RFcreations

...a different perspective

morephE.

etherstore

Etherstore User Guide V2.0

1. INTRODUCTION

Etherstore uses fast solid-state storage to enable you to record and play back the full RF bandwidth of the Etherstore. You can either play back your own recordings, or upload waveforms you have created yourself.

To get started with recording, see Create new recording.

To get started with your own waveforms, see Upload file and Playback.

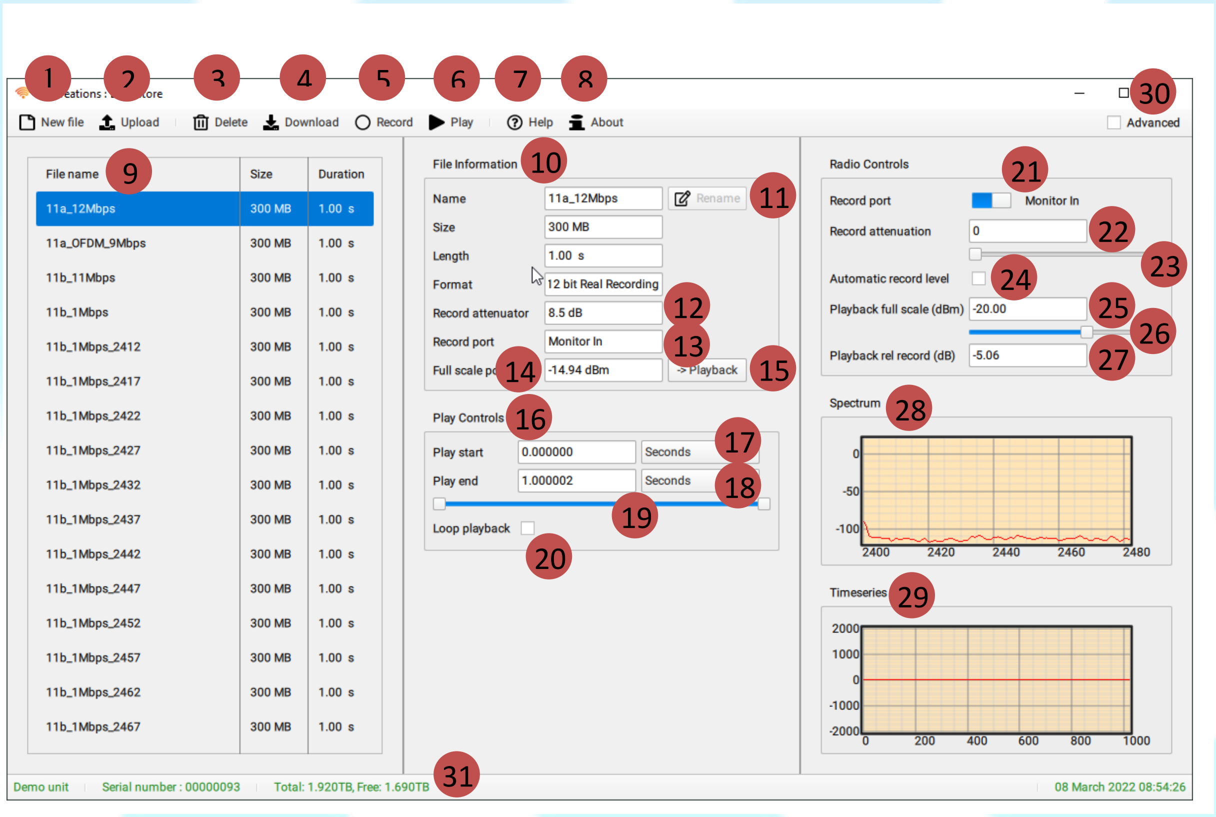
For information about the user interface, see GUI: Basic.

To transfer recordings between Etherstores, see Transfer files between Etherstores.

WARNING	You should exit the application on the host PC before powering down Etherstore to enable an orderly shutdown of the internal disk and prevent data loss.

2. GUI: BASIC

This is the standard interface you will see when you start the application.



