

## implicit operator IQ(Half)

```
public static implicit operator IQ(Half value)
```

Parameters

**value** [Half](#)

Returns

[IQ](#)

## implicit operator IQ(short)

```
public static implicit operator IQ(short value)
```

Parameters

**value** [short](#)

Returns

[IQ](#)

## implicit operator IQ(int)

```
public static implicit operator IQ(int value)
```

Parameters

**value** [int](#)

Returns

[IQ](#)

## implicit operator IQ(long)

```
public static implicit operator IQ(long value)
```

Parameters

**value** [long](#)

Returns

[IQ](#)

## implicit operator IQ(nint)

```
public static implicit operator IQ(nint value)
```

Parameters

**value** [nint](#)

Returns

[IQ](#)

## implicit operator IQ(Complex)

```
public static implicit operator IQ(Complex value)
```

Parameters

**value** [Complex](#)

Returns

[IQ](#)

## implicit operator IQ(sbyte)

```
public static implicit operator IQ(sbyte value)
```

Parameters

**value** [sbyte](#)

Returns

[IQ](#)

## implicit operator IQ(float)

```
public static implicit operator IQ(float value)
```

Parameters

**value** [float](#)

Returns

[IQ](#)

## implicit operator IQ(ushort)

```
public static implicit operator IQ(ushort value)
```

Parameters

**value** [ushort](#)

Returns

[IQ](#)

## implicit operator IQ(uint)

```
public static implicit operator IQ(uint value)
```

Parameters

**value** [uint](#)

Returns

[IQ](#)

## implicit operator IQ(ulong)

```
public static implicit operator IQ(ulong value)
```

Parameters

**value** [ulong](#)

Returns

[IQ](#)

## implicit operator IQ(nuint)

```
public static implicit operator IQ(nuint value)
```

Parameters

**value** [nuint](#)

Returns

[IQ](#)

## operator !=(IQ, IQ)

```
public static bool operator !=(IQ left, IQ right)
```

Parameters

**left** [IQ](#)

**right** [IQ](#)

Returns

[bool](#) ↗

## operator \*(IQ, IQ)

```
public static IQ operator *(IQ left, IQ right)
```

Parameters

**left** [IQ](#)

**right** [IQ](#)

Returns

[IQ](#)

## operator \*(IQ, float)

```
public static IQ operator *(IQ left, float right)
```

Parameters

**left** [IQ](#)

**right** [float](#) ↗

Returns

[IQ](#)

## operator \*(float, IQ)

```
public static IQ operator *(float left, IQ right)
```

Parameters

left [float](#)

right [IQ](#)

Returns

[IQ](#)

## operator -(IQ, IQ)

```
public static IQ operator -(IQ left, IQ right)
```

Parameters

left [IQ](#)

right [IQ](#)

Returns

[IQ](#)

## operator -(IQ, float)

```
public static IQ operator -(IQ left, float right)
```

Parameters

left [IQ](#)

right [float](#)

Returns

## operator -(float, IQ)

```
public static IQ operator -(float left, IQ right)
```

Parameters

**left** [float](#)

**right** [IQ](#)

Returns

[IQ](#)

## operator -(IQ)

```
public static IQ operator -(IQ value)
```

Parameters

**value** [IQ](#)

Returns

[IQ](#)

# Struct InterleavedSample

Namespace: [HackRFDotnet.Api.Streams](#)

Assembly: HackRFDotnet.dll

```
public struct InterleavedSample
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

## Fields

|

```
public sbyte I
```

### Field Value

[sbyte](#)

Q

```
public sbyte Q
```

### Field Value

[sbyte](#)

## Methods

Clone()

```
public InterleavedSample Clone()
```

Returns

[InterleavedSample](#)

# Class SweepingIQStream

Namespace: [HackRFDotnet.Api.Streams](#)

Assembly: HackRFDotnet.dll

```
public class SweepingIQStream
```

## Inheritance

[object](#) ← SweepingIQStream

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

### SweepingIQStream()

```
public SweepingIQStream()
```

# Namespace HackRFDotnet.Api.Streams.Device Classes

[IQDeviceStream](#)

[IQFileStream](#)

# Class IQDeviceStream

Namespace: [HackRFDotnet.Api.Streams.Device](#)

Assembly: HackRFDotnet.dll

```
public class IQDeviceStream : IDisposable, IIQStream
```

## Inheritance

[object](#) ← IQDeviceStream

## Implements

[IDisposable](#), [IIQStream](#)

## Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),  
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

## Constructors

### IQDeviceStream(DigitalRadioDevice)

```
public IQDeviceStream(DigitalRadioDevice rfDevice)
```

## Parameters

rfDevice [DigitalRadioDevice](#)

## Properties

### BufferLength

```
public int BufferLength { get; }
```

## Property Value

[int](#)

# Frequency

```
public RadioBand Frequency { get; }
```

## Property Value

[RadioBand](#)

# SampleRate

```
public SampleRate SampleRate { get; }
```

## Property Value

[SampleRate](#)

# Methods

## Close()

```
public void Close()
```

## Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

## OpenRx(SampleRate?)

```
public void OpenRx(SampleRate? sampleRate = null)
```

Parameters

`sampleRate` [SampleRate](#)

## ReadBuffer(Span<IQ>)

```
public int ReadBuffer(Span<IQ> iqBuffer)
```

Parameters

`iqBuffer` [Span](#)<[IQ](#)>

Returns

[int](#)

## SetSampleRate(SampleRate)

```
public void SetSampleRate(SampleRate sampleRate)
```

Parameters

`sampleRate` [SampleRate](#)

## TxBuffer(Span<IQ>)

```
public int TxBuffer(Span<IQ> iqFrame)
```

Parameters

`iqFrame` [Span](#)<[IQ](#)>

Returns

[int](#)



# Class IQFileStream

Namespace: [HackRFDotnet.Api.Streams.Device](#)

Assembly: HackRFDotnet.dll

```
public class IQFileStream : IIQStream, IDisposable
```

## Inheritance

[object](#) ← IQFileStream

## Implements

[IIQStream](#), [IDisposable](#)

## Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),  
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

# Constructors

## IQFileStream(string)

```
public IQFileStream(string fileName)
```

## Parameters

fileName [string](#)

# Properties

## BufferLength

```
public int BufferLength { get; }
```

## Property Value

[int](#)

## Frequency

```
public RadioBand Frequency { get; set; }
```

### Property Value

[RadioBand](#)

## SampleRate

```
public SampleRate SampleRate { get; set; }
```

### Property Value

[SampleRate](#)

## Methods

### Close()

```
public void Close()
```

### Dispose()

```
public void Dispose()
```

### OpenRx(SampleRate?)

```
public void OpenRx(SampleRate? sampleRate = null)
```

Parameters

`sampleRate` [SampleRate](#)

## ReadBuffer(Span<IQ>)

```
public int ReadBuffer(Span<IQ> iqFrame)
```

Parameters

`iqFrame` [Span](#)<IQ>

Returns

[int](#)

## SetSampleRate(SampleRate)

```
public void SetSampleRate(SampleRate sampleRate)
```

Parameters

`sampleRate` [SampleRate](#)

## TxBuffer(Span<IQ>)

```
public int TxBuffer(Span<IQ> iqFrame)
```

Parameters

`iqFrame` [Span](#)<IQ>

Returns

[int](#)

## WriteBuffer(Span<byte>)

```
public int WriteBuffer(Span<byte> iqFrame)
```

### Parameters

`iqFrame` [Span<byte>](#)

### Returns

[int](#)

# Namespace HackRFDotnet.Api.Streams. Exceptions

## Classes

[NullCallbackException](#)

# Class NullCallbackException

Namespace: [HackRFDotnet.Api.Streams.Exceptions](#)

Assembly: HackRFDotnet.dll

```
public class NullCallbackException : Exception, ISerializable
```

## Inheritance

[object](#) ← [Exception](#) ← NullCallbackException

## Implements

[ISerializable](#)

## Inherited Members

[Exception.GetBaseException\(\)](#) , [Exception.GetType\(\)](#) , [Exception.ToString\(\)](#) , [Exception.Data](#) ,  
[Exception.HelpLink](#) , [Exception.HResult](#) , [Exception.InnerException](#) , [Exception.Message](#) ,  
[Exception.Source](#) , [Exception.StackTrace](#) , [Exception.TargetSite](#) , [Exception.SerializeObjectState](#) ,  
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

# Constructors

## NullCallbackException(string?)

```
public NullCallbackException(string? message)
```

## Parameters

message [string](#)

# Namespace HackRFDotnet.Api.Streams.Interfaces

## Interfaces

[IQStream](#)

# Interface IIQStream

Namespace: [HackRFDotnet.Api.Streams.Interfaces](#)

Assembly: HackRFDotnet.dll

```
public interface IIQStream
```

## Properties

### BufferLength

```
int BufferLength { get; }
```

Property Value

[int](#)

### Frequency

```
RadioBand Frequency { get; }
```

Property Value

[RadioBand](#)

### SampleRate

```
SampleRate SampleRate { get; }
```

Property Value

[SampleRate](#)

# Methods

## Close()

```
void Close()
```

## Dispose()

```
void Dispose()
```

## OpenRx(SampleRate?)

```
void OpenRx(SampleRate? sampleRate = null)
```

### Parameters

sampleRate [SampleRate](#)

## ReadBuffer(Span<IQ>)

```
int ReadBuffer(Span<IQ> iqBuffer)
```

### Parameters

iqBuffer [Span](#)<[IQ](#)>

### Returns

[int](#)

## SetSampleRate(SampleRate)

```
void SetSampleRate(SampleRate sampleRate)
```

Parameters

sampleRate [SampleRate](#)

## TxBuffer(Span<IQ>)

```
int TxBuffer(Span<IQ> iqFrame)
```

Parameters

iqFrame [Span](#)<IQ>

Returns

[int](#)

# Namespace HackRFDotnet.Api.Streams.Signal Processing

## Classes

[RadioBand](#)

[SampleRate](#)

[SignalProcessingBuilder](#)

[SignalProcessingPipeline](#)

# Class RadioBand

Namespace: [HackRFDotnet.Api.Streams.SignalProcessing](#)

Assembly: HackRFDotnet.dll

```
public class RadioBand
```

## Inheritance

[object](#) ← RadioBand

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Constructors

## RadioBand(int)

```
public RadioBand(int hz)
```

## Parameters

hz [int](#)

# Properties

## Hz

```
public int Hz { get; }
```

## Property Value

[int](#)

## Khz

```
public float Khz { get; }
```

### Property Value

[float](#) ↗

## Mhz

```
public float Mhz { get; }
```

### Property Value

[float](#) ↗

## NyquistSampleRate

```
public SampleRate NyquistSampleRate { get; }
```

### Property Value

[SampleRate](#)

## Methods

### FromHz(int)

```
public static RadioBand FromHz(int hz)
```

### Parameters

hz [int](#) ↗

Returns

[RadioBand](#)

## FromKHz(float)

```
public static RadioBand FromKHz(float khz)
```

Parameters

khz [float](#)

Returns

[RadioBand](#)

## FromMHz(float)

```
public static RadioBand FromMHz(float mhz)
```

Parameters

mhz [float](#)

Returns

[RadioBand](#)

# Operators

## operator +(RadioBand, RadioBand)

```
public static RadioBand operator +(RadioBand a, RadioBand b)
```

Parameters

a [RadioBand](#)

b [RadioBand](#)

Returns

[RadioBand](#)

## operator /(RadioBand, int)

```
public static RadioBand operator /(RadioBand a, int b)
```

Parameters

a [RadioBand](#)

b [int](#)

Returns

[RadioBand](#)

## operator ==(RadioBand, RadioBand)

```
public static bool operator ==(RadioBand a, RadioBand b)
```

Parameters

a [RadioBand](#)

b [RadioBand](#)

Returns

[bool](#)

## operator >(RadioBand, RadioBand)

