

EtherStore User Manual

V3.00

19th November 2025

Contents

1. Introduction 4
2 GUI: Basic 5
3 GUI: Advanced 7
4 Create new recording 8
5. Playback 8
6 Recording 9
7. Upload file 9
8. Download file 10
9. Transfer files between EtherStores 10
10. File formats 10
11 C API 12
Overview 12
API server 12
Error handling 12
12 Python API 13
Overview 13
API server 13
Reseting moreph hardware 13
13 Common API operations 14
13.1 Connecting to an EtherStore 14
13.2 Making a recording 15
13.3 Playing back a recording 17
14 Example code 18
15 Library reference 24
15.1 listFiles 24
15.2 filename 24
15.3 fileDuration 25
15.4 fileRxAttenuation 25
15.5 fileRxPort 25
15.6 fileFormat 26
15.7 createFile 26
15.8 renameFile 27
15.9 findFile 27
15.10 deleteFile 27
15.11 rxSetup 28

15.12 setAGC 28
15.13 setRxPort 28
15.14 getAtten 29
15.15 txSetup 29
15.16 recordLevel 30
15.17 record 30
15.18 waitRecord 30
15.19 playback 31
15.20 waitPlayback 31
15.21 playbackForever 31
15.22 playbackCancel 32
15.23 download 32
15.24 upload 32
15.25 waitTransfer 33
15.26 cancelTransfer 33
15.27 show_gui 33
15.28 hide_gui 33
15.29 search 34
15.30 connect 34
15.31 connectSerial 34
15.32 disconnect 35
15.33 hardwareReset 35
15.34 powerDown 35
15.35 getFriendlyName 35
15.36 getSerialNumber 36
15.37 version 36

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.

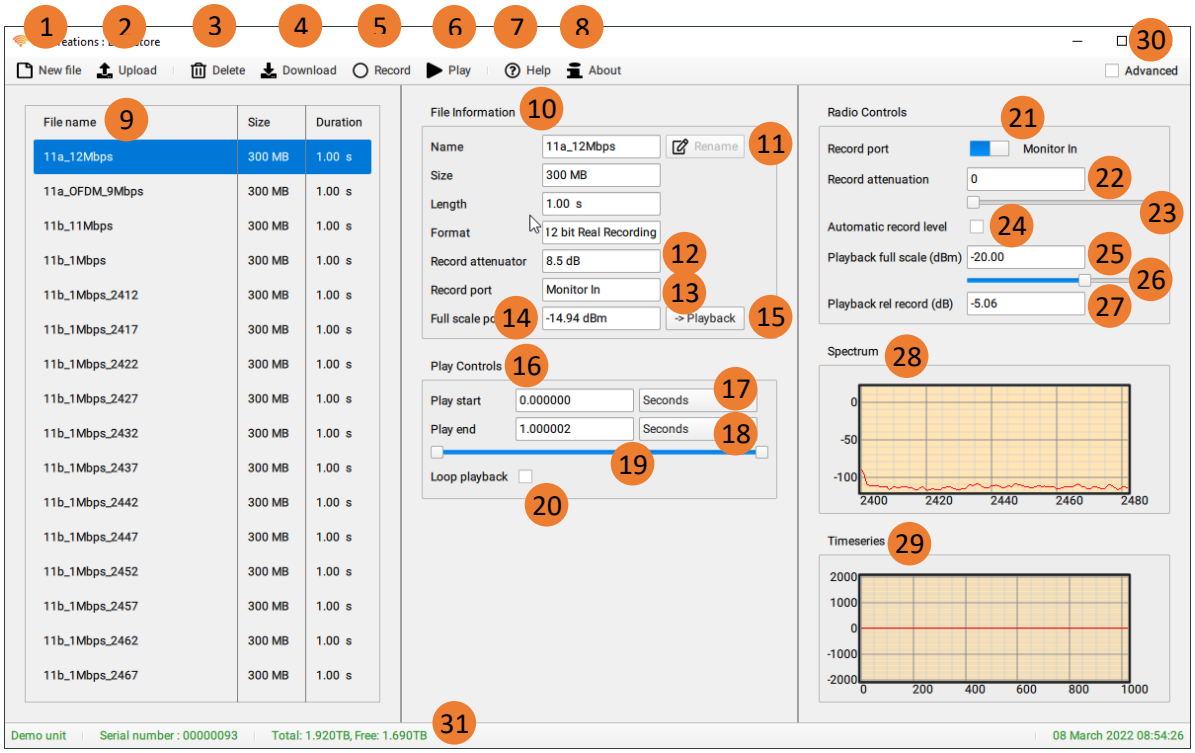
For information about the user interface, see GUI: Basic.