File name	Size	Duration
11a_12Mbps	300 MB	1.00 s
11a_OFDM_9Mbps	300 MB	1.00 s
11b_11Mbps	300 MB	1.00 s
11b_1Mbps	300 MB	1.00 s
11b_1Mbps_2412	300 MB	1.00 s
11b_1Mbps_2417	300 MB	1.00 s
11b_1Mbps_2422	300 MB	1.00 s
11b_1Mbps_2427	300 MB	1.00 s
11b_1Mbps_2432	300 MB	1.00 s
11b_1Mbps_2437	300 MB	1.00 s
11b_1Mbps_2442	300 MB	1.00 s
11b_1Mbps_2447	300 MB	1.00 s
11b_1Mbps_2452	300 MB	1.00 s
11b_1Mbps_2457	300 MB	1.00 s
11b_1Mbps_2462	300 MB	1.00 s
11b_1Mbps_2467	300 MB	1.00 s

File Information
Name: 11a_12Mbps (Rename)
Size: 300 MB
Length: 1.00 s
Format: 12 bit Real Recording
Record attenuator: 8.5 dB
Record port: Monitor In
Full scale po: -14.94 dBm (-> Playback)

Play Controls
Play start: 0.000000 Seconds
Play end: 1.000002 Seconds
Loop playback: ☐

Radio Controls
Record port: Monitor In
Record attenuation: 0
Automatic record level: ☐
Playback full scale (dBm): -20.00
Playback rel record (dB): -5.06

Demo unit | Serial number : 00000093 | Total: 1.920TB, Free: 1.690TB | 08 March 2022 08:54:26

1. Create new file to record into. See Create new recording.
2. Upload a waveform file to the Etherstore. See Upload file.
3. Delete the selected file.

4. Download the selected file. See Download file.
5. Start recording into the currently selected file.

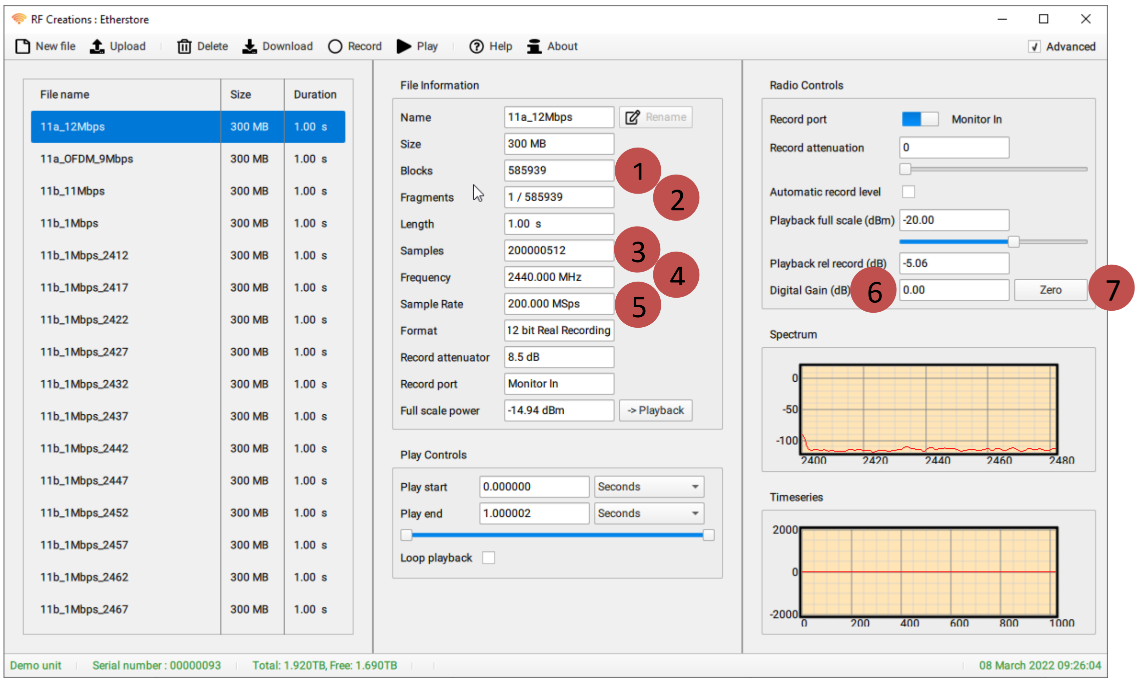
WARNING	This will overwrite the file. If the file is not empty, then a warning dialog will be displayed.