```
public static bool operator >(RadioBand a, RadioBand b)
```

Parameters

a [RadioBand](#)

b [RadioBand](#)

Returns

[bool](#)

## operator >=(RadioBand, RadioBand)

```
public static bool operator >=(RadioBand a, RadioBand b)
```

Parameters

a [RadioBand](#)

b [RadioBand](#)

Returns

[bool](#)

## operator !=(RadioBand, RadioBand)

```
public static bool operator !=(RadioBand a, RadioBand b)
```

Parameters

a [RadioBand](#)

b [RadioBand](#)

Returns

[bool](#)

## operator <(RadioBand, RadioBand)

```
public static bool operator <(RadioBand a, RadioBand b)
```

Parameters

a [RadioBand](#)

b [RadioBand](#)

Returns

[bool](#)

## operator <=(RadioBand, RadioBand)

```
public static bool operator <=(RadioBand a, RadioBand b)
```

Parameters

a [RadioBand](#)

b [RadioBand](#)

Returns

[bool](#)

## operator %(RadioBand, RadioBand)

```
public static RadioBand operator %(RadioBand a, RadioBand b)
```

Parameters

a [RadioBand](#)

b [RadioBand](#)

Returns

[RadioBand](#)

## operator -(RadioBand, RadioBand)

```
public static RadioBand operator -(RadioBand a, RadioBand b)
```

Parameters

a [RadioBand](#)

b [RadioBand](#)

Returns

[RadioBand](#)

## operator -(RadioBand)

```
public static RadioBand operator -(RadioBand a)
```

Parameters

a [RadioBand](#)

Returns

[RadioBand](#)

# Class SampleRate

Namespace: [HackRFDotnet.Api.Streams.SignalProcessing](#)

Assembly: HackRFDotnet.dll

```
public class SampleRate
```

## Inheritance

[object](#) ← SampleRate

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Constructors

## SampleRate(int)

```
public SampleRate(int rate)
```

## Parameters

rate [int](#)

# Properties

## Ksps

```
public float Ksps { get; }
```

## Property Value

[float](#)

## MspS

```
public float MspS { get; }
```

Property Value

[float](#)

## NyquistFrequencyRange

```
public RadioBand NyquistFrequencyRange { get; }
```

Property Value

[RadioBand](#)

## Sps

```
public int Sps { get; }
```

Property Value

[int](#)

## Methods

### FromKSpS(float)

```
public static SampleRate FromKSpS(float ksps)
```

Parameters

ksps [float](#)

Returns

[SampleRate](#)

## FromMSps(float)

```
public static SampleRate FromMSps(float msp)
```

Parameters

msps [float](#)

Returns

[SampleRate](#)

## FromSps(int)

```
public static SampleRate FromSps(int sps)
```

Parameters

sps [int](#)

Returns

[SampleRate](#)

# Operators

## operator +(SampleRate, SampleRate)

```
public static SampleRate operator +(SampleRate a, SampleRate b)
```

Parameters

a [SampleRate](#)

b [SampleRate](#)

Returns

[SampleRate](#)

operator ==(SampleRate, SampleRate)

```
public static bool operator ==(SampleRate a, SampleRate b)
```

Parameters

a [SampleRate](#)

b [SampleRate](#)

Returns

[bool](#) ↗

operator >(SampleRate, SampleRate)

```
public static bool operator >(SampleRate a, SampleRate b)
```

Parameters

a [SampleRate](#)

b [SampleRate](#)

Returns

[bool](#) ↗

operator >=(SampleRate, SampleRate)

```
public static bool operator >=(SampleRate a, SampleRate b)
```

Parameters

a [SampleRate](#)

b [SampleRate](#)

Returns

[bool](#) ↗

## operator !=(SampleRate, SampleRate)

```
public static bool operator !=(SampleRate a, SampleRate b)
```

Parameters

a [SampleRate](#)

b [SampleRate](#)

Returns

[bool](#) ↗

## operator <(SampleRate, SampleRate)

```
public static bool operator <(SampleRate a, SampleRate b)
```

Parameters

a [SampleRate](#)

b [SampleRate](#)

Returns

[bool](#)

## operator <=(SampleRate, SampleRate)

```
public static bool operator <=(SampleRate a, SampleRate b)
```

Parameters

a [SampleRate](#)

b [SampleRate](#)

Returns

[bool](#)

## operator %(SampleRate, SampleRate)

```
public static SampleRate operator %(SampleRate a, SampleRate b)
```

Parameters

a [SampleRate](#)

b [SampleRate](#)

Returns

[SampleRate](#)

## operator -(SampleRate, SampleRate)

```
public static SampleRate operator -(SampleRate a, SampleRate b)
```

Parameters

a [SampleRate](#)

b [SampleRate](#)

Returns

[SampleRate](#)

## operator -(SampleRate)

```
public static SampleRate operator -(SampleRate a)
```

Parameters

a [SampleRate](#)

Returns

[SampleRate](#)

# Class SignalProcessingBuilder

Namespace: [HackRFDotnet.Api.Streams.SignalProcessing](#)

Assembly: HackRFDotnet.dll

```
public class SignalProcessingBuilder
```

## Inheritance

[object](#) ← SignalProcessingBuilder

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

### SignalProcessingBuilder()

```
public SignalProcessingBuilder()
```

## Methods

### AddSignalEffect(SignalEffect)

```
public SignalProcessingBuilder AddSignalEffect(SignalEffect signalEffect)
```

#### Parameters

signalEffect [SignalEffect](#)

#### Returns

[SignalProcessingBuilder](#)

## BuildPipeline()

```
public SignalProcessingPipeline BuildPipeline()
```

Returns

[SignalProcessingPipeline](#)

# Class SignalProcessingPipeline

Namespace: [HackRFDotnet.Api.Streams.SignalProcessing](#)

Assembly: HackRFDotnet.dll

```
public class SignalProcessingPipeline
```

## Inheritance

[object](#) ← SignalProcessingPipeline

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

### SignalProcessingPipeline(SignalEffect[])

```
public SignalProcessingPipeline(SignalEffect[] signalFxPipe)
```

## Parameters

signalFxPipe [SignalEffect](#)[]

## Methods

### ApplyPipeline(Span<IQ>)

```
public int ApplyPipeline(Span<IQ> signalTheta)
```

## Parameters

signalTheta [Span](#)<[IQ](#)>

## Returns

int ↗

# Namespace HackRFDotnet.Api.Streams.Signal Processing.Effects

## Classes

[BasicSignalScanningEffect](#)

[DownSampleEffect](#)

[DownSampleEffect](#) removes extraneous information from your signal using your desired bandwidth.

Example: an FM radio's band is around 200 kHz; the minimum sample rate required to represent this is 400 kS/s (400,000 samples per second). It is recommended that you reduce the sample rate of your audio signal this way before further signal processing to save CPU.

[FftEffect](#)

[FrequencyCenteringEffect](#)

[LowPassFilterEffect](#)

[SignalEffect](#)

[SquelchEffect](#)

# Class BasicSignalScanningEffect

Namespace: [HackRFDotnet.Api.Streams.SignalProcessing.Effects](#)

Assembly: HackRFDotnet.dll

```
public class BasicSignalScanningEffect : SignalEffect, ISignalEffect
```

## Inheritance

[object](#) ← [SignalEffect](#) ← BasicSignalScanningEffect

## Implements

[ISignalEffect](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

BasicSignalScanningEffect(DigitalRadioDevice, RadioBand,  
RadioBand[])

```
public BasicSignalScanningEffect(DigitalRadioDevice digitalRadioDevice, RadioBand bandwidth,  
RadioBand[] scanChannels)
```

## Parameters

[digitalRadioDevice](#) [DigitalRadioDevice](#)

[bandwidth](#) [RadioBand](#)

[scanChannels](#) [RadioBand\[\]](#)

## Methods

AffectSignal(Span<IQ>, int)

```
public override int AffectSignal(Span<IQ> signalTheta, int length)
```

Parameters

signalTheta [Span<IQ>](#)

length [int](#)

Returns

[int](#)

# Class DownSampleEffect

Namespace: [HackRFDotnet.Api.Streams.SignalProcessing.Effects](#)

Assembly: HackRFDotnet.dll

[DownSampleEffect](#) removes extraneous information from your signal using your desired bandwidth. Example: an FM radio's band is around 200 kHz; the minimum sample rate required to represent this is 400 kS/s (400,000 samples per second). It is recommended that you reduce the sample rate of your audio signal this way before further signal processing to save CPU.

```
public class DownSampleEffect : SignalEffect, ISignalEffect, IDisposable
```

## Inheritance

[object](#) ← [SignalEffect](#) ← DownSampleEffect

## Implements

[ISignalEffect](#), [IDisposable](#)

## Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),  
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

## Constructors

DownSampleEffect(SampleRate, SampleRate, out SampleRate, out int)

```
public DownSampleEffect(SampleRate sampleRate, SampleRate reducedSampleRate, out SampleRate newSampleRate, out int producedChunkSize)
```

## Parameters

sampleRate [SampleRate](#)

reducedSampleRate [SampleRate](#)

newSampleRate [SampleRate](#)

producedChunkSize [int](#)

DownSampleEffect(SampleRate, SampleRate, int, out  
SampleRate, out int)

```
public DownSampleEffect(SampleRate sampleRate, SampleRate reducedSampleRate, int  
processingSize, out SampleRate newSampleRate, out int producedChunkSize)
```

Parameters

sampleRate [SampleRate](#)

reducedSampleRate [SampleRate](#)

processingSize [int](#)

newSampleRate [SampleRate](#)

producedChunkSize [int](#)

## Methods

AffectSignal(Span<IQ>, int)

```
public override int AffectSignal(Span<IQ> signalTheta, int length)
```

Parameters

signalTheta [Span](#)<IQ>

length [int](#)

Returns

[int](#)

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

# Class FftEffect

Namespace: [HackRFDotnet.Api.Streams.SignalProcessing.Effects](#)

Assembly: HackRFDotnet.dll

```
public class FftEffect : SignalEffect, ISignalEffect, IDisposable
```

## Inheritance

[object](#) ← [SignalEffect](#) ← FftEffect

## Implements

[ISignalEffect](#), [IDisposable](#)

## Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),  
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

# Constructors

## FftEffect(bool, int)

```
public FftEffect(bool forward, int chunkSize)
```

## Parameters

forward [bool](#)

chunkSize [int](#)

# Methods

## AffectSignal(Span<IQ>, int)

```
public override int AffectSignal(Span<IQ> signalTheta, int length)
```

## Parameters

`signalTheta` [Span](#)<IQ>

`length` [int](#)

Returns

[int](#)

## Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

`public void Dispose()`

# Class FrequencyCenteringEffect

Namespace: [HackRFDotnet.Api.Streams.SignalProcessing.Effects](#)

Assembly: HackRFDotnet.dll

```
public class FrequencyCenteringEffect : SignalEffect, ISignalEffect
```

## Inheritance

[object](#) ← [SignalEffect](#) ← FrequencyCenteringEffect

## Implements

[ISignalEffect](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

FrequencyCenteringEffect(RadioBand, SampleRate)

```
public FrequencyCenteringEffect(RadioBand frequencyOffset, SampleRate sampleRate)
```

## Parameters

frequencyOffset [RadioBand](#)

sampleRate [SampleRate](#)

## Methods

AffectSignal(Span<IQ>, int)

```
public override int AffectSignal(Span<IQ> signalTheta, int length)
```

## Parameters

`signalTheta` [Span](#) <IQ>

`length` [int](#)

Returns

[int](#)

# Class LowPassFilterEffect

Namespace: [HackRFDotnet.Api.Streams.SignalProcessing.Effects](#)

Assembly: HackRFDotnet.dll

```
public class LowPassFilterEffect : SignalEffect, ISignalEffect
```

## Inheritance

[object](#) ← [SignalEffect](#) ← LowPassFilterEffect

## Implements

[ISignalEffect](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

### LowPassFilterEffect(SampleRate, RadioBand)

Apply a low pass filter on the signal. Expects Frequency Domain

