Cambrionix
CHARGE • CONNECT • MANAGE

[API]

Cambrionix API

SOFTWARE USER GUIDE

[Charge] [Connect] [Manage]

Contents

Cambrionix API 1
Introduction 1
Prerequisites 2
OS X installation 3
Windows installation 3
Linux installation 4
Quick start 7
Environment 8
Description of API calls 9
Version 10
Discovery 11
Connection 13
Local or Remote 26
U8S and U8S-EXT 26
Docks 26
Dictionaries 27
Socket connections 34
Timeouts 34
Controlling the LEDs 34
Limitations 35
Remote access 35
Troubleshooting 36
Error codes 37
Revision history 38

Cambrionix API

Introduction

Cambrionix API can be used to control Cambrionix PowerSync and Universal charging units that use the Cambrionix Very Intelligent Charging Protocol.

The Cambrionix API resides in a locally installed daemon called cbrxd. This provides a programming language independent JSON-RPC interface to control Cambrionix units connected to the local machine. The Cambrionix API can also connect with Cambrionix units that are connected to the network eg EtherSync.

There is also a companion daemon, it has different names depending on the OS in use. This companion daemon's job is to monitor the system for USB insertion or removal events which are then notified to cbrxd in order to rescan the USB tree. This is done so that the API will reflect the changed status information in a timely manner.

A simple Python wrapper is provided with a public domain JSON-RPC library that will allow scripts to be written without needing to be overly familiar with JSON. Alternatively you may use the programming language of your choice to connect directly to the daemon over a standard TCP/IP socket and send and receive JSON formatted data.

When the API is used to communicate with a remote network attached Cambrionix unit this is done over an ssh tunnel.

The Cambrionix API supports multiple simultaneous client connections to itself and supports simultaneous access to multiple Cambrionix units.

Prerequisites

Before you can use the Cambrionix API, there are a few steps and checks that need to be completed.

Direct access to USB hardware

For the API to be able to retrieve USB information from connected devices, it must have direct access to the hardware. This means that in general running in a Virtual Machine (VM) such as Parallels or VirtualBox is not supported as the virtualisation prevents the API from being able to determine which USB device is connected to which physical port.

Sync capable charger for USB information

In order for the API to be able to return USB device information such as the Manufacturer or the serial number, there must be a USB connection from the host machine to the connected device. This is only present on sync capable chargers. Charge only chargers have a USB connection to the charger itself but not to connected devices. The Cambrionix API is still functional with charge only chargers but will be unable to return the USB device information.

Firmware version for universal chargers (U8, U16, PP15, PP8 etc.)

Cambrionix Universal Chargers, when used with this API, need to have firmware version 1.52 or later installed and we recommend that the latest version available on our website is installed.

FTDI VCP drivers

The Cambrionix API daemon (cbrxd) needs to be able to communicate with the local Cambrionix unit. Each Cambrionix unit contains an FTDI USB to UART converter that will make them appear to the local operating system as a serial port. The operating system will need to have the appropriate VCP (Virtual COM Port) driver installed. For Linux the default support in the kernel is sufficient.

Do not install the D2XX drivers on Linux or macOS as this conflicts with the required VCP drivers. On Windows only, the D2XX support can coexist with the VCP support.

JSON-RPC library

The API uses JSON-RPC over TCP. Any programming language that has support for JSON-RPC can be used, libraries are widely available for other languages.

Setting up cbrxd

The API is implemented in a daemon process called cbrxd. This needs to be running and contactable to be able to manage the Cambrionix units. The transport used is TCP with a default TCP port of 43424.
If needed, the listening port can be changed:

- Either by calling cbrxd with the option –port=XXXX where XXXX is an alternate port number between 1-65535.
- Or, by changing the value in the configuration file /usr/local/share/cbrxd/config/listeningport

There is also a companion daemon that monitors the system for USB insertion and removal events and notifies the API to rescan the USB bus. This means that USB information such as VID, PID etc is available in a timely manner.

OS X installation

For OS X an installer is provided that will set up cbrxd to run as a daemon process.