6. Play the selected file.
7. Display this help file.
8. Application information.
9. List of files on the Etherstore. Select the file for Delete / Download / Record / Play.
10. Information about the currently selected file.
11. You can change the file name here. Press the Rename button to save the new name to the Etherstore.
12. The format of the current file. See Internal file formats.
13. The attenuation and port that were used during recording for the current file.
14. Full scale power of the current recording.
15. Set this as the transmit power, so the power at the port is the same as during recording.
16. Controls for Playback
17. Skip the beginning of the file. This can be specified in samples, seconds, minutes, or hours
18. End the playback early.
19. The start and end point of the playback can also be controlled using this slider.
20. Loop the playback forever. If this is enabled, the start and end point are ignored.
21. Click this toggle switch to switch the port for recording between the "Monitor In" and "Tx / Rx" port of the Etherstore.
22. Set the receive attenuation. See Recording.
23. Slider to set receive attenuation.
24. Set receive attenuation automatically. Wait 5-10 seconds after enabling this before starting a recording.
25. Set transmit full scale power.
26. Slider to set transmit full scale power.
27. For recordings this allows you to set the transmit power relative to the power recorded.
28. A spectrum plot of the currently received samples, showing frequency (Hz) vs Power (dBm).
29. The raw ADC data. The ADC is 12-bit, so the full-scale range is -2048 to 2047. You must set the receive attenuation high enough that the waveform does not hit the limits of this graph to get an accurate recording.
30. Enable GUI: Advanced mode.
31. The status bar shows the total and free space on the disk and the current progress of any transfers between the host PC and the Etherstore, playbacks or recordings that are in

progress. When a transfer, playback or recording is in progress, a cancel button will be visible here

3. GUI: ADVANCED

If you enable the advanced checkbox in the upper right of the GUI, you will get some extra information



File name	Size	Duration
11a_12Mbps	300 MB	1.00 s
11a_OFDM_9Mbps	300 MB	1.00 s
11b_11Mbps	300 MB	1.00 s
11b_1Mbps	300 MB	1.00 s
11b_1Mbps_2412	300 MB	1.00 s
11b_1Mbps_2417	300 MB	1.00 s
11b_1Mbps_2422	300 MB	1.00 s
11b_1Mbps_2427	300 MB	1.00 s
11b_1Mbps_2432	300 MB	1.00 s
11b_1Mbps_2437	300 MB	1.00 s
11b_1Mbps_2442	300 MB	1.00 s
11b_1Mbps_2447	300 MB	1.00 s
11b_1Mbps_2452	300 MB	1.00 s
11b_1Mbps_2457	300 MB	1.00 s
11b_1Mbps_2462	300 MB	1.00 s
11b_1Mbps_2467	300 MB	1.00 s

File Information

Name	11a_12Mbps
Size	300 MB
Blocks	585939
Fragments	1 / 585939
Length	1.00 s
Samples	200000512
Frequency	2440.000 MHz
Sample Rate	200.000 MSps
Format	12 bit Real Recording
Record attenuator	8.5 dB
Record port	Monitor In
Full scale power	-14.94 dBm

Play Controls

Play start	0.000000	Seconds
Play end	1.000002	Seconds
Loop playback	☐	

Radio Controls

Record port	Monitor In
Record attenuation	0
Automatic record level	☐
Playback full scale (dBm)	-20.00
Playback rel record (dB)	-5.06
Digital Gain (dB)	0.00

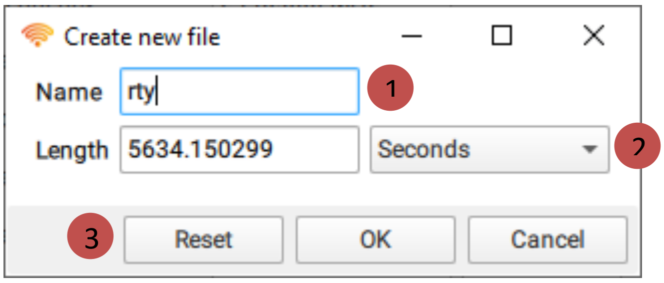
Demo unit | Serial number : 00000093 | Total: 1.920TB, Free: 1.690TB | 08 March 2022 09:26:04

1. Number of blocks used on the SSD.
2. Number of fragments the file is broken into / Size of smallest fragment. If the smallest fragment is very small, or the number of fragments very high, then you may have problems with recording / playback of this file, and you should delete some files on the Etherstore to free up space.
3. Number of samples in the file.
4. Centre frequency of the file.
5. Sample rate of the file.
6. You can set extra digital gain here. Some recordings will be of very low amplitude, which limits the maximum power you can transmit them at. You can compensate for this using digital gain. However, setting the digital gain too high will cause clipping when the amplitude exceeds the full-scale range of the internal DAC.
7. Reset the digital gain to 0.

4. CREATE NEW RECORDING

Before recording, you must first create an empty file to record into. Click 🗋 to show the dialog and enter the name and length. Once the file is created, simply press ○ on the toolbar to begin recording. See Recording for more information.