```
public LowPassFilterEffect(SampleRate sampleRate, RadioBand bandwith)
```

## Parameters

sampleRate [SampleRate](#)

bandwith [RadioBand](#)

## Methods

### AffectSignal(Span<IQ>, int)

```
public override int AffectSignal(Span<IQ> signalTheta, int length)
```

Parameters

signalTheta [Span](#)<IQ>

length [int](#)

Returns

[int](#)

# Class SignalEffect

Namespace: [HackRFDotnet.Api.Streams.SignalProcessing.Effects](#)

Assembly: HackRFDotnet.dll

```
public abstract class SignalEffect : ISignalEffect
```

Inheritance

[object](#) ← SignalEffect

Implements

[ISignalEffect](#)

Derived

[BasicSignalScanningEffect](#), [DownSampleEffect](#), [FftEffect](#), [FrequencyCenteringEffect](#), [LowPassFilterEffect](#),  
[SquelchEffect](#)

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),  
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

## Methods

### AffectSignal(Span<IQ>, int)

```
public abstract int AffectSignal(Span<IQ> signalTheta, int length)
```

Parameters

signalTheta [Span](#)<IQ>

length [int](#)

Returns

[int](#)

# Class SquelchEffect

Namespace: [HackRFDotnet.Api.Streams.SignalProcessing.Effects](#)

Assembly: HackRFDotnet.dll

```
public class SquelchEffect : SignalEffect, ISignalEffect
```

## Inheritance

[object](#) ← [SignalEffect](#) ← SquelchEffect

## Implements

[ISignalEffect](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

### SquelchEffect(SampleRate)

```
public SquelchEffect(SampleRate sampleRate)
```

## Parameters

sampleRate [SampleRate](#)

## Methods

### AffectSignal(Span<IQ>, int)

```
public override int AffectSignal(Span<IQ> signalTheta, int length)
```

## Parameters

signalTheta [Span](#)<IQ>

`length` [int ↗](#)

Returns

[int ↗](#)

# Namespace HackRFDotnet.Api.Streams.Signal Streams

## Classes

[SignalStream](#)

# Class SignalStream

Namespace: [HackRFDotnet.Api.Streams.SignalStreams](#)

Assembly: HackRFDotnet.dll

```
public class SignalStream : IDisposable
```

## Inheritance

[object](#) ← SignalStream

## Implements

[IDisposable](#)

## Derived

[WaveSignalStream](#), [QpskSignalStream](#)

## Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),  
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

# Constructors

## SignalStream(IIQStream, SignalProcessingPipeline?, bool)

```
public SignalStream(IIQStream iqStream, SignalProcessingPipeline? processingPipeline = null,  
bool keepOpen = true)
```

## Parameters

`iQStream` [IIQStream](#)

`processingPipeline` [SignalProcessingPipeline](#)

`keepOpen` [bool](#)

# Fields

## \_iQStream

```
protected readonly IIQStream _iQStream
```

### Field Value

[IIQStream](#)

## \_keepOpen

```
protected readonly bool _keepOpen
```

### Field Value

[bool](#)

## \_processingPipeline

```
protected SignalProcessingPipeline? _processingPipeline
```

### Field Value

[SignalProcessingPipeline](#)

## Properties

### BandWidth

```
public RadioBand BandWidth { get; protected set; }
```

### Property Value

[RadioBand](#)

## Center

```
public RadioBand Center { get; protected set; }
```

### Property Value

[RadioBand](#)

## SampleRate

```
public SampleRate SampleRate { get; }
```

### Property Value

[SampleRate](#)

## Methods

### Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

### ReadSpan(Span<IQ>)

```
public void ReadSpan(Span<IQ> iqPairs)
```

### Parameters

`iqPairs` [Span](#)<[IQ](#)>

### SetBand(RadioBand, RadioBand)

Set the band and bandwidth the filtering engine will use.

```
public void SetBand(RadioBand center, RadioBand bandwidth)
```

## Parameters

center [RadioBand](#)

bandwidth [RadioBand](#)

# Namespace HackRFDotnet.Api.Streams.Signal Streams.Analogue

## Classes

[AmSignalStream](#)

[FmSignalStream](#)

[WaveSignalStream](#)

# Class AmSignalStream

Namespace: [HackRFDotnet.Api.Streams.SignalStreams.Analogue](#)

Assembly: HackRFDotnet.dll

```
public class AmSignalStream : WaveSignalStream, ISampleProvider, IDisposable
```

## Inheritance

[object](#) ← [SignalStream](#) ← [WaveSignalStream](#) ← AmSignalStream

## Implements

ISampleProvider, [IDisposable](#)

## Inherited Members

[WaveSignalStream.WaveFormat](#), [WaveSignalStream.NormalizeRms\(Span<float>, float\)](#),  
[SignalStream.Center](#), [SignalStream.BandWidth](#), [SignalStream.SampleRate](#),  
[SignalStream.processingPipeline](#), [SignalStream.iQStream](#), [SignalStream.keepOpen](#),  
[SignalStream.ReadSpan\(Span<IQ>\)](#), [SignalStream.SetBand\(RadioBand, RadioBand\)](#),  
[SignalStream.Dispose\(\)](#), [object.Equals\(object\)](#), [object.Equals\(object, object\)](#),  
[object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#),  
[object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

## Constructors

AmSignalStream(IIQStream, SampleRate,  
SignalProcessingPipeline?, bool)

```
public AmSignalStream(IIQStream deviceStream, SampleRate sampleRate,  
SignalProcessingPipeline? processingPipeline = null, bool keepOpen = true)
```

## Parameters

deviceStream [IIQStream](#)

sampleRate [SampleRate](#)

processingPipeline [SignalProcessingPipeline](#)

keepOpen [bool](#)

## Methods

### Read(float[], int, int)

Fill the specified buffer with 32 bit floating point samples

```
public override int Read(float[] buffer, int offset, int count)
```

#### Parameters

**buffer** [float](#)[]

The buffer to fill with samples.

**offset** [int](#)

Offset into buffer

**count** [int](#)

The number of samples to read

#### Returns

[int](#)

the number of samples written to the buffer.

# Class FmSignalStream

Namespace: [HackRFDotnet.Api.Streams.SignalStreams.Analogue](#)

Assembly: HackRFDotnet.dll

```
public class FmSignalStream : WaveSignalStream, ISampleProvider, IDisposable
```

## Inheritance

[object](#) ← [SignalStream](#) ← [WaveSignalStream](#) ← FmSignalStream

## Implements

ISampleProvider, [IDisposable](#)

## Inherited Members

[WaveSignalStream.WaveFormat](#), [WaveSignalStream.NormalizeRms\(Span<float>, float\)](#),  
[SignalStream.Center](#), [SignalStream.BandWidth](#), [SignalStream.SampleRate](#),  
[SignalStream.processingPipeline](#), [SignalStream.iQStream](#), [SignalStream.keepOpen](#),  
[SignalStream.ReadSpan\(Span<IQ>\)](#), [SignalStream.SetBand\(RadioBand, RadioBand\)](#),  
[SignalStream.Dispose\(\)](#), [object.Equals\(object\)](#), [object.Equals\(object, object\)](#),  
[object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#),  
[object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

## Constructors

FmSignalStream(IIQStream, SampleRate, bool,  
SignalProcessingPipeline?, bool)

```
public FmSignalStream(IIQStream deviceStream, SampleRate sampleRate, bool stereo = true,  
SignalProcessingPipeline? processingPipeline = null, bool keepOpen = true)
```

## Parameters

deviceStream [IIQStream](#)

sampleRate [SampleRate](#)

stereo [bool](#)

processingPipeline [SignalProcessingPipeline](#)

keepOpen [bool](#)

## Methods

### Read(float[], int, int)

Fill the specified buffer with 32 bit floating point samples

```
public override int Read(float[] buffer, int offset, int count)
```

#### Parameters

**buffer** [float](#)[]

The buffer to fill with samples.

**offset** [int](#)

Offset into buffer

**count** [int](#)

The number of samples to read

#### Returns

[int](#)

the number of samples written to the buffer.

# Class WaveSignalStream

Namespace: [HackRFDotnet.Api.Streams.SignalStreams.Analogue](#)

Assembly: HackRFDotnet.dll

```
public class WaveSignalStream : SignalStream, ISampleProvider, IDisposable
```

## Inheritance

[object](#) ← [SignalStream](#) ← WaveSignalStream

## Implements

ISampleProvider, [IDisposable](#)

## Derived

[AmSignalStream](#), [FmSignalStream](#)

## Inherited Members

[SignalStream.Center](#), [SignalStream.BandWidth](#), [SignalStream.SampleRate](#),  
[SignalStream.ProcessingPipeline](#), [SignalStream.IQStream](#), [SignalStream.KeepOpen](#),  
[SignalStream.ReadSpan\(Span<IQ>\)](#), [SignalStream.SetBand\(RadioBand, RadioBand\)](#),  
[SignalStream.Dispose\(\)](#), [object.Equals\(object\)](#), [object.Equals\(object, object\)](#),  
[object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#),  
[object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

## Constructors

WaveSignalStream(IQStream, SampleRate, bool,  
SignalProcessingPipeline?, bool)

```
public WaveSignalStream(IQStream deviceStream, SampleRate sampleRate, bool stereo = true,  
SignalProcessingPipeline? processingPipeline = null, bool keepOpen = true)
```

## Parameters

deviceStream [IQStream](#)

sampleRate [SampleRate](#)

stero [bool](#)

processingPipeline [SignalProcessingPipeline](#)

keepOpen [bool](#)

## Properties

### WaveFormat

Gets the WaveFormat of this Sample Provider.

```
public WaveFormat? WaveFormat { get; protected set; }
```

#### Property Value

WaveFormat

The wave format.

## Methods

### NormalizeRms(Span<float>, float)

```
protected void NormalizeRms(Span<float> buffer, float targetRms = 0.04)
```

#### Parameters

buffer [Span](#)<[float](#)>

targetRms [float](#)

### Read(float[], int, int)

Fill the specified buffer with 32 bit floating point samples

```
public virtual int Read(float[] buffer, int offset, int count)
```

## Parameters

**buffer** [float\[\]](#)[]

The buffer to fill with samples.

**offset** [int](#)[]

Offset into buffer

**count** [int](#)[]

The number of samples to read

## Returns

[int](#)[]

the number of samples written to the buffer.

# Namespace HackRFDotnet.Api.Streams.Signal Streams.Digital

## Classes

[HdRadioSignalStream](#)

[QpskSignalStream](#)

# Class HdRadioSignalStream

Namespace: [HackRFDotnet.Api.Streams.SignalStreams.Digital](#)

Assembly: HackRFDotnet.dll

```
public class HdRadioSignalStream : QpskSignalStream, IDisposable, ISampleProvider
```

## Inheritance

[object](#) ← [SignalStream](#) ← [QpskSignalStream](#) ← HdRadioSignalStream

## Implements

[IDisposable](#), ISampleProvider

## Inherited Members

[QpskSignalStream.Read\(Span<byte>, int\)](#), [SignalStream.Center](#), [SignalStream.BandWidth](#),  
[SignalStream.SampleRate](#), [SignalStream.processingPipeline](#), [SignalStream.iQStream](#),  
[SignalStream.keepOpen](#), [SignalStream.ReadSpan\(Span<IQ>\)](#),  
[SignalStream.SetBand\(RadioBand, RadioBand\)](#), [SignalStream.Dispose\(\)](#), [object.Equals\(object\)](#),  
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),  
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

## Constructors

HdRadioSignalStream(IIQStream, SampleRate, bool,  
SignalProcessingPipeline?, bool)

```
public HdRadioSignalStream(IIQStream iQStream, SampleRate sampleRate, bool stereo = true,  
SignalProcessingPipeline? processingPipeline = null, bool keepOpen = true)
```

## Parameters

iQStream [IIQStream](#)

sampleRate [SampleRate](#)

stereo [bool](#)

processingPipeline [SignalProcessingPipeline](#)

`keepOpen` [bool](#)

## Properties

### WaveFormat

Gets the WaveFormat of this Sample Provider.