OS X installation -- setting up the JSON-RPC library

The cbrxd package includes a Python JSON-RPC package in /usr/local/share/cbrxd/python, to install this go into an appropriate directory where you can unpack the installer and do:

```
$ mkdir cbrxapi
$ cd cbrxapi
$ tar xvzf /usr/local/share/cbrxd/python/jsonrpc-0.1.tar.gz
$ cd jsonrpc-0.1/
$ sudo python setup.py install
```

Internet access will be needed during the install in order to access and install any additional components that may be required.

Windows installation

For Windows a self extracting installer is provided that will set up cbrxd to run as a Windows service. In order to use the example Python code you will also need to install Python eg https://www.python.org/downloads/

FTDI drivers

The FTDI drivers are required to access Cambrionix chargers that are connected to the host machine. These drivers are included in the installer and can be installed by ticking the tickbox. However installation of these drivers does not complete until a Cambrionix charger is attached to the host machine. If you install the API and the FTDI drivers before the first time you connect a Cambrionix charger then the API will not start and you will need to reboot the host machine after connecting a Cambrionix charger that triggers the completion of the FTDI driver installation, in order to ensure that the API service is correctly started.

Windows installation – setting up the JSON-RPC library

The installer will place a zip file, jsonrpc-0.1.zip into the directory “C:\Program Files (x86)\Cambrionix\Cambrionix API\python”. Please extract the files from this zip file. Once Python is installed you can then, using “Command Prompt”, change directory to where you extracted the files from the zip file and then execute the following command

python setup.py install

This installs the Python wrapper into your system so that you may use it to access the API. Internet access will be needed during the install in order to access and install any additional components that may be required.

Linux installation

This will vary somewhat per distribution, but as a general guideline:

- The package needs to be unpacked to a suitable location, i.e. /usr/local:

  ```
  $ cd /usr/local
  ```

- In the following command substitute the path/download name and the name of the tar.gz file:

  ```
  $ sudo tar xvzf ~/Downloads/cbrxd-0.5.tar.gz
  ```

- The main binary cbrxd needs to be able to find the support libraries, do not separate them from the main binary.
- Main binary will be located in /usr/local/bin
- Documentation will be located in /usr/local/share/cbrxd/doc
- Examples will be located in /usr/local/share/cbrxd/examples
- Python installer for json-rpc will be located in /usr/local/share/cbrxd/python
- Setup scripts will be located in /usr/local/share/cbrxd/setup
  Please use these setup scripts, or the information in them, in order to install the two daemons, cbrxd and cbrxudevmonitor. cbrxd is delivered as a binary but cbrxudevmonitor is delivered as source code that needs to be compiled into a binary. cbrxudevmonitor is an optional component, the API will be functional without it.
- Dependencies, may be different depending on your Linux distribution, the below are for Ubuntu 14.04
  libc6:i386
  libglib2.0-0:i386
  libicu52:i386
  libncurses5:i386
  libstdc++6:i386
  libudev1
  avahi-daemon
  gcc

Linux installation -- setting up the JSON-RPC library

The cbrxd package includes a Python JSON-RPC package in /usr/local/share/cbrxd/python, to install this go into an appropriate directory where you can unpack the installer and do:

```
$ sudo apt-get install python-setuptools
$ mkdir cbrxapi
$ cd cbrxapi
$ tar xvzf /usr/local/share/cbrxd/python/jsonrpc-0.1.tar.gz
$ cd jsonrpc-0.1/
$ sudo python setup.py install
```

Internet access will be needed during the install in order to access and install any additional components that may be required.

Linux installation -- setting up cbrxd to start up automatically

There are many Linux distributions available and there are also different systems used to boot the machine and bring up all the required services. Rather than make an attempt to navigate the complexities of this situation, which may well fail, we give you the tools and examples to make the necessary adjustments to your system in order to have the Cambrionix API start on system boot.