4.1. Dialog



Name: rty

Length: 5634.150299 Seconds

1. Set file name shown in the file list.
2. Length of the new recording, which can be specified in samples, seconds, minutes, or hours. The default value is the maximum length available in the remaining disk space.
3. Reset to maximum length.

5. PLAYBACK

Before playing a file you should:

1. Set the desired playback range in "Play Controls", or enable "Loop" to play the file repeatedly until it is manually cancelled.

2. Set the desired transmit power in the "Tx dBm" field or using the slider below. The transmit power is calibrated for a full-scale sine wave, for example the first Python example will be output at that power. Many other waveforms will be backed off from this power to accommodate the peak power, so the actual average power will be lower. This will apply to all recordings. It's up to you to work out what the offset is for your specific waveform.

3. In GUI: Advanced you can add digital gain to increase the available transmit power range, but this may cause clipping. It should not exceed the dBFS value of the highest amplitude sample in the waveform. For example, if it's a 12 bit waveform, the maximum value is 2047, so if the maximum absolute value actually present in the file is 123 you can add up to 20 * log10(2047 / 123) = 24dB of digital gain.

4. To start playback press ▶, and to cancel press ⊘ in the status bar.

6. RECORDING

Before recording you should:

1. Select the port you're using.

2. Select an appropriate Rx attenuation. This should be set as low as possible for the best noise figure, but high enough that the peak power expected fits into the full-scale range of the ADC. You can use the ADC Data display to check this. If you check "Rx Atten Auto" it will continually select an appropriate power based on the last ~5 seconds of data. This will be automatically disabled during a recording so that it is possible to correctly reproduce the waveform power on playback. The Spectrum plot will show you how the noise floor is affected and provides a sanity check to ensure your signal is being received.

To start recording press ○.

7. UPLOAD FILE

To upload a waveform you have created click and enter the information required in the Dialog. There are 3 formats supported for upload:

1. 12 bit real signed integer
2. 16 bit real signed integer
3. 16 bit complex, which should have the real and imaginary parts interleaved

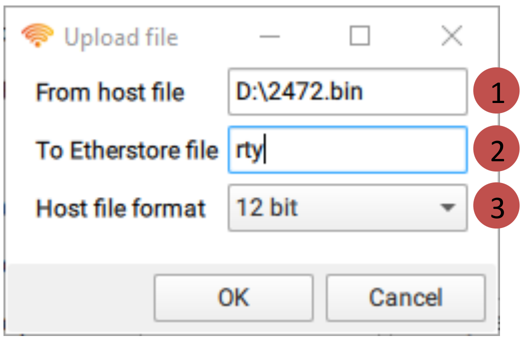
The file is interpreted as a plain binary file with all samples packed. A complex waveform will be centred at 2440MHz on playback, whereas a real waveform is effectively single sided with 0Hz at 2390MHz. Real waveforms are played at 200MSPS, complex at 100MSPS.

NOTE Uploading a recording you previously downloaded in "raw" mode as 12 bit real may appear to work, but the correction factors will be lost, so there will be a frequency dependent amplitude error when it is played.

See Python example for an example of how to create a file for upload using python and numpy.

See Playback for how to play your file once you have uploaded it.

7.1 Dialog


Upload file
From host file: D:\2472.bin
To Etherstore file: rty
Host file format: 12 bit

1. File name on PC. Click here to reopen the file dialog.
2. File name on Etherstore.
3. Format of file. See Internal file formats.

7.2 Python example

To create a 16 bit real file with a full scale sine wave at 2430MHz (2390 + 40):

```
from numpy import int16, pi, sin, arange
f = 40
x = (32767*sin(2*pi*f/200*arange(1e8))).astype(int16)
x.tofile("sin16.bin")
```

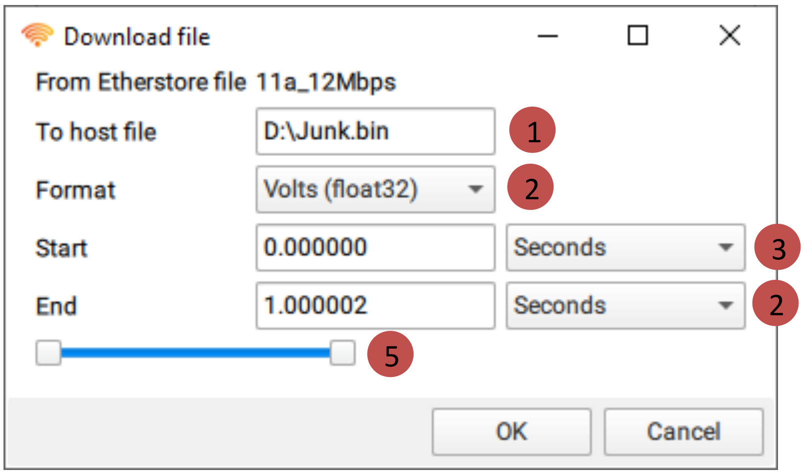
To create a 16 bit complex file with a sine wave at 2450MHz (2440 + 10), -12dBFS:

```
from numpy import int16, pi, sin, cos, arange, empty
x = empty(int(1e8), dtype=int16)
f = 10
x[::2] = 2**13*sin(2*pi*f/100*arange(len(x)/2))
x[1::2] = 2**13*cos(2*pi*f/100*arange(len(x)/2))
x.tofile("cplx16.bin")
```