```
public WaveFormat? WaveFormat { get; protected set; }
```

### Property Value

#### WaveFormat

The wave format.

## Methods

### Read(float[], int, int)

Fill the specified buffer with 32 bit floating point samples

```
public int Read(float[] buffer, int offset, int count)
```

### Parameters

`buffer` [float](#)[]

The buffer to fill with samples.

`offset` [int](#)

Offset into buffer

`count` [int](#)

The number of samples to read

### Returns

int ↗

the number of samples written to the buffer.

# Class QpskSignalStream

Namespace: [HackRFDotnet.Api.Streams.SignalStreams.Digital](#)

Assembly: HackRFDotnet.dll

```
public class QpskSignalStream : SignalStream, IDisposable
```

## Inheritance

[object](#) ← [SignalStream](#) ← QpskSignalStream

## Implements

[IDisposable](#)

## Derived

[HdRadioSignalStream](#)

## Inherited Members

[SignalStream.Center](#) , [SignalStream.BandWidth](#) , [SignalStream.SampleRate](#) ,  
[SignalStream.ProcessingPipeline](#) , [SignalStream.IQStream](#) , [SignalStream.KeepOpen](#) ,  
[SignalStream.ReadSpan\(Span<IQ>\)](#) , [SignalStream.SetBand\(RadioBand, RadioBand\)](#) ,  
[SignalStream.Dispose\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.GetHashCode\(\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

**QpskSignalStream(IQStream, SignalProcessingPipeline?, bool)**

```
public QpskSignalStream(IQStream iqStream, SignalProcessingPipeline? processingPipeline = null, bool keepOpen = true)
```

## Parameters

**iQStream** [IQStream](#)

**processingPipeline** [SignalProcessingPipeline](#)

**keepOpen** [bool](#)

## Methods

### Read(Span<byte>, int)

```
public int Read(Span<byte> buffer, int count)
```

#### Parameters

buffer [Span<byte>](#)

count [int](#)

#### Returns

[int](#)

# Namespace HackRFDotnet.Api.Utilities

## Classes

[BinaryUtilities](#)

[SignalUtilities](#)

# Class BinaryUtilities

Namespace: [HackRFDotnet.Api.Utilities](#)

Assembly: HackRFDotnet.dll

```
public static class BinaryUtilities
```

## Inheritance

[object](#) ← BinaryUtilities

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Methods

## NextPowerOfTwo(int)

```
public static int NextPowerOfTwo(int n)
```

### Parameters

n [int](#)

### Returns

[int](#)

# Class SignalUtilities

Namespace: [HackRFDotnet.Api.Utilities](#)

Assembly: HackRFDotnet.dll

```
public class SignalUtilities
```

## Inheritance

[object](#) ← SignalUtilities

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

### ApplyFrequencyOffset(Span<IQ>, RadioBand, SampleRate)

```
public static void ApplyFrequencyOffset(Span<IQ> iqFrame, RadioBand freqOffset,  
SampleRate sampleRate)
```

#### Parameters

**iqFrame** [Span](#)<[IQ](#)>

**freqOffset** [RadioBand](#)

**sampleRate** [SampleRate](#)

### CalculateRmsDb(ReadOnlySpan<IQ>)

```
public static float CalculateRmsDb(ReadOnlySpan<IQ> iqFrame)
```

#### Parameters

**iqFrame** [ReadOnlySpan](#)<[IQ](#)>

Returns

[float](#)

## CalculateSignalDb(ReadOnlySpan<IQ>)

```
public static float CalculateSignalDb(ReadOnlySpan<IQ> iqFrame)
```

Parameters

[iqFrame](#) [ReadOnlySpan](#)<[IQ](#)>

Returns

[float](#)

## FrequencyResolution(int, SampleRate, bool)

```
public static int FrequencyResolution(int length, SampleRate sampleRate, bool positiveOnly  
= true)
```

Parameters

[length](#) [int](#)

[sampleRate](#) [SampleRate](#)

[positiveOnly](#) [bool](#)

Returns

[int](#)

## IQCorrection(Span<IQ>)

```
public static void IQCorrection(Span<IQ> iqFrame)
```

## Parameters

iqFrame [Span](#)<IQ>

# Namespace HackRFDotnet.ManagedApi.Streams.SignalProcessing.Interfaces

## Interfaces

[ISignalEffect](#)

# Interface ISignalEffect

Namespace: [HackRFDotnet.ManagedApi.Streams.SignalProcessing.Interfaces](#)

Assembly: HackRFDotnet.dll

```
public interface ISignalEffect
```

## Methods

### AffectSignal(Span<IQ>, int)

```
int AffectSignal(Span<IQ> signalTheta, int lendth)
```

Parameters

signalTheta [Span<IQ>](#)

lendth [int](#)

Returns

[int](#)

# Namespace HackRFDotnet.NativeApi

## Classes

[NativeConstants](#)

# Class NativeConstants

Namespace: [HackRFDotnet.NativeApi](#)

Assembly: HackRFDotnet.dll

```
public class NativeConstants
```

## Inheritance

[object](#) ← NativeConstants

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Fields

### BOARD\_ID\_HACKRF\_ONE

These deprecated board ID names are provided for API compatibility.

```
public const HackrfBoardId BOARD_ID_HACKRF_ONE = BOARD_ID_HACKRF1_0G
```

#### Field Value

[HackrfBoardId](#)

### BOARD\_ID\_INVALID

These deprecated board ID names are provided for API compatibility.

```
public const HackrfBoardId BOARD_ID_INVALID = BOARD_ID_UNDETECTED
```

#### Field Value

[HackrfBoardId](#)

## BYTES\_PER\_BLOCK

Number of bytes per tuning for sweeping

```
public const uint BYTES_PER_BLOCK = 16384
```

Field Value

[uint](#)

## HACKRF\_BOARD\_REV\_GSG

Made by GSG bit in @ref hackrf\_board\_rev enum and in platform ID

```
public const uint HACKRF_BOARD_REV_GSG = 128
```

Field Value

[uint](#)

## HACKRF\_OPERACAKE\_ADDRESS\_INVALID

Invalid Opera Cake add-on board address, placeholder in @ref hackrf\_get\_operacake\_boards

```
public const uint HACKRF_OPERACAKE_ADDRESS_INVALID = 255
```

Field Value

[uint](#)

## HACKRF\_OPERACAKE\_MAX\_BOARDS

Maximum number of connected Opera Cake add-on boards

```
public const uint HACKRF_OPERACAKE_MAX_BOARDS = 8
```

Field Value

[uint](#) ↗

## HACKRF\_OPERACAKE\_MAX\_DWELL\_TIMES

Maximum number of specifiable dwell times for Opera Cake add-on boards

```
public const uint HACKRF_OPERACAKE_MAX_DWELL_TIMES = 16
```

Field Value

[uint](#) ↗

## HACKRF\_OPERACAKE\_MAX\_FREQ\_RANGES

Maximum number of specifiable frequency ranges for Opera Cake add-on boards

```
public const uint HACKRF_OPERACAKE_MAX_FREQ_RANGES = 8
```

Field Value

[uint](#) ↗

## HACKRF\_PLATFORM\_HACKRF1\_OG

HACKRF ONE (pre r9) platform bit in result of @ref hackrf\_supported\_platform\_read

```
public const uint HACKRF_PLATFORM_HACKRF1_OG = 2
```

Field Value

[uint](#) ↗

## HACKRF\_PLATFORM\_HACKRF1\_R9

HACKRF ONE (r9 or later) platform bit in result of @ref hackrf\_supported\_platform\_read

```
public const uint HACKRF_PLATFORM_HACKRF1_R9 = 8
```

Field Value

[uint](#)

## HACKRF\_PLATFORM\_JAWBREAKER

JAWBREAKER platform bit in result of @ref hackrf\_supported\_platform\_read

```
public const uint HACKRF_PLATFORM_JAWBREAKER = 1
```

Field Value

[uint](#)

## HACKRF\_PLATFORM\_RAD10

RAD10 platform bit in result of @ref hackrf\_supported\_platform\_read

```
public const uint HACKRF_PLATFORM_RAD10 = 4
```

Field Value

[uint](#)

## MAX\_SWEEP\_RANGES

Maximum number of sweep ranges to be specified for @ref hackrf\_init\_sweep

```
public const uint MAX_SWEEP_RANGES = 10
```

Field Value

[uint](#) ↗

## SAMPLES\_PER\_BLOCK

Number of samples per tuning when sweeping

```
public const uint SAMPLES_PER_BLOCK = 8192
```

Field Value

[uint](#) ↗

# Namespace HackRFDotnet.NativeApi.Enums

## Enums

[RfPathFilter](#)

[SweepStyle](#)

# Enum RfPathFilter

Namespace: [HackRFDotnet.NativeApi.Enums](#)

Assembly: HackRFDotnet.dll

```
public enum RfPathFilter
```

## Fields

RF\_PATH\_FILTER\_BYPASS = 0

No filter is selected, **the mixer is bypassed**,  $f_{center} = f_{IF}$

RF\_PATH\_FILTER\_HIGH\_PASS = 2

HPF is selected,  $f_{center} = f_{IF} + f_{LO}$

RF\_PATH\_FILTER\_LOW\_PASS = 1

LPF is selected,  $f_{center} = f_{IF} - f_{LO}$

# Enum SweepStyle

Namespace: [HackRFDotnet.NativeApi.Enums](#)

Assembly: HackRFDotnet.dll

```
public enum SweepStyle
```

## Fields

INTERLEAVED = 1

LINEAR = 0

# Namespace HackRFDotnet.NativeApi.Enums. Peripherals

## Enums

[LedState](#)

[OperacakePorts](#)

[OperacakeSwitchingMode](#)

# Enum LedState

Namespace: [HackRFDotnet.NativeApi.Enums.Peripherals](#)

Assembly: HackRFDotnet.dll

```
public enum LedState : byte
```

## Fields

RxLight = 2

TxLight = 4

UsbLight = 0

# Enum OperacakePorts

Namespace: [HackRFDotnet.NativeApi.Enums.Peripherals](#)

Assembly: HackRFDotnet.dll

```
public enum OperacakePorts
```

## Fields

OPERACAKE\_PA1 = 0

OPERACAKE\_PA2 = 1

OPERACAKE\_PA3 = 2

OPERACAKE\_PA4 = 3

OPERACAKE\_PB1 = 4

OPERACAKE\_PB2 = 5

OPERACAKE\_PB3 = 6

OPERACAKE\_PB4 = 7

# Enum OperacakeSwitchingMode

Namespace: [HackRFDotnet.NativeApi.Enums.Peripherals](#)

Assembly: HackRFDotnet.dll

```
public enum OperacakeSwitchingMode
```

## Fields

OPERACAKE\_MODE\_FREQUENCY = 1

OPERACAKE\_MODE\_MANUAL = 0

OPERACAKE\_MODE\_TIME = 2

# Namespace HackRFDotnet.NativeApi.Enums.System

## Enums

[HackrfBoardId](#)

[HackrfBoardRev](#)

[HackrfError](#)

[HackrfUsbBoardId](#)

# Enum HackrfBoardId

Namespace: [HackRFDotnet.NativeApi.Enums.System](#)

Assembly: HackRFDotnet.dll

```
public enum HackrfBoardId
```

## Fields

BOARD\_ID\_HACKRF1\_OG = 2

BOARD\_ID\_HACKRF1\_R9 = 4

BOARD\_ID\_JAWBREAKER = 1

BOARD\_ID\_JELLYBEAN = 0

BOARD\_ID\_RAD10 = 3

BOARD\_ID\_UNDETECTED = 255

BOARD\_ID\_UNRECOGNIZED = 254

# Enum HackrfBoardRev

Namespace: [HackRFDotnet.NativeApi.Enums.System](#)

Assembly: HackRFDotnet.dll