Three example startup scripts are included in /usr/local/share/cbrxd/setup, choose the one appropriate for your configuration:

SysV init

The file needed is cbrxd.sh
Inspect the contents and verify that this does what you need for your local installation.
To install it:

```
$ sudo ./install_sysv.sh
```

systemd

The file needed is cbrxd.service
Inspect the contents and verify that this does what you need for your local installation.
To install it:

```
$ sudo ./install_service.sh
```

Upstart

The file needed is cbrxd.conf
Inspect the contents and verify that this does what you need for your local installation.
To install it:

```
$ sudo ./install_upstart.sh
```

Command line options for cbrxd

`--version`: Return the version of cbrxd and then exit:
Example:

```
% cbrxd --version
version 0.5.0 build 23
```

`--port=XXXXX`: Run cbrxd with an alternate TCP listening port. Specifying the command line option overrides both the default value of 43424 or any value configured in /usr/local/share/cbrxd/config/listening port.

Example:

```
% cbrxd --port=54321
```

Return codes for cbrxd

The following values will be returned on exit by cbrxd:
0 on successful exit
(i.e. on doing `cbrxd -version`)
1 on unsuccessful exit
(i.e. on cbrxd being passed an invalid port number or failed to open the listening port)

Logging

Log messages generated by cbrxd go to syslog.

Quick start

Some example scripts are included in `/usr/local/share/cbrxd/examples`.

Minimal example

Here is a minimal example of using the API, the code is written in Python 2.7.8:

```
1  import sys
   from cbrxapi import cbrxapi
2  result = cbrxapi.cbrx_discover("local")
   if result==False:
       print "No Cambrionix unit found."
       sys.exit(0)

   unitId = result[0]
3  handle = cbrxapi.cbrx_connection_open(unitId)
4  nrOfPorts = cbrxapi.cbrx_connection_get(handle, "nrOfPorts")
5  cbrxapi.cbrx_connection_close(handle)

6  print "The Cambrionix unit " + unitId + " has " + str(nrOfPorts) + "
   ports."
```

A brief explanation:

1. Import the cbrxapi library.
2. Call cbrx_discover with “local” to find any locally attached Cambrionix units. This will return a list of local Cambrionix units. This example always uses the first Cambrionix unit returned.
3. Open a connection to the Cambrionix unit, which will return a handle for the connection.
4. Using the handle, get the property “nrOfPorts” from the Cambrionix unit.
5. Done using the Cambrionix unit, close the handle.
6. Finally, print out the information retrieved from the Cambrionix unit.

Error handling

Note that the code above doesn’t check for errors so the Python script will stop on any failure. This should be made more robust by catching any exceptions and dealing with them appropriately.

A JSON-RPC error will return an error member containing the following members:
- code (mandatory) – an integer indicating
either a pre-defined JSON-RPC error code in the range -32768 to -32000
or a CBRXAPI error code as documented in the section”CBRXAPI specific errors” section.
- message (optional) – a message string explaining the error code
- data (optional) – extra information about the error like debug messages or handles.

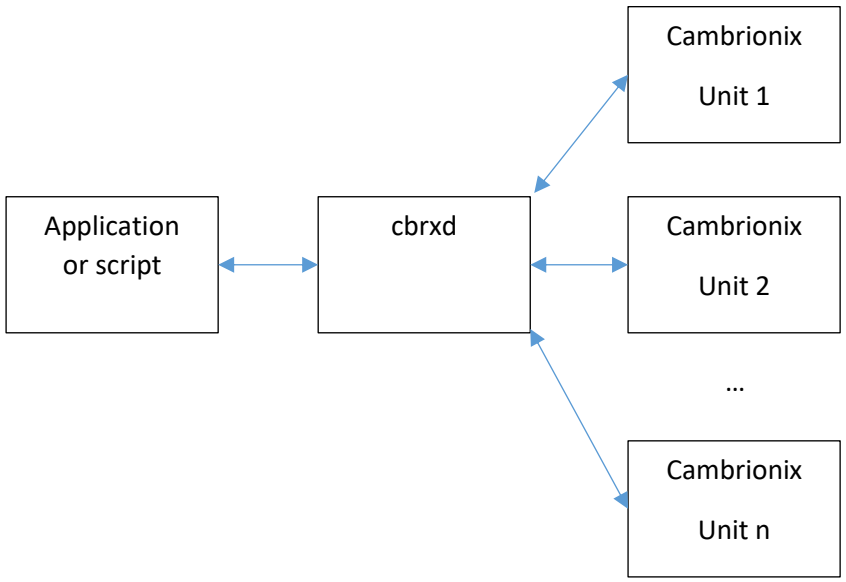
The Python JSON-RPC used causes an exception for an error response with the following mapping:
member code is returned in e.error_code
member message is returned in e.error_message
member data is returned in e.error_data.