8. DOWNLOAD FILE

Press ⭳ to start downloading the current file. A file selection dialog will pop up to choose the destination location, followed by a Dialog to choose the format and length to download.

8.1. Dialog


Download file

From Etherstore file 11a_12Mbps

To host file	D:\Junk.bin	
Format	Volts (float32)	
Start	0.000000	Seconds
End	1.000002	Seconds

1. File name on PC. Click here to reopen the file dialog.
2. Format of file to save. See File formats.
3. Skip the beginning of the file.
4. Skip the end of the file.
5. Set the start and end point by slider.

9. TRANSFER FILES BETWEEN ETHERSTORES

"Raw" files downloaded using Etherstore V2.0 software or later can be transferred between Etherstores or uploaded to the same Etherstore without loss of information regarding the state of the receiver during recording, ie the information concerning the receiver port and frontend attenuation is preserved.

To transfer a file between Etherstores:

1. Download the file from the first Etherstore using “Raw (int12 packed)” format
2. Upload the file to the second Etherstore using “12bit” format.

10. FILE FORMATS

There are 2 output file formats supported for recordings, "Volts" and "Raw":

1. "Volts" returns the file stored in 32-bit floating point format, with the correction filter applied and scaled to volts across 50Ω at the input to the Etherstore.
2. "Raw" returns the raw 12-bit signed ADC samples. See Internal file formats.

Uploaded files are always returned in the same format they were uploaded with.

In "Volts" mode, the file saved is much larger ($^{32}/_{12}$ x bigger). We intentionally set the meaningless low order bits to 0, so the file is compressible. To save space, you can put it into an archive such as a .zip file, or enable file compression in the operating system. On Windows, you can do this by right clicking on the file, click 'Properties', click 'Advanced', check "Compress contents to save disk space". This will reduce the disk usage by ~30%. This will only work on a local disk with the default NTFS filesystem, the option is not available on a FAT formatted USB drive, for example.

10.1 Internal file formats

There are four file formats supported on the disk:

1. 12 bit Real
2. 16 bit Real
3. 16 bit Complex
4. 12 bit Real Recording

All should be packed binary signed integers. The first 3 are Upload file formats. The recording format is similar to the 12 bit real format, it is also 12 bit packed integers, but it also contains some information about the port and attenuation used for the recording to allow an amplitude calibration to be applied. The calibration is applied automatically during playback, or when you download the file in "Volts" mode. If you download a file in "Raw" mode, you should apply the correction yourself. The correction is available as comma separated floating point values in the "Rx FIR" field in GUI: Advanced. It is a 39 tap FIR filter that should be applied after extracting the 12 bit "Raw" values and converting to floating point.

11. PYTHON API

Support is provided for driving the Moreph30 directly from Python on a Windows platform. This support is available for Python 2.7 onwards. Both 32 bit and 64 bit versions are available.

11.1 Prerequistes

To import the Etherstore module, include the location of the appropriate PyRFCreations library on the current path. Then import the following:

```
sys.path.append('D:/Program Files/RF Creations/PyRFCreations_3v8_64bit')

from RFCreations import MorephDevice, MorephSearch, MorephInterface,
EtherStoreInterface, vector_fd
```

11.3 Connecting to the Etherstore

To connect to the Etherstore it is first necessary to create a search engine by calling MorephSearch.get(). Having obtained a search engine, the following operations should be performed:

1. Attach a callback to the search engine by invoking the callback() method. The callback will process the Moreph30s and Etherstores that are discovered by the search engine.

2. Start the search engine by invoking the start() method.

Whenever a Moreph30 or Etherstore is discovered, the callback will be entered with a handle to the device and a flag to indicate whether it was discovered on USB or Ethernet. In the example code below, the callback appends the device to a list if it is a USB device or an Ethernet device.