```
public enum HackrfBoardRev
```

## Fields

BOARD\_REV\_GSG HACKRF1\_R10 = 133

BOARD\_REV\_GSG HACKRF1\_R6 = 129

BOARD\_REV\_GSG HACKRF1\_R7 = 130

BOARD\_REV\_GSG HACKRF1\_R8 = 131

BOARD\_REV\_GSG HACKRF1\_R9 = 132

BOARD\_REV\_HACKRF1\_OLD = 0

BOARD\_REV\_HACKRF1\_R10 = 5

BOARD\_REV\_HACKRF1\_R6 = 1

BOARD\_REV\_HACKRF1\_R7 = 2

BOARD\_REV\_HACKRF1\_R8 = 3

BOARD\_REV\_HACKRF1\_R9 = 4

BOARD\_REV\_UNDETECTED = 255

BOARD\_REV\_UNRECOGNIZED = 254

# Enum HackrfError

Namespace: [HackRFDotnet.NativeApi.Enums.System](#)

Assembly: HackRFDotnet.dll

```
public enum HackrfError
```

## Fields

**HACKRF\_ERROR\_BUSY = -6**

Resource is busy, possibly the device is already opened.

**HACKRF\_ERROR\_INVALID\_PARAM = -2**

The function was called with invalid parameters.

**HACKRF\_ERROR\_LIBUSB = -1000**

LibUSB error, use @ref hackrf\_error\_name to get a human-readable error string (using `libusb_strerror()`)

**HACKRF\_ERROR\_NOT\_FOUND = -5**

USB device not found, returned at opening.

**HACKRF\_ERROR\_NOT\_LAST\_DEVICE = -2000**

Can not exit library as one or more HackRFs still in use

**HACKRF\_ERROR\_NO\_MEM = -11**

Memory allocation (on host side) failed

**HACKRF\_ERROR\_OTHER = -9999**

Unspecified error

**HACKRF\_ERROR\_STREAMING\_EXIT\_CALLED = -1004**

Streaming thread exited (normally)

**HACKRF\_ERROR\_STREAMING\_STOPPED = -1003**

Streaming thread stopped due to an error

**HACKRF\_ERROR\_STREAMING\_THREAD\_ERR = -1002**

Streaming thread could not start due to an error

**HACKRF\_ERROR\_THREAD = -1001**

Error setting up transfer thread (pthread-related error)

**HACKRF\_ERROR\_USB\_API\_VERSION = -1005**

The installed firmware does not support this function

**HACKRF\_SUCCESS = 0**

no error happened

**HACKRF\_TRUE = 1**

TRUE value, returned by some functions that return boolean value. Only a few functions can return this variant, and this fact should be explicitly noted at those functions.

# Enum HackrfUsbBoardId

Namespace: [HackRFDotnet.NativeApi.Enums.System](#)

Assembly: HackRFDotnet.dll

```
public enum HackrfUsbBoardId
```

## Fields

USB\_BOARD\_ID\_HACKRF\_ONE = 24713

USB\_BOARD\_ID\_INVALID = 65535

USB\_BOARD\_ID\_JAWBREAKER = 24651

USB\_BOARD\_ID\_RAD10 = 52245

# Namespace HackRFDotnet.NativeApi.Lib

## Classes

[HackRfNativeLib](#)

[HackRfNativeLib.Debug](#)

[HackRfNativeLib.DeviceStreaming](#)

[HackRfNativeLib.Devices](#)

[HackRfNativeLib.Firmware](#)

[HackRfNativeLib.Operacake](#)

# Class HackRfNativeLib

Namespace: [HackRFDotnet.NativeApi.Lib](#)

Assembly: HackRFDotnet.dll

```
public static class HackRfNativeLib
```

## Inheritance

[object](#) ← HackRfNativeLib

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Methods

## Exit()

Exit libhackrf. Should be called before application exit.

```
public static extern int Exit()
```

## Returns

[int](#)

## Init()

Initialize libhackrf. Should be called before any other function.

```
public static extern int Init()
```

## Returns

[int](#)

## LibraryRelease()

Get library release string.

```
public static extern sbyte* LibraryRelease()
```

Returns

sbyte↗\*

## LibraryVersion()

Get library version string.

```
public static extern sbyte* LibraryVersion()
```

Returns

sbyte↗\*

# Class HackRfNativeLib.Debug

Namespace: [HackRFDotnet.NativeApi.Lib](#)

Assembly: HackRFDotnet.dll

```
public static class HackRfNativeLib.Debug
```

## Inheritance

[object](#) ← HackRfNativeLib.Debug

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Methods

## GetErrorMessage(HackrfError)

Convert @ref hackrf\_error into human-readable string

```
public static extern sbyte* GetErrorMessage(HackrfError errcode)
```

## Parameters

errcode [HackrfError](#)

## Returns

[sbyte](#)\*

# Class HackRfNativeLib.DeviceStreaming

Namespace: [HackRFDotnet.NativeApi.Lib](#)

Assembly: HackRFDotnet.dll

```
public static class HackRfNativeLib.DeviceStreaming
```

## Inheritance

[object](#) ← HackRfNativeLib.DeviceStreaming

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

### ComputeBasebandFilterBandWidth(uint)

Compute nearest valid baseband filter bandwidth to specified value.

The result can be used via [SetBasebandFilterBandwidth\(HackRFDevice\\*, uint\)](#).

```
public static extern uint ComputeBasebandFilterBandWidth(uint bandwidth_hz)
```

#### Parameters

bandwidth\_hz [uint](#)

Desired filter bandwidth in Hz

#### Returns

[uint](#)

Nearest valid filter bandwidth in Hz

### ComputeBasebandFilterBandWidth\_round\_down\_It(uint)

Compute nearest valid baseband filter bandwidth lower than a specified value.

The result can be used via [SetBasebandFilterBandwidth\(HackRFDevice\\*, uint\)](#).

```
public static extern uint ComputeBasebandFilterBandWidth_round_down_lt(uint bandwidth_hz)
```

Parameters

`bandwidth_hz uint`

Desired filter bandwidth in Hz.

Returns

`uint`

The highest valid filter bandwidth lower than `bandwidth_hz` in Hz.

## EnableAmp(HackRFDevice\*, byte)

Enable/disable 14dB RF amplifier.

Enable / disable the ~11dB RF RX/TX amplifiers U13/U25 via controlling switches U9 and U14.

```
public static extern HackrfError EnableAmp(HackRFDevice* device, byte value)
```

Parameters

`device HackRFDevice*`

Device to configure.

`value byte`

Enable (1) or disable (0) amplifier.

Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## EnableAntenna(HackRFDevice\*, byte)

Enable / disable bias-tee (antenna port power).

Enable or disable the **3.3V (max 50mA)** bias-tee (antenna port power). Defaults to disabled.

**NOTE:** the firmware auto-disables this after returning to IDLE mode, so a perma-set is not possible, which means all software supporting HackRF devices must support enabling bias-tee, as setting it externally is not possible like it is with RTL-SDR for example.

```
public static extern HackrfError EnableAntenna(HackRFDevice* device, byte value)
```

Parameters

**device** [HackRFDevice\\*](#)

Device to configure.

**value** [byte](#)

Enable (1) or disable (0) bias-tee.

Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## EnableTxFlush(HackRFDevice\*, HackRFFlushCallback, void\*)

Setup flush (end-of-transmission) callback.

This callback will be called when all the data was transmitted and all data transfers were completed. First parameter is supplied context, second parameter is success flag.

```
public static extern HackrfError EnableTxFlush(HackRFDevice* device, HackRFFlushCallback  
callback, void* flush_ctx)
```

Parameters

**device** [HackRFDevice\\*](#)

Device to configure.

### [callback](#) [HackRFFlushCallback](#)

callback to call when all transfers were completed.

### [flush\\_ctx](#) [void](#) ↳\*

context (1st parameter of callback).

Returns

### [HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## FilterPathName(RfPathFilter)

RF filter path setting enum. Used only when performing explicit tuning using [SetFrequency\(HackRFDevice\\*, ulong, ulong, RfPathFilter\)](#), or can be converted into a human readable string using [FilterPathName\(RfPathFilter\)](#). This can select the image rejection filter(U3, U8 or none) to use - using switches U5, U6, U9 and U11. When no filter is selected, the mixer itself is bypassed.

```
public static extern sbyte* FilterPathName(RfPathFilter path)
```

Parameters

### [path](#) [RfPathFilter](#)

Returns

### [sbyte](#) ↳\*

## GetTransferBufferSize(HackRFDevice\*)

Get USB transfer buffer size.

```
public static extern nuint GetTransferBufferSize(HackRFDevice* device)
```

## Parameters

`device HackRFDevice*`

Unused.

## Returns

`nuint`

Size in bytes.

## GetTransferQueueDepth(HackRFDevice\*)

Get the total number of USB transfer buffers.

```
public static extern uint GetTransferQueueDepth(HackRFDevice* device)
```

## Parameters

`device HackRFDevice*`

Unused.

## Returns

`uint`

Number of buffers.

## InitSweep(HackRFDevice\*, ushort\*, int, uint, uint, uint, Sweep Style)

Initialize sweep mode.

In this mode, in a single data transfer(single call to the RX transfer callback), multiple blocks of size `num_bytes` bytes are received with different center frequencies. At the beginning of each block, a 10-byte frequency header is present in `0x7F - 0x7F - uint64_t frequency(LSBFIRST, in Hz)` format, followed by the actual samples.

Requires USB API version 0x0102 or above!

```
public static extern HackrfError InitSweep(HackRFDevice* device, ushort* frequency_list, int num_ranges, uint num_bytes, uint step_width, uint offset, SweepStyle style)
```

## Parameters

**device** [HackRFDevice](#)\*

Device to configure.

**frequency\_list** [ushort](#)\*

List of start-stop frequency pairs in MHz.

**num\_ranges** [int](#)

Length of array **frequency\_list** (in pairs, so total array length / 2!). Must be less than .

**num\_bytes** [uint](#)

Number of bytes to capture per tuning, must be a multiple of [BYTES\\_PER\\_BLOCK](#).

**step\_width** [uint](#)

Width of each tuning step in Hz.

**offset** [uint](#)

Frequency offset added to tuned frequencies.sample\_rate / 2 is a good value.

**style** [SweepStyle](#)

Sweep style.

## Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## IsStreaming(HackRFDevice\*)

Query device streaming status

```
public static extern HackrfError IsStreaming(HackRFDevice* device)
```

## Parameters

device [HackRFDevice\\*](#)

Device to query

## Returns

[HackrfError](#)

[HACKRF\\_TRUE](#) If the device is streaming, else one of [HACKRF\\_ERROR\\_STREAMING\\_THREAD\\_ERR](#),  
[HACKRF\\_ERROR\\_STREAMING\\_STOPPED](#) or [HACKRF\\_ERROR\\_STREAMING\\_EXIT\\_CALLED](#).

## SetBasebandFilterBandwidth(HackRFDevice\*, uint)

Set baseband filter bandwidth.

Possible values: 1.75, 2.5, 3.5, 5, 5.5, 6, 7, 8, 9, 10, 12, 14, 15, 20, 24, 28MHz, default  $\leq 0.75 \cdot F_s$ .  
The functions [ComputeBasebandFilterBandWidth\(uint\)](#) and [ComputeBasebandFilterBandWidth\\_round\\_down\\_lt\(uint\)](#) can be used to get a valid value nearest to a given value.

Setting the sample rate causes the filter bandwidth to be (re)set to its default  $\leq 0.75 \cdot F_s$  value, so setting sample rate should be done before setting filter bandwidth.