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.



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.

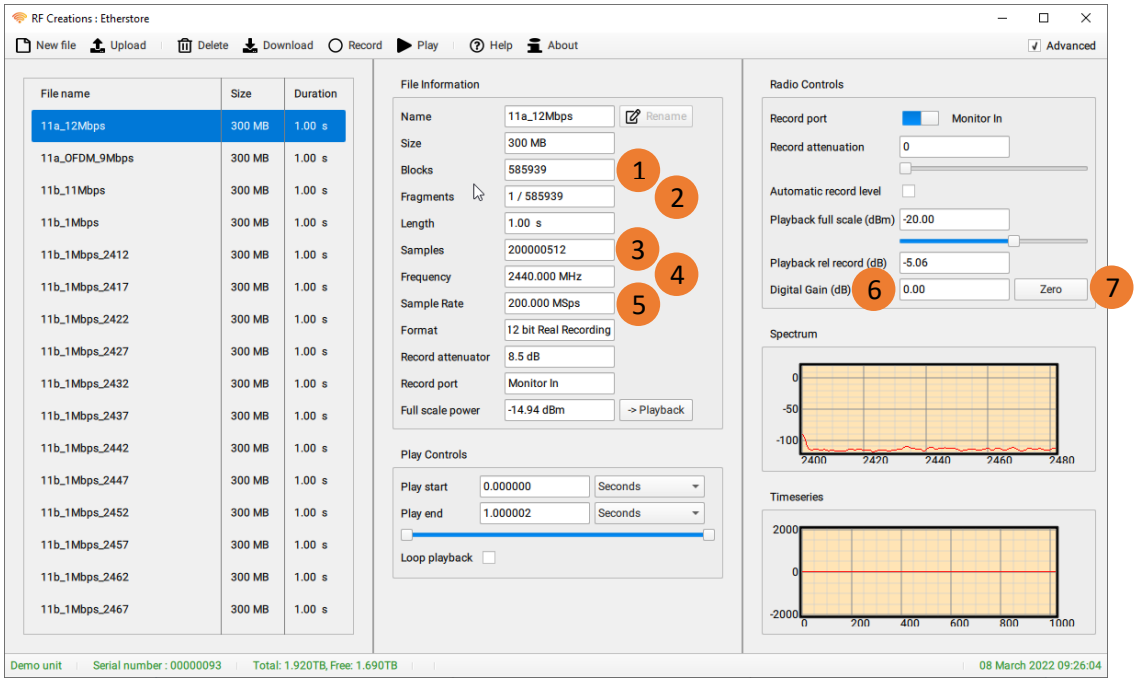
WARNING	This will overwrite the file. If the file is not empty, then a warning dialog will be displayed.

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 in order 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

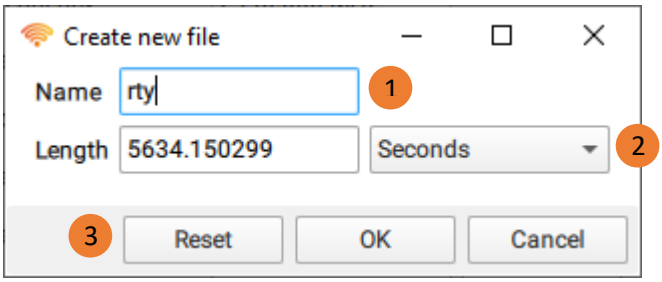
If you enable the advanced checkbox in the upper right of the GUI, you will get some extra information.



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.