In the example program, the main thread monitors the contents of the list. Whenever a new entry is placed on the list, the main thread obtains an interface to the device by invoking the MorephInterface() method. Having obtained an interface to the device, the friendly name, serial number and current IP address are interrogated. If these satisfy certain search criteria, then the required device has been found and opened, otherwise the entry is discarded from the list.

Once the required device has been found, the search engine is stopped by invoking the stop() method.

```
from __future__ import print_function
import sys, struct
from collections import namedtuple
from time import sleep

sys.path.append('D:/Users/timbo/build-pyRFCreations_MSVC2017_32bit_2v7-Release')

from RFCreations import MorephDevice, MorephSearch, MorephInterface,
SapphireInterface, vector_uchar, vector_float, vector_int16, vector_char

MorephName = "My first Moreph"
MorephSerialNumber = 172

"""
Connect to first device found which matches either MorephName or MorephSerialNumber
"""

def connect():
    moreph = []

    ms = MorephSearch.get()

    def callback(m):
        if ( (m.type == MorephDevice.USB) | (m.type == MorephDevice.Eth) ):
            moreph.append(m)
    ms.set_callback(callback)
    ms.start()

    found = False
    while True:
        while len(moreph):
            try:
                transport = moreph[0].getTransport();
                mi = MorephInterface(transport)
                name = mi.getFriendlyName()
                sn = mi.getSerialNumber()
                print( "Found" , name , sn )
                if( (name == MorephName) | (sn == MorephSerialNumber) ):
                    found = True
                    break
                del moreph[0]
                break
            except:
                del moreph[0]
        if found:
            break
        sleep(0.1)

    print( "Stop" )
    ms.stop()

    print("Opening Etherstore" )

    return transport
```

11.4 Launching the Etherstore application

The Etherstore application is launched by calling EtherStoreInterface(). This returns an interface to the Etherstore application.

```
    # Connect to an Etherstore
    transport = connect()

    # Launch the EtherStore application
    etherStore = EtherStoreInterface(transport)
```

11.5 Example code

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 11 19:29:36 2017

@author: tim
"""
from __future__ import print_function
import sys

sys.path.append('D:/Program Files/RF Creations/PyRFCreations_3v8_64bit')

from time import sleep
from RFCreations import MorephDevice, MorephSearch, MorephInterface, EtherStoreInterface, vector_fd

"""
Set either the name or the serial to be the same as the Moreph device you wish to connect to
"""
MorephName = "Demo unit"
MorephSerialNumber = 147

etherStore = 0

"""
Connect to first device found which matches either MorephName or MorephSerialNumber
"""

def connect():
    moreph = []

    ms = MorephSearch.get()

    def callback(m):
        if (m.type == MorephDevice.USB):
            moreph.append(m)

    ms.set_callback(callback)
    ms.start()

    found = False
    while True:
        while len(moreph):
            try:
                transport = moreph[0].getTransport();
                mi = MorephInterface(transport)
                name = mi.getFriendlyName()
                sn = mi.getSerialNumber()
                print( "Found" , name , sn )
                """
                To connect to the first Moreph device found, ignore test and set found = True
                """
                if( (name == MorephName) | (sn == MorephSerialNumber) ):
                    found = True
                    break
                del moreph[0]
                break
            except:
                del moreph[0]
        if found:
            break
        sleep(0.1)

    ms.stop()

    print("Opening Moreph" )

    return transport

def listFiles():
    fds = etherStore.listFiles()
    for fd in fds:
        print( etherStore.fileName(fd) )
        print( "    Duration : " , etherStore.fileDuration(fd) , "s" )
        print( "    Rx Atten : " , etherStore.fileRxAttenuation(fd) , "dB" )
        if( etherStore.fileRxPort(fd) > 0 ):
            print( "    Rx Port  : Monitor In" );
        elif( etherStore.fileRxPort(fd) == 0 ):
            print( "    Rx Port  : Tx/Rx" );
        else:
            print( "    Rx Port  : Unknown" );
        if( etherStore.fileFormat(fd) == 0 ):
            print( "    Format   : 12bit Real" );
        elif( etherStore.fileFormat(fd) == 1 ):
            print( "    Format   : 16bit Real" );
        elif( etherStore.fileFormat(fd) == 2 ):
            print( "    Format   : 12bit Real Recording" );
        elif( etherStore.fileFormat(fd) == 5 ):
            print( "    Format   : 16bit Complex" );
        else:
            print( "    Format   : Unknown" );

def findFile( name ):
    fds = etherStore.listFiles()
    for fd in fds:
        if( etherStore.fileName(fd) == name ):
            return fd
    return -1