```
public static extern HackrfError SetBasebandFilterBandwidth(HackRFDevice* device,  
uint bandwidth_hz)
```

## Parameters

device [HackRFDevice\\*](#)

device to configure

bandwidth\_hz [uint](#)

baseband filter bandwidth in Hz

## Returns

## [HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant

# SetBiasTOptions(HackRFDevice\*, HackRFBiasTUserSettingReq\*)

Configure bias tee behavior of the HackRF device when changing RF states.

This function allows the user to configure bias tee behavior so that it can be turned on or off automatically by the HackRF when entering the RX, TX, or OFF state. By default, the HackRF switches off the bias tee when the RF path switches to OFF mode.

The bias tee configuration is specified via a bitfield: 0000000TmmRmmOmm

Where setting T/R/O bits indicates that the TX/RX/Off behavior should be set to mode 'mm', 0 = don't modify

mm specifies the bias tee mode:

00 - do nothing 01 - reserved, do not use 10 - disable bias tee 11 - enable bias tee

```
public static extern HackrfError SetBiasTOptions(HackRFDevice* device,  
HackRFBiasTUserSettingReq* req)
```

## Parameters

**device** [HackRFDevice](#)\*

Device to configure.

**req** [HackRFBiasTUserSettingReq](#)\*

Bias tee states, as a bitfield.

## Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

# SetClockSampleRate(HackRFDevice\*, uint, uint)

```
public static extern HackrfError SetClockSampleRate(HackRFDevice* device, uint freq_hz,  
uint divider)
```

## Parameters

device [HackRFDevice\\*](#)

freq\_hz [uint](#)

divider [uint](#)

## Returns

[HackrfError](#)

## SetFrequency(HackRFDevice\*, ulong)

Set the center frequency.

Simple(auto) tuning via specifying a center frequency in Hz.

This setting is not exact and depends on the PLL settings. Exact resolution is not determined, but the actual tuned frequency will be queryable in the future.

```
public static extern HackrfError SetFrequency(HackRFDevice* device, ulong freq_hz)
```

## Parameters

device [HackRFDevice\\*](#)

Device to tune.

freq\_hz [ulong](#)

freq\_hz center frequency in Hz. Defaults to 900MHz. Should be in range 1-6000MHz, but 0-7250MHz is possible. The resolution is ~50Hz, I could not find the exact number.

## Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## SetFrequency(HackRFDevice\*, ulong, ulong, RfPathFilter)

Set the center frequency via explicit tuning.

Center frequency is set to  $f_{\text{center}} = f_{\text{IF}} k \cdot f_{\text{LO}}$  where  $k \in \{-1; 0; 1\}$ , depending on the value of [path](#). See the documentation of [RfPathFilter](#) for details.

```
public static extern HackrfError SetFrequency(HackRFDevice* device, ulong if_freq_hz, ulong  
lo_freq_hz, RfPathFilter path)
```

### Parameters

[device](#) [HackRFDevice](#)\*

Device to tune.

[if\\_freq\\_hz](#) [ulong](#) ↗

Tuning frequency of the MAX2837 transceiver IC in Hz. Must be in the range of 2150-2750MHz.

[lo\\_freq\\_hz](#) [ulong](#) ↗

Tuning frequency of the RFFC5072 mixer/synthesizer IC in Hz. Must be in the range 84.375-5400MHz, defaults to 1000MHz. No effect if [path](#) is set to [RF\\_PATH\\_FILTER\\_BYPASS](#).

[path](#) [RfPathFilter](#)

Filter path for mixer. See the documentation for [RfPathFilter](#) for details.

### Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## SetLnaGain(HackRFDevice\*, uint)

Set LNA gain.

Set the RF RX gain of the MAX2837 transceiver IC ("IF" gain setting) in decibels. Must be in range 0-40dB, with 8dB steps.

```
public static extern HackrfError SetLnaGain(HackRFDevice* device, uint value)
```

## Parameters

**device** [HackRFDevice\\*](#)

Device to configure.

**value** [uint](#)

RX IF gain value in dB.

## Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## SetRxOverrunLimit(HackRFDevice\*, uint)

Set receive overrun limit

When this limit is set, after the specified number of samples (bytes, not whole IQ pairs) missing the device will automatically return to IDLE mode, thus stopping operation. Useful for handling cases like program/computer crashes or other problems. The default value 0 means no limit.

Requires USB API version 0x0106 or above!

```
public static extern HackrfError SetRxOverrunLimit(HackRFDevice* device, uint value)
```

## Parameters

**device** [HackRFDevice\\*](#)

Device to configure.

**value** [uint](#)

Number of samples to wait before auto-stopping.

Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## SetSampleRate(HackRFDevice\*, double)

```
public static extern HackrfError SetSampleRate(HackRFDevice* device, double freq_hz)
```

Parameters

device [HackRFDevice](#)\*

freq\_hz [double](#)

Returns

[HackrfError](#)

## SetTxBlockCompleteCallback(HackRFDevice\*, HackRFTxBlockCompleteCallback)

Setup callback to be called when an USB transfer is completed.

This callback will be called whenever an USB transfer to the device is completed, regardless if it was successful or not (indicated by the second parameter).

```
public static extern HackrfError SetTxBlockCompleteCallback(HackRFDevice* device,  
HackRFTxBlockCompleteCallback callback)
```

Parameters

device [HackRFDevice](#)\*

Device to configure.

callback [HackRFTxBlockCompleteCallback](#)

Callback to call when a transfer is completed.

Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## SetTxUnderrunLimit(HackRFDevice\*, uint)

Set transmit underrun limit

When this limit is set, after the specified number of samples (bytes, not whole IQ pairs) missing the device will automatically return to IDLE mode, thus stopping operation. Useful for handling cases like program/computer crashes or other problems. The default value 0 means no limit.

Requires USB API version 0x0106 or above!

```
public static extern HackrfError SetTxUnderrunLimit(HackRFDevice* device, uint value)
```

Parameters

**device** [HackRFDevice\\*](#)

Device to configure.

**value** [uint](#)

Number of samples to wait before auto-stopping.

Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## SetTxVgaGain(HackRFDevice\*, uint)

Set RF TX gain of the MAX2837 transceiver IC ("IF" or "VGA" gain setting) in decibels. Must be in range 0-47dB in 1dB steps.

```
public static extern HackrfError SetTxVgaGain(HackRFDevice* device, uint value)
```

## Parameters

**device** [HackRFDevice\\*](#)

Device to configure.

**value** [uint](#)

TX IF gain value in dB.

## Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## SetVgaGain(HackRFDevice\*, uint)

Set baseband RX gain of the MAX2837 transceiver IC ("BB" or "VGA" gain setting) in decibels. Must be in range 0-62dB with 2dB steps.

```
public static extern HackrfError SetVgaGain(HackRFDevice* device, uint value)
```

## Parameters

**device** [HackRFDevice\\*](#)

Device to configure.

**value** [uint](#)

RX BB gain value in dB.

## Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## StartRx(HackRFDevice\*, HackRFSampleBlockCallback, void\*)

Start receiving.

Should be called after setting gains, frequency and sampling rate, as these values won't get reset but instead keep their last value, thus their state is unknown.

The callback is called with a [HackrfTransfer](#) object whenever the buffer is full. The callback is called in an async context so no libhackrf functions should be called from it. The callback should treat its argument as read-only.

```
public static extern HackrfError StartRx(HackRFDevice* device, HackRFSampleBlockCallback  
callback, void* rx_ctx)
```

Parameters

**device** [HackRFDevice](#)\*

Device to configure.

**callback** [HackRFSampleBlockCallback](#)

Rx\_callback.

**rx\_ctx** [void](#)\*<sup>↗</sup>

User provided RX context. Not used by the library, but available to **callback** as [rx\\_ctx](#).

Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## StartRxSweep(HackRFDevice\*, HackRFSampleBlockCallback, void\*)

Start RX sweep

See [InitSweep\(HackRFDevice\\*, ushort\\*, int, uint, uint, uint, SweepStyle\)](#) for more info

Requires USB API version 0x0104 or above!

```
public static extern HackrfError StartRxSweep(HackRFDevice* device,  
HackRFSampleBlockCallback callback, void* rx_ctx)
```

## Parameters

**device** [HackRFDevice\\*](#)

Device to start sweeping.

**callback** [HackRFSampleBlockCallback](#)

Rx callback processing the received data.

**rx\_ctx** [void](#)\*

User provided RX context. Not used by the library, but available to **callback** as [rx\\_ctx](#).

## Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## StartTx(HackRFDevice\*, HackRFSampleBlockCallback, void\*)

Start transmitting (TX).

**⚠ Warning:** Transmitting radio signals may be subject to national and international regulations. Use of this function without the appropriate license or authorization may violate FCC regulations (or equivalent regulatory authorities in your region) and could result in legal penalties.

```
public static extern HackrfError StartTx(HackRFDevice* device, HackRFSampleBlockCallback  
callback, void* tx_ctx)
```

## Parameters

**device** [HackRFDevice\\*](#)

Device to configure.

**callback** [HackRFSampleBlockCallback](#)

Tx\_callback.

**tx\_ctx** [void](#)\*

User provided TX context. Not used by the library, but available to **callback** as [tx\\_ctx](#).

Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## StopRx(HackRFDevice\*)

Stop receiving,

```
public static extern HackrfError StopRx(HackRFDevice* device)
```

Parameters

**device** [HackRFDevice](#)\*

device to stop RX on

Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

## StopTx(HackRFDevice\*)

Stop transmission.

```
public static extern HackrfError StopTx(HackRFDevice* device)
```

Parameters

**device** [HackRFDevice](#)\*

Device to stop TX on.

Returns

[HackrfError](#)

[HACKRF\\_SUCCESS](#) on success or [HackrfError](#) variant.

# Class HackRfNativeLib.Devices

Namespace: [HackRFDotnet.NativeApi.Lib](#)

Assembly: HackRFDotnet.dll

```
public static class HackRfNativeLib.Devices
```

## Inheritance

[object](#) ← HackRfNativeLib.Devices

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

### CloseDevice(HackRFDevice\*)

Close a previously opened device

```
public static extern int CloseDevice(HackRFDevice* device)
```

#### Parameters

device [HackRFDevice](#)\*

#### Returns

[int](#)

### DeviceListFree(HackRFDeviceList\*)

Free a previously allocated @ref hackrf\_device\_list list.

```
public static extern void DeviceListFree(HackRFDeviceList* list)
```

Parameters

`list HackRFDeviceList*`

## DeviceListOpen(HackRFDeviceList\*, int, HackRFDevice\*\*)

Open a @ref hackrf\_device from a device list

```
public static extern int DeviceListOpen(HackRFDeviceList* list, int idx,  
HackRFDevice** device)
```

Parameters

`list HackRFDeviceList*`

`idx int`

`device HackRFDevice**`

Returns

`int`

## OpenDevice(HackRFDevice\*\*)

Open first available HackRF device

```
public static extern int OpenDevice(HackRFDevice** device)
```

Parameters

`device HackRFDevice**`

Returns

`int`

## OpenDeviceBySerial(string, HackRFDevice\*\*)

Open HackRF device by serial number

```
public static extern int OpenDeviceBySerial(string desired_serial_number,  
HackRFDevice** device)
```

Parameters

`desired_serial_number` [string](#)

`device` [HackRFDevice](#)\*\*

Returns

[int](#)

## QueryDeviceList()

List connected HackRF devices

```
public static extern HackRFDeviceList* QueryDeviceList()
```

Returns

[HackRFDeviceList](#)\*

## ResetDevice(HackRFDevice\*)

Reset HackRF device

```
public static extern int ResetDevice(HackRFDevice* device)
```

Parameters

`device` [HackRFDevice](#)\*

Returns

[int](#)

## SetDeviceLeds(HackRFDevice\*, byte)

Turn on or off (override) the LEDs of the HackRF device This function can turn on or off the LEDs of the device. There are 3 controllable LEDs on the HackRF one: USB, RX and TX. On the Rad1o, there are 4 LEDs. Each LED can be set individually, but the setting might get overridden by other functions.

The LEDs can be set via specifying them as bits of a 8 bit number @p state, bit 0 representing the first (USB on the HackRF One) and bit 3 or 4 representing the last LED. The upper 4 or 5 bits are unused. For example, binary value 0bxxxxx101 turns on the USB and TX LEDs on the HackRF One.