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. 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.

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 also 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

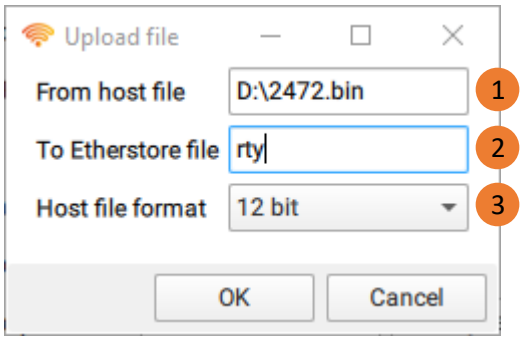
The file is interpreted as a plain binary file with all samples packed. A complex waveform will be centred at 2440MHz on playback, whereas a real waveform is effectively single sided with 0Hz at 2390MHz. Real waveforms are played at 200MSPS, complex at 100MSPS.

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 below 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.



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.

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

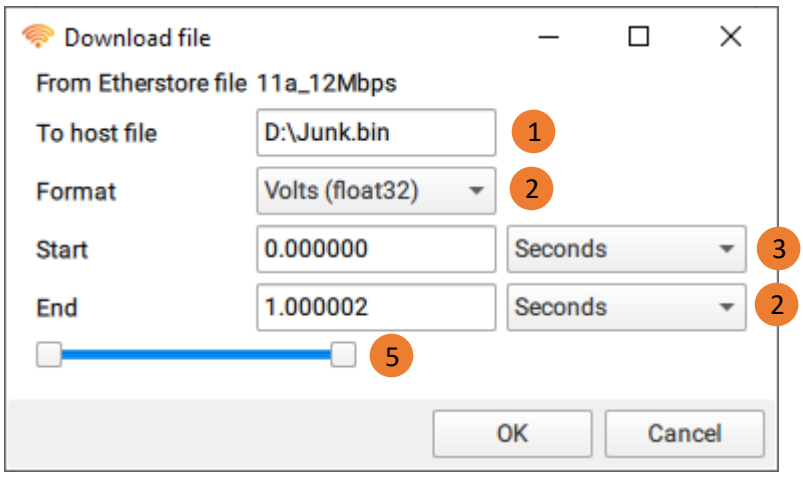
```
from numpy import int16, pi, sin, arange
f = 40
x = (32767*sin(2*pi*f/200*arange(1e8))).astype(int16)
x.tofile("sin16.bin")
```

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 dialof to choose the format and length to download.



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 fairly 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.

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 C API

Overview

The C API provides access to most features of the UI.

The API is reliant on an API server. It is the user's responsibility to start and terminate the API server appropriately.

The C API permits the UI to be either visible or hidden.

The header file EtherAPI.h must be included in any user code The executable must have access to the dynamic libraries contained in the install directory. Note that EtherStore.rfcplugin is a dynamic library.

API server

The API is reliant on the API server running. The API server must be started prior to any API calls.

The API server executable is named EtherAPI_server.exe and resides in the install directory, which by default is C:/Program Files/RFcreations/Phy.

The required commands for launching the API server are may be OS dependent.

The API server will be terminated by any call to `Ether_disconnect()`. If the user wishes to make further calls to the API after calling `Ether_disconnect()`, then it is essential that they restart the API server first.

Error handling

Most API calls return a `Ether_error_t` which are listed in EtherAPI.h.

If the call returns `ETHER_ERROR_UNKNOWN` then further information on the error may be contained in the character string returned by `Ether_error_msg()`.

12 Python API

Overview

Function calls in the Python API are identical to those for the C API with the following exceptions. Enums are replaced by character strings. For example, the C enum ETHER_MonitorIn simply becomes the string “ETHER_MonitorIn”.

The python library is held in EtherAPI.py in the install directory.

An example Python program named EtherStore_example.py is provided in the installation directory.

API server

The API server can be started from a Python script as follows:

```
import subprocess

# Start the API server
# --------------------

subprocess.Popen(["C:/Program Files/RFcreations/Phy/EtherAPI_server"])
```

The API server will be closed by any call to `Ether_disconnect()`, and must be restarted before any further API calls can be made.