"""
Main script
"""

if __name__ == '__main__':

    # Connect to an Etherstore
    transport = connect()

    # Launch the EtherStore application
    etherStore = EtherStoreInterface(transport)

    # List the files on the EtherStore
    print( "Existing files" )
    listFiles()

    # Create a new file of duration 10s
    # A file descriptor to the file is returned
    print( "Create new file" )
    fd = etherStore.createFile( 10 , "New file" )
    listFiles()

    # Rename the new file
    print( "Rename new file" )
    etherStore.renameFile( fd , "Renamed file" )
    listFiles()

    # Find the file descriptor for a named file
```

```
fd = findFile( "Renamed file" )
print( "Found file descriptor " , fd , " for file 'Renamed file'" )

# Set the receiver to Monitor In port and 0 attenuation
print( "Set receiver port to Monitor In" )
etherStore.rxSetup( 1 , 0 );

# Turn on AGC to find received signal level
print( "Turn on AGC" )
etherStore.setAGC( 1 )

# Search for the maximum attenuation that the AGC thinks is required
rxAtten = 0
for k in range(5):
    atten = etherStore.getAtten()
    print( "New attenuation value " , atten , "dB" )
    if( atten > rxAtten ):
        rxAtten = atten
print( "Recording attenuation is " , rxAtten , "dB" )

# Turn off the AGC
print( "Turning off AGC" )
etherStore.setAGC( 0 )

# Set the receiver to Monitor In port and appropriate attenuation
print( "Setting up receiver for Monitor In and attenuation " , rxAtten )
etherStore.rxSetup( 1 , rxAtten );

# Record into the new file
print( "Start recording" )
etherStore.record( fd )

# Wait for the recording to complete with a timeout of 15s (expected time to record is 10s)
print( "Waiting for recording to end" )
etherStore.waitRecord( 15 )
print( "Recording ended" )

# Setup the transmitter to playback at maximum possible power
print( "Set transmitter for maximum power" )
etherStore.txSetup( 5 )

# Playback part of the recording starting at 2s and for a duration of 4s
print( "Playback from 2s to 6s" )
```

```
etherStore.playback( fd , 2 , 4 )

# Wait for playback to end with a timeout of 10s (expected time to playback is 4s)
print( "Wait for playback" )
etherStore.waitPlayback( 10 )
print( "Playback ended" )

# Playback recording at same level as received, if this is too loud then use maximum possible power
print( "Set transmitter to playback at same level as recording" )
level = etherStore.recordLevel( fd )
if( level > 5 ):
    level = 5
print( "Setting transmit level to " , level )
etherStore.txSetup( level )

# Playback the recording, repeating forever
print( "Playback with repeat forever" )
etherStore.playbackForever( fd )
sleep(15)

# Cancel playback
print( "Cancelling playback" )
etherStore.cancelPlayback()

# Delete the file
print( "Delete file" )
etherStore.deleteFile( fd )
listFiles()

# Close the application
etherStore.exitApp()
etherStore = 0
transport = 0
```

12 PYTHON LIBRARY REFERENCE

This section lists the Python library commands which are available for the Etherstore application.

12.1 listFiles

listFiles() will return an list of the file descriptors for the files held on the Etherstore,

```
fds = etherStore.listFiles()
fds is a list of file descriptors.
```

12.2 filename

filename() will return the name of a file associated with a file descriptor.

name = etherStore.fileName(fd)

name is the name of the file associated with file descriptor **fd**

12.3 fileDuration

fileDuration() will return the duration of a file associated with a file descriptor.

duration = etherStore.fileDuration(fd)

duration is the duration in seconds of the file associated with file descriptor **fd**

12.4 fileRxAttenuation

fileRxAttenuation() will return the receiver frontend attenuation used during the recording of the file

attenuation = etherStore.fileRxAttenuation(fd)

attenuation is the receiver frontend attenuation used during the recording of the file associated with file descriptor **fd**

12.5 fileRxPort

fileRxPort() returns the receiver port used during the recording of the file

port = etherStore.fileRxPort(fd)

port is the receiver port used during the recording of the file associated with file descriptor **fd**. Possible values are:

0: Monitor In
1: Tx/Rx
any other value indicates that the port is unknown (for example, it is user generated file)

12.6 fileFormat

fileFormat() returns the format of the file

format = etherStore.fileFormat(fd)

format is the format of the file associated with file descriptor **fd**. Possible values are:

0: 12bit real data
1: 16bit real data
2: 12bit real recording
3: 16bit complex data

12.7 createFile

createFile() creates a new file on the Etherstore into which data can be recorded.