In step 3 you could catch an error response with:

```
3  try:
    handle = cbrxapi.cbrx_connection_open(id)
   except jsonrpc.RPCFault as e:
     gotException = True
     errorCode = e.error_code
     errorMessage = e.error_message
     errorData = e.error_data
```

Depending on the errorCode returned different actions can be taken, i.e. the user could be prompted to check whether the device is plugged in before retrying or asked to verify that cbrxd is installed.

Environment



The Cambrionix API is implemented in cbrxd, which sits between the application and the Cambrionix units. It maps the properties of the Cambrionix units into API commands.
For example:

- to disable a USB port
  `cbrx_connection_set (connectionHandle, "Port.2.mode", "o")`
- to reset a Cambrionix unit
  `cbrx_connection_set (connectionHandle, "Reboot", True)`
- to get the number of USB ports of a Cambrionix unit
  `cbrx_connection_get(connectionHandle, "nrOfPorts")`

Description of API calls

The descriptions of the API calls contain a sample Python call and the raw jsonrpc requests / responses as you would see them on the wire.

JSON-RPC requests

The JSON-RPC implementation should hide these details.
The Python request `cbrxapi.cbrx_connection_get(7654, "nrOfPorts")`
translates into a JSONRPC request containing the method name:

```
"method": "cbrx_connection_get",
```

and a JSON representation of the parameters, which is a JSON array of values:

```
"params": [ 7654, "nrOfPorts" ]
```

Two further key-value pairs need to be passed to complete the JSON-request;
One indicating the version of jsonrpc being used, in this case 2.0:

```
"jsonrpc": "2.0"
```

and an id identifying this request:

```
"id": 0
```

The id is mandatory but only relevant if multiple requests can be outstanding simultaneously over the same connection. It helps to match responses to (asynchronous) requests. The response for a request will be given the matching id by cbrxd.

Grouping this all together will give the complete JSON-RPC request:

```
{       "jsonrpc": "2.0",
        "method": "cbrx_connection_get",
        "params": [ 7654,
                    "nrOfPorts" ],
        "id": 0

}
```

There are 3 groups of calls in the API:

- Version
- Discovery
- Connection

Version

cbrx_apiversion

Return the interface version of the local API running.

Input: none

Returns:

On success:
Retuns a pair of integers (major, minor) indicating the API version.

The current version is major 1, minor 7.

On failure: a JSON-error object will be returned.

Example Python call:

```
cbrxapi.cbrx_apiversion()
```

Example JSONRPC request:

```
{       "jsonrpc": "2.0",
        "method": "cbrx_apiversion",
        "id": 0

}
```

Example successful response:

```
{       "jsonrpc":"2.0",
        "id":0,
        "result":[1,7]

}
```

Discovery

cbrx_discover

Discover Cambrionix units.

Input: “local” for Cambrionix units attached to the local machine
“remote” for Cambrionix network attached units eg EtherSync
“docks” for units that contain more than one Cambrionix charger

Returns:

On success: the unit IDs for the discovered Cambrionix units will be returned as an array of strings. Each unit ID is guaranteed to be unique. The unit ID is based on the serial number of the Cambrionix unit.
On failure: a JSON-error object will be returned.

Example Python call:

```
cbrxapi.cbrx_discover("local")
```

Example JSONRPC request:

```
{       "jsonrpc": "2.0",
        "method": "cbrx_discover",
        "params": ["local"],
        "id": 0

}
```

Example successful response:

```
{       "jsonrpc":"2.0",
        "id":0,
        "result":["DB0074F5"]

}
```

Example unsuccessful response:

```
{       "jsonrpc":"2.0",
        "id":0,
        "error":
        {       "code":-32602,
                "message": "Invalid params"
        }

}
```

cbrx_discover_id_to_os_reference

Map a unit ID for a discovered Cambrionix unit to a device name as used by the OS.