Reseting moreph hardware

An example Python program for resetting the moreph hardware can be found in EtherStore_example_reset.py.

13 Common API operations

13.1 Connecting to an EtherStore

The EtherStore devices connected to the host computer can be found using `Ether_search()`. These devices can then be scanned to find the one of interest, for example, by looking for a specific serial number. The selected device can then be connected to using `Ether_connect()`. The code snippet below simply connects to the first EtherStore device found.

```
# Connect to an EtherStore
# ------------------------

print( "Search for EtherStores" )

morephs = []
while (not morephs):
    morephs = Ether_search()
    time.sleep(1)

print( "EtherStore found" , morephs[0] )

Ether_connect(0)

print( "Connected to EtherStore" )
```

Alternatively, it is possible to connect directly to an EtherStore with a given serial number:

```
Ether_connect_serial(uint32_t serial_number);
```

If it is wished to display the GUI, then call Ether_show_gui():

```
Ether_show_gui();
```

Once all operations have been completed, the EtherStore can be disconnected and the API server stopped by calling Ether_disconnect():

```
Ether_disconnect();
```

13.2 Making a recording

To make a recording it is first necessary to create a file in which to place the recorded data. The length of the required recording is specified when creating the file:

```
# Create a new file of duration 10s
# ---------------------------------

# A file descriptor to the file is returned
print( "Create new file" )
Ether_createFile( 10 , "New file" )
```

Having created the new file, it is necessary to obtain a file descriptor to the file for subsequent operations:

```
# Get the file descriptor for the new file
print( "Get file descriptor" )
fd = Ether_findFile( "New file" )
print( "File descriptor of new file is " , fd )
```

Most recordings will be made off-air, so the receiver input port must be specified as MonitorIn. The initial value of the receiver front-end attenuation can be set to zero at the same time:

```
# Set the receiver to Monitor In port and 0 attenuation
# -----------------------------------------------------

print( "Set receiver port to Monitor In" )
Ether_rxSetup( "ETHER_MonitorIn" , 0 );
```

Prior to the recording it is necessary to set the record level, ie receiver front-end attenuation. The attenuation must be sufficient large that loud signals do not saturate the recording whilst being low enough to permit weak signals to be recorded. The recommended way of doing this is by enabling the automatic level control and reading back the attenuation settings it suggests. The maximum of these suggestions is then used.

```
# Turn on AGC to find received signal level
# -----------------------------------------

print( "Turn on AGC" )
Ether_setAGC( True )

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

# Set the receiver to Monitor In port and appropriate attenuation
# ---------------------------------------------------------------

# Turn off the AGC
print( "Turning off AGC" )
Ether_setAGC( False )

print( "Setting up receiver for Monitor In and attenuation " , rxAtten )
Ether_rxSetup( "ETHER_MonitorIn" , rxAtten );
```

Having set all the record parameters, the recording can now be made:

```
# Record into the new file
# ------------------------

print( "Start recording" )
Ether_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" )
Ether_waitRecord( 60 )
print( "Recording ended" )
```

If it is wished to save then recording to the host computer then this can be done by:

```
# Download the file to the host
# -----------------------------

print( "Downloading file" )
Ether_download( fd, "C:/Users/TimNewton/Downloads/junk.bin" , 0 , 10 );

# Wait for download
print( "Wait for download" )
Ether_waitTransfer( 2 * 60 );
```

Either the full file or just part of the file can be downloaded.