```
public static extern int SetDeviceLeds(HackRFDevice* device, byte state)
```

Parameters

device [HackRFDevice](#)\*

state [byte](#)

Returns

[int](#)

## SetDeviceUiEnabled(HackRFDevice\*, byte)

Enable / disable UI display (RAD1O, PortaPack, etc.) Enable or disable the display on display-enabled devices (Rad1o, PortaPack)

```
public static extern int SetDeviceUiEnabled(HackRFDevice* device, byte value)
```

Parameters

device [HackRFDevice](#)\*

value [byte](#)

## Returns

[int ↗](#)

# Class HackRfNativeLib.Firmware

Namespace: [HackRFDotnet.NativeApi.Lib](#)

Assembly: HackRFDotnet.dll

```
public static class HackRfNativeLib.Firmware
```

## Inheritance

[object](#) ← HackRfNativeLib.Firmware

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

### BoardIdName(HackrfBoardId)

Convert @ref hackrf\_board\_id into human-readable string

```
public static extern sbyte* BoardIdName(HackrfBoardId board_id)
```

#### Parameters

board\_id [HackrfBoardId](#)

#### Returns

[sbyte](#)\*

### BoardRevName(HackrfBoardRev)

Convert board revision name

```
public static extern nint BoardRevName(HackrfBoardRev board_rev)
```

Parameters

`board_rev` [HackrfBoardRev](#)

Returns

[int](#)

## ClearSpiflashStatus(HackRFDevice\*)

Clear the status registers of the W25Q80BV SPI flash chip See the datasheet for details of the status registers.

```
public static extern int ClearSpiflashStatus(HackRFDevice* device)
```

Parameters

`device` [HackRFDevice](#)\*

Returns

[int](#)

## EraseSpiflash(HackRFDevice\*)

Erase firmware image on the SPI flash Should be followed by writing a new image, or the HackRF will be soft-bricked (still rescuable in DFU mode)

```
public static extern int EraseSpiflash(HackRFDevice* device)
```

Parameters

`device` [HackRFDevice](#)\*

Returns

[int](#)

## GetClkinStatus(HackRFDevice\*, byte\*)

Get CLKIN status Check if an external clock signal is detected on the CLKIN port.

```
public static extern int GetClkinStatus(HackRFDevice* device, byte* status)
```

Parameters

device [HackRFDevice](#)\*

status [byte](#)\*

Returns

[int](#)

## GetMcuState(HackRFDevice\*, HackRFM0State\*)

Get the state of the M0 code on the LPC43xx MCU

```
public static extern int GetMcuState(HackRFDevice* device, HackRFM0State* value)
```

Parameters

device [HackRFDevice](#)\*

value [HackRFM0State](#)\*

Returns

[int](#)

## LookupBoardIdPlatform(HackrfBoardId)

Lookup platform ID (HACKRF\_PLATFORM\_xxx) from board id (@ref hackrf\_board\_id)

```
public static extern uint LookupBoardIdPlatform(HackrfBoardId board_id)
```

Parameters

`board_id` [HackrfBoardId](#)

Returns

[uint](#)

## ReadBoardId(HackRFDevice\*, byte\*)

Read @ref hackrf\_board\_id from a device The result can be converted into a human-readable string via @ref hackrf\_board\_id\_name

```
public static extern int ReadBoardId(HackRFDevice* device, byte* value)
```

Parameters

`device` [HackRFDevice](#)\*

`value` [byte](#)\*

Returns

[int](#)

## ReadBoardPartIdSerialNo(HackRFDevice\*, ReadPartidSerialNo\*)

Read board part ID and serial number Read MCU part id and serial number. See the documentation of the MCU for details!

```
public static extern int ReadBoardPartIdSerialNo(HackRFDevice* device,
ReadPartidSerialNo* read_partid_serialno)
```

Parameters

`device` [HackRFDevice](#)\*

`read_partid_serialno` [ReadPartidSerialNo](#)\*

Returns

[int↗](#)

## ReadBoardRev(HackRFDevice\*, byte\*)

Read board revision of device

```
public static extern int ReadBoardRev(HackRFDevice* device, byte* value)
```

Parameters

device [HackRFDevice\\*](#)

value [byte↗\\*](#)

Returns

[int↗](#)

## ReadMax2837(HackRFDevice\*, byte, ushort\*)

Directly read the registers of the MAX2837 transceiver IC, Intended for debugging purposes only!

```
public static extern int ReadMax2837(HackRFDevice* device, byte register_number,  
ushort* value)
```

Parameters

device [HackRFDevice\\*](#)

register\_number [byte↗](#)

value [ushort↗\\*](#)

Returns

[int↗](#)

## ReadRffc5071(HackRFDevice\*, byte, ushort\*)

Directly read the registers of the RFFC5071/5072 mixer-synthesizer IC Intended for debugging purposes only!

```
public static extern int ReadRffc5071(HackRFDevice* device, byte register_number,  
ushort* value)
```

Parameters

device [HackRFDevice\\*](#)

register\_number [byte](#)

value [ushort](#)\*

Returns

[int](#)

## ReadSi5351c(HackRFDevice\*, ushort, ushort\*)

Directly read the registers of the Si5351C clock generator IC Intended for debugging purposes only!

```
public static extern int ReadSi5351c(HackRFDevice* device, ushort register_number,  
ushort* value)
```

Parameters

device [HackRFDevice\\*](#)

register\_number [ushort](#)

value [ushort](#)\*

Returns

[int](#)

**ReadSpiflash(HackRFDevice\*, uint, ushort, byte\*)**

Read firmware image on the SPI flash Should only be used for firmware verification.

```
public static extern int ReadSpiflash(HackRFDevice* device, uint address, ushort length,  
byte* data)
```

## Parameters

device HackRFDevice\*

address uint ↗

**length** ushort ↗

data byte ↗\*

## Returns

int ↗

**ReadSupportedPlatform(HackRFDevice\*, uint\*)**

Read supported platform of device Returns a combination of @ref HACKRF\_PLATFORM\_JAWBREAKER | @ref HACKRF\_PLATFORM\_HACKRF1\_OG | @ref HACKRF\_PLATFORM\_RAD1O | @ref HACKRF\_PLATFORM\_HACKRF1\_R9

```
public static extern int ReadSupportedPlatform(HackRFDevice* device, uint* value)
```

## Parameters

device HackRFDevice\*

value `uint`<sup>↗\*</sup>

## Returns

int

## ReadUsbApiVersion(HackRFDevice\*, ushort\*)

Read version as MM.mm 16-bit value, where MM is the major and mm is the minor version, encoded as the hex digits of the 16-bit number.

```
public static extern int ReadUsbApiVersion(HackRFDevice* device, ushort* version)
```

Parameters

`device HackRFDevice*`

`version ushort*`

Returns

`int`

## ReadVersion(HackRFDevice\*, byte\*, byte)

Read HackRF firmware version as a string

```
public static extern int ReadVersion(HackRFDevice* device, byte* version, byte length)
```

Parameters

`device HackRFDevice*`

`version byte*`

`length byte`

Returns

`int`

## SetClkoutEnable(HackRFDevice\*, byte)

Enable / disable CLKOUT

```
public static extern int SetClkoutEnable(HackRFDevice* device, byte value)
```

Parameters

device [HackRFDevice\\*](#)

value [byte](#)

Returns

[int](#)

## SetHardwareSyncMode(HackRFDevice\*, byte)

Set hardware sync mode (hardware triggering) See the documentation on hardware triggering for details

```
public static extern int SetHardwareSyncMode(HackRFDevice* device, byte value)
```

Parameters

device [HackRFDevice\\*](#)

value [byte](#)

Returns

[int](#)

## SpiflashStatus(HackRFDevice\*, byte\*)

Read the status registers of the W25Q80BV SPI flash chip See the datasheet for details of the status registers. The two registers are read in order.

```
public static extern int SpiflashStatus(HackRFDevice* device, byte* data)
```

Parameters

`device HackRFDevice*`

`data byte[]*`

Returns

`int[]`

## UsbBoardIdName(HackrfUsbBoardId)

Convert @ref hackrf\_usb\_board\_id into human-readable string.

```
public static extern sbyte* UsbBoardIdName(HackrfUsbBoardId usb_board_id)
```

Parameters

`usb_board_id HackrfUsbBoardId`

Returns

`sbyte[]*`

## WriteCpld(HackRFDevice\*, byte\*, uint)

Write configuration bitstream into the XC2C64A-7VQ100C CPLD device will need to be reset after  
`hackrf_cpld_write`

```
public static extern int WriteCpld(HackRFDevice* device, byte* data, uint total_length)
```

Parameters

`device HackRFDevice*`

`data byte[]*`

`total_length uint[]`

Returns

[int](#)

## WriteRffc5071(HackRFDevice\*, byte, ushort)

Directly write the registers of the RFFC5071/5072 mixer-synthesizer IC Intended for debugging purposes only!

```
public static extern int WriteRffc5071(HackRFDevice* device, byte register_number,  
ushort value)
```

Parameters

device [HackRFDevice](#)\*

register\_number [byte](#)

value [ushort](#)

Returns

[int](#)

## WriteSi5351c(HackRFDevice\*, ushort, ushort)

Directly write the registers of the Si5351 clock generator IC Intended for debugging purposes only!

```
public static extern int WriteSi5351c(HackRFDevice* device, ushort register_number,  
ushort value)
```

Parameters

device [HackRFDevice](#)\*

register\_number [ushort](#)

value [ushort](#)

Returns

[int](#)

## WriteSpiflash(HackRFDevice\*, uint, ushort, byte\*)

Write firmware image on the SPI flash Should only be used for firmware updating. Can brick the device, but it's still rescuable in DFU mode.

```
public static extern int WriteSpiflash(HackRFDevice* device, uint address, ushort length,  
byte* data)
```

Parameters

device [HackRFDevice](#)\*

address [uint](#)

length [ushort](#)

data [byte](#)\*

Returns

[int](#)

## hackrf\_max2837\_write(HackRFDevice\*, byte, ushort)

Directly write the registers of the MAX2837 transceiver IC Intended for debugging purposes only!