Input: a unit ID as returned by cbrx_discover

Returns:

On success: the device name as used by the OS for the connection that the Cambrionix unit identified by the unit ID is connected to
On failure: a JSON-error object will be returned.

Note:

This only makes sense for locally attached Cambrionix units.

Example Python call:

```
cbrxapi.cbrx_discover_id_to_os_reference(unitId)
```

Example JSONRPC request:

```
{     "jsonrpc": "2.0",
      "method": "cbrx_discover_id_to_os_reference",
      "params": ["DB0074F5"],
      "id": 0

}
```

Example successful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "result":["/dev/ttyUSB0"]

}
```

Example unsuccessful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "error":
      {     "code":-32602,
            "message": "Invalid params"
      }

}
```

Connection

cbrx_connection_remote_set_password

Set a password for a remote device to be used when opening a connection to that remote device.

For a remote unit this call must be made before cbrx_connection_open in order to provide the password needed for the ssh connection.

Input parameter:

1. a unit ID as returned by a previous call to cbrx_discover
2. password for the remote unit

Returns:

On success: the boolean value true will be returned

On failure: a JSON-error object will be returned.

Example Python call:

```
result =
      cbrxapi.cbrx_connection_remote_set_password("EtherSyncxxyyzz.lo
      cal.", "passW0rd")
```

Example JSONRPC request:

```
{     "jsonrpc": "2.0",
      "method": " cbrx_connection_remote_set_password",
      "params": ["EtherSyncxxyyzz.local.", "passW0rd"],
      "id": 0

}
```

Example successful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "result":true

}
```

Example unsuccessful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "error":
      {     "code":-10001,
            "message": "ID not found"

}

}
```

cbrx_connection_remote_add_device

Add a remote device that cannot be reached by auto-discovery.

Input: a remote device hostname or IP address

Returns:

On success: the boolean value true will be returned

On failure: a JSON-error object will be returned.

Example Python call:

```
result = cbrxapi.cbrx_connection_remote_add_device("remoteHostName")
```

Example JSONRPC request:

```
{     "jsonrpc": "2.0",
      "method": " cbrx_connection_remote_add_device",
      "params": ["remoteHostName"],
      "id": 0

}
```

Example successful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "result":true

}
```

Example unsuccessful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "error":
      {     "code":-32602,
            "message": "Invalid params"

}

}
```

cbrx_connection_remote_clear_devices

Clear the list of remote devices, currently open devices will be retained.

Input:

Returns:

On success: the boolean value true will be returned

On failure: a JSON-error object will be returned.

Example Python call:

```
result = cbrxapi.cbrx_connection_remote_clear_devices()
```

Example JSONRPC request:

```
{     "jsonrpc": "2.0",
      "method": " cbrx_connection_remote_clear_devices",
      "id": 0

}
```

Example successful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "result":true

}
```

Example unsuccessful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "error":
      {     "code":-32602,
            "message": "Invalid params"

}

}
```

cbrx_connection_open

Open a connection to the Cambrionix unit specified.

A succesful open results in a connection handle that can be used for further calls, which needs to be closed with a call to cbrx_connection_close.
An unsuccessful open does not need a corresponding call to cbrx_connection_close.

Input parameter:

1. a unit ID as returned by a previous call to cbrx_discover
2. The second parameter is optional but will default to “local” when absent.
   For “remote” or “docks”, these values must be specified.

Returns:

On success: a connection handle will be returned as an integer

On failure: a JSON-error object will be returned.

Example Python call:

```
connectionHandle = cbrxapi.cbrx_connection_open("DB0074F5","local")
```

Example JSONRPC request:

```
{     "jsonrpc": "2.0",
      "method": "cbrx_connection_open",
      "params": ["DB0074F5"],
      "id": 0

}
```

Example successful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "result":7654

}
```

Example unsuccessful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "error":
      {     "code":-10001,
            "message": "ID not found"

}

}
```

cbrx_connection_close

Close a connection to a Cambrionix unit previously opened, as specified by the connection handle.