13.3 Playing back a recording

If the file to be played back is on the host, then it can be downloaded to the EtherStore:

```
# Upload the file to the moreph
# -----------------------------

print( "Uploading file" )
Ether_upload( "C:/Users/TimNewton/Downloads/junk.bin" , "New file" );

# Wait for upload
print( "Wait for upload" )
Ether_waitTransfer( 4 * 60 );
```

It is now necessary to find the file descriptor for the file to be played back:

```
# Get the file descriptor
print( "Get file descriptor" )
fd = Ether_findFile( "New file" )
print( "File descriptor of new file is " , fd )
```

The level at which the file will be played back must be set. If it is being used to generate an interference environment, it would normally be played back at maximum volume, which is +5dBm for DAC fullscale deflection.

```
# Setup the transmitter to playback at maximum possible power
print( "Set transmitter for maximum power" )
Ether_txSetup( 5 )
```

The duration of the file to be played back can be determined:

```
duration = Ether_fileDuration(fd)
```

The file can now be played back. Either the entire file or part of the file can be played back.

```
# Playback part of the recording starting at 2s and for a duration of 4s
print( "Playback entire file" )
Ether_playback( fd , 0 , duration )

# Wait for playback to end with a timeout
print( "Wait for playback" )
Ether_waitPlayback( duration + 2 )
print( "Playback ended" )
```

14 Example code

```
import subprocess
import sys
import time

# Import the API
# --------------

sys.path.insert(0,"C:/Program Files/RFcreations/Phy")
from EtherAPI import *

# Routine for searching for specific files in EtherStore directory
# ----------------------------------------------------------------

def listFiles():
    fds = Ether_listFiles()
    found = False
    for fd in fds:
        fileName = Ether_filename(fd)
        if( (fileName == "New file") | (fileName == "Renamed file") ):
            found = True
            print( fileName )
            print( "    Duration : " , Ether_fileDuration(fd) , "s" )
            print( "    Rx Atten : " , Ether_fileRxAttenuation(fd) , "dB" )
            print( "    Rx Port  : " , Ether_fileRxPort(fd) )
            print( "    Format   : " , Ether_fileFormat(fd) )
    if( not found ):
        print( "    Files not found" )


# Start the API server
# --------------------

print( "Start EtherAPI server" )

subprocess.Popen(["C:/Program Files/RFcreations/Phy/EtherAPI_server"])
```

```
# Connect to a Moreph
# -------------------

print( "Search for Morephs" )

morephs = []
while (not morephs):
    morephs = Ether_search()
    time.sleep(1)

print( "Moreph found" , morephs[0] )

Ether_connect(0)

print( "Connected to Moreph" )

# Make UI visible
# ---------------

Ether_show_gui()
print( "UI launched" )

# 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" )
Ether_createFile( 10 , "New file" )
listFiles()

# Get the file descriptor for the new file
print( "Get file descriptor" )
fd = Ether_findFile( "New file" )
print( "File descriptor of new file is " , fd )
```

```
# Rename the new file
# -------------------

print( "Rename new file" )
Ether_renameFile( fd , "Renamed file" )
listFiles()

# Find the file descriptor for a named file
fd = Ether_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" )
Ether_rxSetup( "ETHER_MonitorIn" , 0 );

# Turn on AGC to find received signal level
# -----------------------------------------

print( "Turn on AGC" )
Ether_setAGC( True )

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

# Set the receiver to Monitor In port and appropriate attenuation
# ---------------------------------------------------------------

# Turn off the AGC
print( "Turning off AGC" )
Ether_setAGC( False )
```

```
print( "Setting up receiver for Monitor In and attenuation " , rxAtten )
Ether_rxSetup( "ETHER_MonitorIn" , rxAtten );

# Record into the new file
# ------------------------

print( "Start recording" )
Ether_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" )
Ether_waitRecord( 60 )
print( "Recording ended" )

# Playback file at maximum volume
# -------------------------------

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

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

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

# Playback at same level as received
# ----------------------------------

# 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 = Ether_recordLevel( fd )
if( level > 5 ):
    level = 5
print( "Setting transmit level to " , level )
Ether_txSetup( level )
```

```
# Playback the recording, repeating forever
print( "Playback with repeat forever" )
Ether_playbackForever( fd )
time.sleep(15)

# Cancel playback
print( "Cancelling playback" )
Ether_cancelPlayback()

# Download the file to the host
# -----------------------------

print( "Downloading file" )
Ether_download( fd, "C:/Users/TimNewton/Downloads/junk.bin" , 0 , 10 );

# Wait for download
print( "Wait for download" )
Ether_waitTransfer( 2 * 60 );

# Upload the file to the moreph
# -----------------------------

print( "Uploading file" )
Ether_upload( "C:/Users/TimNewton/Downloads/junk.bin" , "New file" );

# Wait for upload
print( "Wait for upload" )
Ether_waitTransfer( 4 * 60 );
listFiles()

# Delete all created files
# ------------------------

# Delete the file
print( "Delete file" )
Ether_deleteFile( fd )
listFiles()

# Delete the uploaded file
```

```
print( "Delete uploaded file" )
fd = Ether_findFile( "New file" )
Ether_deleteFile( fd )
listFiles()

# Close the application and API server
# ------------------------------------

# Close the application
Ether_disconnect()
```