```
public static extern int hackrf_max2837_write(HackRFDevice* device, byte register_number,  
ushort value)
```

Parameters

device [HackRFDevice](#)\*

register\_number [byte](#)

value [ushort](#)

Returns

[int ↗](#)

# Class HackRfNativeLib.Operacake

Namespace: [HackRFDotnet.NativeApi.Lib](#)

Assembly: HackRFDotnet.dll

```
public static class HackRfNativeLib.Operacake
```

## Inheritance

[object](#) ← HackRfNativeLib.Operacake

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

### GetOperacakeBoards(HackRFDevice\*, byte\*)

Query connected Opera Cake boards Returns a @ref HACKRF\_OPERACAKE\_MAX\_BOARDS size array of addresses, with @ref HACKRF\_OPERACAKE\_ADDRESS\_INVALID as a placeholder

```
public static extern int GetOperacakeBoards(HackRFDevice* device, byte* boards)
```

#### Parameters

device [HackRFDevice](#)\*

boards [byte](#)\*

#### Returns

[int](#)

### GetOperacakeMode(HackRFDevice\*, byte, OperacakeSwitching Mode\*)

Query Opera Cake mode

```
public static extern int GetOperacakeMode(HackRFDevice* device, byte address,  
OperacakeSwitchingMode* mode)
```

## Parameters

device [HackRFDevice](#)\*

address [byte](#)

mode [OperacakeSwitchingMode](#)\*

## Returns

[int](#)

## OperacakeGpioTest(HackRFDevice\*, byte, ushort\*)

Perform GPIO test on an Opera Cake addon board Value 0xFFFF means "GPIO mode disabled", and hackrf\_operacake advises to remove additional add-on boards and retry. Value 0 means all tests passed. In any other values, a 1 bit signals an error. Bits are grouped in groups of 3. Encoding: 0 - u1ctrl - u3ctrl0 - u3ctrl1 - u2ctrl0 - u2ctrl1

```
public static extern int OperacakeGpioTest(HackRFDevice* device, byte address,  
ushort* test_result)
```

## Parameters

device [HackRFDevice](#)\*

address [byte](#)

test\_result [ushort](#)\*

## Returns

[int](#)

## SetOperacakeDwellTimes(HackRFDevice\*, HackRFOperacakeDwellTime\*, byte)

Setup Opera Cake dwell times in @ref OPERACAKE\_MODE\_TIME mode operation Should be called after @ref hackrf\_set\_operacake\_mode **Note:** this configuration applies to all Opera Cake boards in @ref OPERACAKE\_MODE\_TIME mode

```
public static extern int SetOperacakeDwellTimes(HackRFDevice* device,  
HackRFOperacakeDwellTime* dwell_times, byte count)
```

### Parameters

device [HackRFDevice](#)\*

dwell\_times [HackRFOperacakeDwellTime](#)\*

count [byte](#)

### Returns

[int](#)

## SetOperacakeFrequencyRanges(HackRFDevice\*, HackRFOperacakeFreqRange\*, byte)

Setup Opera Cake frequency ranges in @ref OPERACAKE\_MODE\_FREQUENCY mode operation Should be called after @ref hackrf\_set\_operacake\_mode **Note:** this configuration applies to all Opera Cake boards in @ref OPERACAKE\_MODE\_FREQUENCY mode

```
public static extern int SetOperacakeFrequencyRanges(HackRFDevice* device,  
HackRFOperacakeFreqRange* freq_ranges, byte count)
```

### Parameters

device [HackRFDevice](#)\*

freq\_ranges [HackRFOperacakeFreqRange](#)\*

count [byte](#)

Returns

[int](#)

## SetOperacakeMode(HackRFDevice\*, byte, OperacakeSwitching Mode)

Setup Opera Cake operation mode

```
public static extern int SetOperacakeMode(HackRFDevice* device, byte address,  
OperacakeSwitchingMode mode)
```

Parameters

device [HackRFDevice](#)\*

address [byte](#)

mode [OperacakeSwitchingMode](#)

Returns

[int](#)

## SetOperacakePorts(HackRFDevice\*, byte, byte, byte)

Setup Opera Cake ports in @ref OPERACAKE\_MODE\_MANUAL mode operation Should be called after @ref hackrf\_set\_operacake\_mode. A0 and B0 must be connected to opposite sides (A->A and B->B or A->B and B->A but not A->A and B->A or A->B and B->B)

```
public static extern int SetOperacakePorts(HackRFDevice* device, byte address, byte port_a,  
byte port_b)
```

Parameters

device [HackRFDevice](#)\*

address [byte](#)

`port_a` [byte](#)

`port_b` [byte](#)

Returns

[int](#)

## SetOperacakeRanges(HackRFDevice\*, byte\*, byte)

Setup Opera Cake frequency ranges in @ref OPERACAKE\_MODE\_FREQUENCY mode operation Old function to set ranges with. Use @ref hackrf\_set\_operacake\_freq\_ranges instead! **Note:** this configuration applies to all Opera Cake boards in @ref OPERACAKE\_MODE\_FREQUENCY mode

```
[Obsolete("Use hackrf_set_operacake_freq_ranges instead.")]
public static extern int SetOperacakeRanges(HackRFDevice* device, byte* ranges,
byte num_ranges)
```

Parameters

`device` [HackRFDevice](#)\*

`ranges` [byte](#)\*

`num_ranges` [byte](#)

Returns

[int](#)

# Namespace HackRFDotnet.NativeApi.Structs

## Structs

[HackRBiasTUserSettingReq](#)

[HackRFBoolUserSetting](#)

[HackrfTransfer](#)

## Delegates

[HackRFFlushCallback](#)

[HackRFSampleBlockCallback](#)

[HackRFTxBlockCompleteCallback](#)

# Struct HackRFBiasTUserSettingReq

Namespace: [HackRFDotnet.NativeApi.Structs](#)

Assembly: HackRFDotnet.dll

```
public struct HackRFBiasTUserSettingReq
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

## Fields

off

```
public HackRFBoolUserSetting off
```

Field Value

[HackRFBoolUserSetting](#)

rx

```
public HackRFBoolUserSetting rx
```

Field Value

[HackRFBoolUserSetting](#)

tx

```
public HackRFBoolUserSetting tx
```

## Field Value

[HackRFBoolUserSetting](#)

# Struct HackRFBoolUserSetting

Namespace: [HackRFDotnet.NativeApi.Structs](#)

Assembly: HackRFDotnet.dll

```
public struct HackRFBoolUserSetting
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

## Fields

### change\_on\_mode\_entry

Change value on mode entry

```
public bool change_on_mode_entry
```

Field Value

[bool](#)

### do\_update

If true, update default values

```
public bool do_update
```

Field Value

[bool](#)

### enabled

Enabled

```
public bool enabled
```

Field Value

[bool](#) ↗

# Delegate HackRFFlushCallback

Namespace: [HackRFDotnet.NativeApi.Structs](#)

Assembly: HackRFDotnet.dll

```
public delegate void HackRFFlushCallback(nint flush_ctx, int status)
```

Parameters

flush\_ctx [nint](#)

status [int](#)

# Delegate HackRFSampleBlockCallback

Namespace: [HackRFDotnet.NativeApi.Structs](#)

Assembly: HackRFDotnet.dll

```
public delegate int HackRFSampleBlockCallback(HackrfTransfer* transfer)
```

Parameters

transfer [HackrfTransfer\\*](#)

Returns

[int↗](#)

# Delegate HackRFTxBlockCompleteCallback

Namespace: [HackRFDotnet.NativeApi.Structs](#)

Assembly: HackRFDotnet.dll

```
public delegate void HackRFTxBlockCompleteCallback(HackrfTransfer* transfer, int status)
```

Parameters

transfer [HackrfTransfer](#)\*

status [int](#)

# Struct HackrfTransfer

Namespace: [HackRFDotnet.NativeApi.Structs](#)

Assembly: HackRFDotnet.dll

```
public struct HackrfTransfer
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

## Fields

### buffer

```
public byte* buffer
```

#### Field Value

[byte](#)\*

### buffer\_length

```
public int buffer_length
```

#### Field Value

[int](#)

### device

```
public HackRFDevice* device
```

Field Value

[HackRFDevice](#)\*

rx\_ctx

```
public void* rx_ctx
```

Field Value

[void](#)\*

tx\_ctx

```
public void* tx_ctx
```

Field Value

[void](#)\*

valid\_length

```
public int valid_length
```

Field Value

[int](#)

# Namespace HackRFDotnet.NativeApi.Structs.Devices

## Structs

[HackRFDevice](#)

[HackRFDeviceList](#)

[HackRFOperacakeDwellTime](#)

[HackRFOperacakeFreqRange](#)

# Struct HackRFDevice

Namespace: [HackRFDotnet.NativeApi.Structs.Devices](#)

Assembly: HackRFDotnet.dll

```
public struct HackRFDevice
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

# Struct HackRFDeviceList

Namespace: [HackRFDotnet.NativeApi.Structs.Devices](#)

Assembly: HackRFDotnet.dll

```
public struct HackRFDeviceList
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

## Fields

### devicecount

Number of connected HackRF devices

```
public int devicecount
```

Field Value

[int](#)

### serial\_numbers

Array of human-readable serial numbers. Each entry can be NULL.

```
public char** serial_numbers
```

Field Value

[char](#)\*\*

### usb\_board\_ids

ID of each board, based on USB product ID

```
public HackrfUsbBoardId* usb_board_ids
```

Field Value

[HackrfUsbBoardId\\*](#)

## usb\_device\_index

USB device index for each HW entry

```
public int* usb_device_index
```

Field Value

[int↗\\*](#)

## usb\_devicecount

Number of all queried USB devices

```
public int usb_devicecount
```

Field Value

[int↗](#)

## usb\_devices

All USB devices (as libusb\_device\*\* array)

```
public void** usb_devices
```

Field Value

void ↴ \*\*

# Struct HackRFOperacakeDwellTime

Namespace: [HackRFDotnet.NativeApi.Structs.Devices](#)

Assembly: HackRFDotnet.dll

```
public struct HackRFOperacakeDwellTime
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

## Fields

### dwell

Dwell time for port (in number of samples)

```
public uint dwell
```

Field Value

[uint](#)

### port

Port to connect A0 to (B0 mirrors this choice) Must be one of operacake\_ports

```
public byte port
```

Field Value

[byte](#)

# Struct HackRFOperacakeFreqRange

Namespace: [HackRFDotnet.NativeApi.Structs.Devices](#)

Assembly: HackRFDotnet.dll

```
public struct HackRFOperacakeFreqRange
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

## Fields

### freq\_max

Stop frequency (in MHz)

```
public ushort freq_max
```

#### Field Value

[ushort](#)

### freq\_min

Start frequency (in MHz)

```
public ushort freq_min
```

#### Field Value

[ushort](#)

### port

Port (A0) to use for that frequency range. Port B0 mirrors this. Must be one of operacake\_ports

```
public byte port
```

Field Value

[byte](#) ↗

# Namespace HackRFDotnet.NativeApi.Structs. System

## Structs

[HackRFM0State](#)

[ReadPartidSerialNo](#)

# Struct HackRFM0State

Namespace: [HackRFDotnet.NativeApi.Structs.System](#)

Assembly: HackRFDotnet.dll

```
public struct HackRFM0State
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

## Fields

### active\_mode

Active mode. Same values as requested\_mode

```
public uint active_mode
```

#### Field Value

[uint](#)

### error

Error that caused M0 to revert to IDLE. 0(NONE), 1(RX\_TIMEOUT), 2(TX\_TIMEOUT), 3(MISSED\_DEADLINE)

```
public int error
```

#### Field Value

[uint](#)

### longest\_shortfall

Longest shortfall in bytes

```
public uint longest_shortfall
```

Field Value

[uint](#)

## m0\_count

Number of bytes transferred by the M0

```
public uint m0_count
```

Field Value

[uint](#)

## m4\_count

Number of bytes transferred by the M4

```
public uint m4_count
```

Field Value

[uint](#)

## next\_mode

Mode which will be switched to when threshold is reached

```
public uint next_mode
```

Field Value

[uint](#)

## num\_shortfalls

Number of shortfalls

```
public uint num_shortfalls
```

Field Value

[uint](#)

## request\_flag

Request flag. 0 = completed, others = pending

```
public ushort request_flag
```

Field Value

[ushort](#)

## requested\_mode

Requested mode. Possible values: 0(IDLE), 1(WAIT), 2(RX), 3(TX\_START), 4(TX\_RUN)

```
public ushort requested_mode
```

Field Value

[ushort](#)

## shortfall\_limit

Shortfall limit in bytes

```
public uint shortfall_limit
```

Field Value

[uint](#) ↗

## threshold

Threshold m0\_count value (in bytes) for next mode change

```
public uint threshold
```

Field Value

[uint](#) ↗

# Struct ReadPartidSerialNo

Namespace: [HackRFDotnet.NativeApi.Structs.System](#)

Assembly: HackRFDotnet.dll

```
public struct ReadPartidSerialNo
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

## Fields

### part\_id

```
public uint* part_id
```

#### Field Value

[uint](#)\*

### serial\_no

```
public uint* serial_no
```

#### Field Value

[uint](#)\*