Input parameter:

a connection handle as returned by a previous call to cbrx_connection_open

Returns:

On success: the boolean value true will be returned

On failure: a JSON-error object will be returned.

Note:

It is important to receive the response before closing the socket to ensure the operation has time to be actioned.

Example Python call:

```
result = cbrxapi.cbrx_connection_close(connectionHandle)
```

Example JSONRPC request:

```
{     "jsonrpc": "2.0",
      "method": "cbrx_connection_close",
      "params": [7654],
      "id": 0

}
```

Example successful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "result":true

}
```

Example unsuccessful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "error":
      {     "code":-10005,
            "message": "Invalid handle"

}

}
```

cbrx_connection_remote_apiversion

Return the API version from a remote connection that has previously been opened.

Input parameter:

a connection handle as returned by a previous call to cbrx_connection_open

Returns:

On success:
Retuns a pair of integers (major, minor) indicating the remote API version.

The current version is major 1, minor 5.

On failure: a JSON-error object will be returned.

Example Python call:

```
cbrxapi.cbrx_connection_remote_apiversion(connectionHandle)
```

Example JSONRPC request:

```
{     "jsonrpc": "2.0",
      "method": "cbrx_conection_remote_apiversion",
      "params": [7654],
      "id": 0

}
```

Example successful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "result":[1,5]

}
```

cbrx_connection_getdetail

Retrieve the detailed state output from all ports on the Cambrionix unit specified by connectionHandle. This is useful for debugging issues where a port reports an error.

Input parameter:

a connection handle as returned by a previous call to cbrx_connection_open

Returns:

On success: a string containing the detailed state for every port of the Cambrionix unit

On failure: a JSON-error object will be returned.

Example Python call:

```
cbrxapi.cbrx_connection_getdetail(connectionHandle)
```

Example JSONRPC request:

```
{     "jsonrpc": "2.0",
      "method": "cbrx_connection_getdetail",
      "params": [7654],
      "id": 0

}
```

Example successful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "result":"Port: 1
        vstate
          port_mode:           2
          charge_state:        0
          can_sync:            0
          attached:            0
                   ...
        profile_list_len:      0
        profile:               None
        adet_blanking_timer:   0"
      }
```

Example unsuccessful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "error":
      {     "code":-10005,
            "message": "Invalid handle"
      }

}
```

cbrx_connection_getdictionary

List all tags that can return information on the Cambrionix unit specified by connectionHandle.

Input parameter:

a connection handle as returned by a previous call to cbrx_connection_open

Returns:

On success: an array of strings containing the names of the readable tags for the Cambrionix unit

On failure: a JSON-error object will be returned.

Example Python call:

cbrxapi.cbrx_connection_getdictionary(connectionHandle)

Example JSONRPC request:

```
{     "jsonrpc": "2.0",
      "method": "cbrx_connection_getdictionary",
      "params": [7654],
      "id": 0

}
```

Example successful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "result":[  "SystemTitle",
                  "Hardware",
                  "Firmware",
                  ...
               ]

}
```

Example unsuccessful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "error":
      {     "code":-10005,
            "message": "Invalid handle"
      }

}
```

cbrx_connection_get

From the Cambrionix unit specified by the connection handle, get the value of the tag

Input parameters:

1. connectionHandle as returned by a previous call to cbrx_connection_open
2. tag as returned by a call to cbrx_connection_getdictionary

Returns:

On success: the value of the tag specified

On failure: a JSON-error object will be returned.

Example Python call:

```
value = cbrxapi.cbrx_connection_get(connectionHandle, "nrOfPorts")
```

Example JSONRPC request:

```
{     "jsonrpc": "2.0",
      "method": "cbrx_connection_get",
      "params": [ 7654,
                  "nrOfPorts" ],
      "id": 0

}
```

Example successful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "result":8

}
```

Example unsuccessful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "error":
      {     "code":-10003,
            "message": "Key not found"