15 Library reference

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

15.1 listFiles

Ether_listFiles() will return an list of the file descriptors for the files held on the EtherStore,

Python:

```
fds = Ether_listFiles()
```

fds is a list of file descriptors.

C dll:

```
Ether_error_t Ether_listFiles(uint64_t **fds, uint32_t *n);
```

fds is a pointer to the list of file descriptors
n will contain the number of file descriptors

15.2 filename

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

Python:

```
name = Ether_fileName(fd)
```

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

C dll:

```
Ether_error_t Ether_filename(uint64_t fd, char *filename, uint32_t nMax);
```

fd is a file descriptor
filename will contain the name of the file
nMax is the maximum number of characters which can be copied into **filename**

15.3 fileDuration

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

Python:

```
duration = Ether_fileDuration(fd)
```

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

C dll:

```
Ether_error_t Ether_fileDuration(uint64_t fd, double *duration);
```

fd is a file descriptor
duration will contain the duration of the data within the file in units of seconds

15.4 fileRxAttenuation

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

Python:

```
attenuation = Ether_fileRxAttenuation(fd)
```

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

C dll:

```
Ether_error_t Ether_fileRxAttenuation(uint64_t fd, double *atten);
```

fd is a file descriptor
atten will contain the receiver frontend attenuation in units of dB used during the recording of the file

15.5 fileRxPort

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

Python:

```
port = Ether_fileRxPort(fd)
```

port is the receiver port used during the recording of the file associated with file descriptor **fd**. Possible values are “ETHER_MonitorIn” and “ETHER_TxRx”

C dll:

```
ETHERSTORE_API Ether_error_t Ether_fileRxPort(uint64_t fd, uint32_t *port);
```

fd is a file descriptor
port will contain the receiver port used during the recording of the file as port_t enum.

15.6 fileFormat

Ether_fileFormat() returns the format of the file

Python:

```
format = Ether_fileFormat(fd)
```

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

"ETHER_12bit_real_data"
"ETHER_16bit_real_data"
"ETHER_12bit_real_recording"
"ETHER_16bit_complex_data"

C dll:

Ether_error_t Ether_fileFormat(uint64_t fd, uint32_t *fileFormat);

fd is a file descriptor
fileFormat will contain the receiver port used during the recording of the file as file_format_t enum.

15.7 createFile

Ether_createFile() creates a new file on the EtherStore into which data can be recorded.

Python:

```
Ether_createFile( duration , name )
```

duration is the length of the file to create in seconds
name is the name of the file to create

C dll:

```
Ether_error_t Ether_createFile(double duration, const char* name);
```

duration is the length of the file to create in seconds
name is the name of the file to create

15.8 renameFile

Ether_renameFile() changes the name of a file held on the EtherStore.

Python:

```
Ether_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

C dll:

```
ETHERSTORE_API Ether_error_t Ether_renameFile(uint64_t fd, const char* 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

15.9 findFile

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

Python:

```
fd = Ether_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

C dll:

```
Ether_error_t Ether_findFile(uint64_t *fd, const char* fileName);
```

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

15.10 deleteFile

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

Python:

```
Ether_deleteFile( fd )
```

fd the file descriptor associated with the file to be deleted.

C dll:

```
Ether_error_t Ether_deleteFile(uint64_t fd);
```

fd the file descriptor associated with the file to be deleted.

15.11 rxSetup

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

Python:

```
Ether_rxSetup( port , attenuation )
```

port is the receiver port to be used for the recording. Possible values are “ETHER_MonitorIn” or “ETHER_TxRx”.
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.

C dll:

```
Ether_error_t Ether_rxSetup(uint32_t port, double attenuation);
```

port is the receiver port to be used which is a port_t enum.
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.

15.12 setAGC

Ether_setAGC() enabled or disables the automatic gain control algorithm

Python:

```
Ether_setAGC( on_off )
```

on_off is a Boolean which either enables or disables the automatic gain control algorithm

C dll:

```
Ether_error_t Ether_setAGC(bool on_off);
```

on_off is a Boolean which either enables or disables the automatic gain control algorithm

15.13 setRxPort

Ether_setRxPort() set the receiver port to be used for recording

Python:

```
Ether_setRxPort(  )
```

port is the receiver port to be used for the recording. Possible values are “ETHER_MonitorIn” or “ETHER_TxRx”.

C dll:

```
Ether_error_t Ether_setRxPort(port_t port);
```

port is the receiver port to be used which is a port_t enum.

15.14 getAtten

Ether_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()**.

Python:

```
atten = Ether_getAtten()
```

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

C dll:

```
Ether_error_t Ether_getAtten(double *atten);
```

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

15.15 txSetup

Ether_txSetup()set the transmit output power which will correspond to full scale deflection of the DAC.

Python:

```
Ether_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.

C dll:

```
Ether_error_t Ether_txSetup(double fullScalePwr);
```

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.

15.16 recordLevel

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

Python:

level = Ether_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

C dll:

```
Ether_error_t Ether_recordLevel(uint64_t fd, double *level);
```

level will be set to 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

15.17 record

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

Python:

```
Ether_record( fd )
```

fd is the file descriptor associated with the file into which the recorded data will be placed.

C dll:

```
Ether_error_t Ether_record(uint64_t fd);
```

fd is the file descriptor associated with the file into which the recorded data will be placed.

15.18 waitRecord

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

Python:

```
Ether_waitRecord( timeout )
```

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

C dll:

```
Ether_error_t Ether_waitRecord(uint32_t timeout);
```

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

15.19 playback

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

Python:

```
Ether_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.

C dll:

```
Ether_error_t Ether_playback(uint64_t fd, double start, double 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.

15.20 waitPlayback

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

Python:

```
Ether_waitPlayback( timeout )
```

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

C dll:

```
Ether_error_t Ether_waitPlayback(uint32_t timeout);
```

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

15.21 playbackForever

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

Python:

```
Ether_playbackForever( fd )
```

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

C dll:

```
Ether_error_t Ether_playbackForever(uint64_t fd);
```

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

15.22 playbackCancel

Ether_playbackCancel()stops the current playback.

Python:

Ether_cancelPlayback()

C dll:

```
Ether_error_t Ether_cancelPlayback(void);
```

15.23 download

Ether_download() downloads a file from the EtherStore to the host.

Python:

```
Ether_download(fd, fileName, start, stop)
```

fd the file descriptor associated with the file to be downloaded to the host
fileName is the name of the file on the host into which the EtherStore file will be copied
start is the first sample of the file to be downloaded, in units of seconds
stop is the last sample of the file to be downloaded, in units of seconds

C dll:

```
Ether_error_t Ether_download(uint64_t fd, const char *fileName, double start,
double stop);
```

fd the file descriptor associated with the file to be downloaded to the host
fileName is the name of the file on the host into which the EtherStore file will be copied
start is the first sample of the file to be downloaded, in units of seconds
stop is the last sample of the file to be downloaded, in units of seconds

15.24 upload

Ether_upload() uploads a file from the host to the EtherStore.

Python:

```
Ether_upload(fileFrom, fileTo)
```

fileFrom is the name of the file on the host from which the file will be copied
fileTo is the name of the file on the EtherStore into which the host file will be copied. This fille will be created.

C dll:

```
Ether_error_t Ether_upload(const char *fileFrom, const char *fileTo);
```

fileFrom is the name of the file on the host from which the file will be copied
fileTo is the name of the file on the EtherStore into which the host file will be copied. This fille will be created.