```
fd = etherStore.createFile( duration , name )
```

```
fd is the file descriptor associated with the newly created file
duration is the length of the file to create in seconds
name is the name of the file to create
```

12.8 renameFile

renameFile() changes the name of a file held on the Etherstore.

```
etherStore.renameFile( fd , newName )
```

fd is the file descriptor of the file which is to be renamed
newName is the new name to be associated with the file

12.9 findFile

findFile() returns the file descriptor associated with a file name

```
fd = findFile( fileName )
```

fd is the file descriptor associated with the file
fileName is the name of the file for which the file descriptor is requested

12.10 deleteFile

deleteFile() deletes the file associated with a specific file descriptor.

```
etherStore.deleteFile( fd )
fd the file descriptor associated with the file to be deleted.
```

12.11 rxSetup

rxSetup sets the receiver port and attenuation which will be used for a recording

```
etherStore.rxSetup( port , attenuation )
```

port is the receiver port to be used for the recording. Possible values are:
0: Monitor In
1: Tx/Rx

attenuation is the receiver frontend attenuation to be used for the recording. Valid values are in the range 0 to 31.5dB in steps of 0.5dB.

12.12 setAGC

setAGC() enabled or disables the automatic gain control algorithm

```
etherStore.setAGC( on_off )
on_off is a Boolean which either enables or disables the automatic gain control
algorithm
```

12.13 getAtten

getAtten() returns the receiver frontend attenuation which is currently in use. This may have been set either by the automatic gain control algorithm or by calling **rxSetup()**.

atten = etherStore.getAtten()

atten the current receiver frontend attenuation in dB. Permissible values are 0 to 31.5dB in steps of 0.5dB.

12.14 txSetup

txSetup()set the transmit output power which will correspond to full scale deflection of the DAC. The DAC has a range of +/- 32767.

etherStore.txSetup(fullScalePower)

fullScalePower is output power of the transmitter corresponding to full scale deflection of the DAC. The maximum output power which can be specified is +5dBm.

12.15 recordLevel

recordLevel() returns the full scale transmitter output power necessary to replay a recorded file at the same level as it was received.

level = etherStore.recordLevel(fd)

level the full scale output power necessary to replay the file at the same level as which it was received. Note: this may be greater than the maximum transmit power of +5dBm.
fd the file descriptor associated with the file for which the full-scale output power is required.

12.16 record

record() instructs the Etherstore to record data into the file associated with a specified file descriptor. Recording will cease once the file is full.

```
etherStore.record( fd )
fd is the file descriptor associated with the file into which the recorded data
will be placed.
```

12.17 waitRecord

waitRecord() waits for the current recording to complete or for a timeout to expire.

```
etherStore.waitRecord( timeout )
```

timeout the maximum time to wait for the recording to complete in units of seconds.

12.18 playback

playback() instructs the Etherstore of playback a portion of the file associated with a specific file descriptor.

```
etherStore.playback( fd , start , stop )
fd is the file descriptor associated with the file to be played back.
start is the time offset from the start of the file at which playback should
commence in units of seconds.
Stop is the time offset from the start of the file which playback should cease
in units of seconds.
```

12.19 waitPlayback

waitPlayback() waits for the current playback to complete or for a timeout to expire.

```
etherStore.waitPlayback( timeout )
```

timeout the maximum time to wait for the playback to complete in units of seconds.

12.20 playbackForever

playbackForever() instructs the Etherstore to playback the file associated with a specific file descriptor in a continuous loop.

```
etherStore.playbackForever( fd )
```

`fd the file descriptor associated with the file to be played back`

12.21 playbackCancel

playbackCancel()stops the current playback.

etherStore.cancelPlayback()

12.22 exitApp

exitApp() will cause the Sapphire application to exit and control return to the *Moreph30* supervistor program,

res = etherStore.exitApp()

res will be set to False if the command fails.

12.23 hardwareReset

hardwareReset() will cause the Etherstore to reboot.

etherstore.hardwareReset()

12.24 powerDown

powerDown() will power down the Etherstore.

etherStore.powerDown()

12.25 getFriendlyName

getFriendlyName() will return the friendly name of the Etherstore unit.

name = etherStore.getFriendlyName()

name is a string containing the friendly name of the Etherstore unit.

12.26 getSerialNumber

geSerialNumber() will return the serial number of the Etherstore unit.

sn = etherStore.getSerialNumber()

sn is an integer containing the serial number of the Etherstore unit.