15.25 waitTransfer

Ether_waitTransfer() waits for a file to be transferred either from the host to the EtherStore or from the EtherStore to the host.

Python:

```
Ether_waitTransfer( timeout )
```

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

C dll:

```
Ether_error_t Ether_waitTransfer( uint32_t timeout );
```

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

15.26 cancelTransfer

Ether_cancelTransfer() cancels any ongoing file transfer either from the host to the EtherStore or from the EtherStore to the host.

Python:

```
Ether_cancelTransfer()
```

C dll:

```
Ether_error_t Ether_cancelTransfer( void );
```

15.27 show_gui

Ether_show_gui() causes the user interface to become visible

Python:

```
Ether_show_gui()
```

C dll:

```
Ether_error_t Ether_show_gui( void );
```

15.28 hide_gui

Ether_hide_gui() causes the user interface to become invisible

Python:

```
Ether_hide_gui()
```

C dll:

```
Ether_error_t Ether_hide_gui( void );
```

15.29 search

Ether_search() returns a list of the EtherStore devices which are currently available. This includes devices visible on both USB and Ethernet.

Python:

```
morephs = Ether_search()
```

morephs is a list of handles to the EtherStore devices.

C dll:

```
Ether_error_t Ether_search(Ether_search_result_t** results, int* N);
```

Results is a pointer to a list of handles to the EtherStore devices
N is the number of EtherStore devices, and hence handles, which have been discovered

15.30 connect

Ether_connect() connects to a EtherStore which has been discovered using **Ether_search()**.

Python:

```
Ether_connect(index)
```

index is the zero based index of the list of EtherStores returned by **Ether_search()**.

C dll:

```
Ether_error_ t Ether_connect(Ether_handle_t* handle);
```

handle is one of the handles returned by **Ether_search()**.

15.31 connectSerial

Ether_connectSerial() connects to a EtherStore with a specified serial number.

Python:

```
Ether_connectSerial(serial_number)
```

serial_number is the serial number of the EtherStore to connect to.

C dll:

```
Ether_error_t Ether_connect_serial(uint32_t serial_number);
```

serial_number is the serial number of the EtherStore to connect to.

15.32 disconnect

Ether_disconnect() disconnects from the currently connected EtherStore.

Python:

```
Ether_disconnect()
```

C dll:

```
Ether_error_t Ether_disconnect(void);
```

15.33 hardwareReset

Ether_hardwareReset() will cause the Etherstore to reboot.

Python:

```
Ether_hardwareReset(handle)
```

handle is a handle to a moreph unit returned by Ether_search().

C dll:

```
Ether_error_t Ether_hardwareReset(Ether_handle_t* handle);
```

handle is a handle to a moreph unit returned by Ether_search().

15.34 powerDown

Ether_powerDown() will power down the EtherStore.

Python:

```
Ether_powerDown()
```

C dll:

```
Ether_error_t Ether_powerDown(void);
```

15.35 getFriendlyName

Ether_getFriendlyName() will return the friendly name of the EtherStore unit.

Python:

```
friendlyName = Ether_getFriendlyName()
```

friendlyName is a string containing the friendly name of the EtherStore unit.

C dll:

```
Ether_error_t Ether_getFriendlyName(char *friendlyName, uint32_t nMax);
```

friendlyName will receive the friendly name of the EtherStore unit.
nMax is the maximum number of characters which can be written into **friendlyName**.

15.36 getSerialNumber

Ether_geSerialNumber() will return the serial number of the EtherStore unit.

Python:

```
sn = Ether_getSerialNumber()
```

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

C dll:

```
Ether_error_t Ether_getSerialNumber(uint32_t *sn);
```

sn will receive the serial number of the EtherStore unit.

15.37 version

Ether_version() will return a string containing the version information of the API.

Python:

```
version = Ether_version()
```

version is an string containing the version information of the API.

C dll:

```
Ether_error_t Ether_version(char *version, uint32_t n);
```

version will receive the version information of the API.
n is the maximum number of characters which can be written into **version**.