}

}
```

cbrx_connection_setdictionary

List all writable value tags and command tags for the Cambrionix unit specified by connectionHandle.

Input parameter:

a connection handle as returned by a previous call to cbrx_connection_open

Returns:

On success: an array of strings containing the names of the writable tags and command tags for the device

On failure: a JSON-error object will be returned.

Example Python call:

```
cbrxapi.cbrx_connection_setdictionary(connectionHandle)
```

Example JSONRPC request:

```
{     "jsonrpc": "2.0",
      "method": "cbrx_connection_setdictionary",
      "params": [7654],
      "id": 0

}
```

Example successful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "result":[  "Port.1.mode",
                  "Port.2.mode",
                  ...
                  "ClearRebootFlag ",
                  "Reboot",
                  ...
               ]

}
```

Example unsuccessful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "error":
      {     "code":-10005,
            "message": "Invalid handle"

}

}
```

cbrx_connection_set

On the Cambrionix unit specified by the connection handle, set the tag to the value specified.

Input parameters:

1. connectionHandle as returned by a previous call to cbrx_connection_open
2. tag as returned by a call to cbrx_connection_setdictionary
3. value, the value to set the tag to.

Returns:

On success: the Boolean value true

On failure: a JSON-error object will be returned.

Note:

It is important to receive the response before closing the socket to ensure the operation has time to be actioned.

Example Python call:

```
cbrxapi.cbrx_connection_set(connectionHandle, "Reboot", True)
```

Example JSONRPC request:

```
{       "jsonrpc": "2.0",
        "method": "cbrx_connection_set",
        "params": [ 7654,
                    "TwelveVoltRail.OverVoltage",
                    true
                  ],
        "id": 0

}
```

Example successful response:

```
{       "jsonrpc":"2.0",
        "id":0,
        "result":true

}
```

Example unsuccessful response:

```
{       "jsonrpc":"2.0",
        "id":0,
        "error":
        {       "code":-10004,
                "message": "Error setting value"
        }

}
```

cbrx_connection_closeandlock

Forcibly close all connections to a Cambrionix unit and lock it against further use until released by cbrx_connection_unlock. Other processes that were using these connections will get errors returned if trying to access this Cambrionix unit.
Locking a Cambrionix unit that was not previously opened does no harm and will succeed.

Input parameter:

a unit ID as returned by a previous call to discover

Returns:

On success: the boolean value true will be returned

On failure: a JSON-error object will be returned.

Note:

It is important to receive the response before closing the socket to ensure the operation has time to be actioned.

Example Python call:

```
cbrxapi.cbrx_connection_closeandlock("DB0074F5")
```

Example JSONRPC request:

```
{     "jsonrpc": "2.0",
      "method": "cbrx_connection_closeandlock",
      "params": ["DB0074F5"],
      "id": 0

}
```

Example successful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "result":true

}
```

Example unsuccessful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "error":
      {     "code":-10001,
            "message": "ID not found"

}

}
```

cbrx_connection_unlock

Unlock a Cambrionix unit that was previously locked.
Unlocking a Cambrionix unit that was not previously locked does no harm and will succeed.

Input parameter:

a unit ID as returned by a previous call to discover

Returns:

On success: the boolean value true will be returned

On failure: a JSON-error object will be returned.

Note:

It is important to receive the response before closing the socket to ensure the operation has time to be actioned.

Example Python call:

```
cbrxapi.cbrx_connection_unlock("DB0074F5")
```

Example JSONRPC request

```
{     "jsonrpc": "2.0",
      "method": "cbrx_connection_unlock",
      "params": ["DB0074F5"],
      "id": 0

}
```

Example successful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "result":true

}
```

Example unsuccessful response:

```
{     "jsonrpc":"2.0",
      "id":0,
      "error":
      {     "code":-10001,
            "message": "ID not found"

}

}
```

Local or Remote

Local and remote Cambrionix units need to be treated slightly differently due to the way they are accessed.

A 'local' Cambrionix unit is one that is connected with a USB cable to the machine which is running the API. If you do not specify the location to cbrx_discover or cbrx_connection_open then the API will assume that you are using a local Cambrionix unit.

A 'remote' Cambrionix unit is not connected to the machine which is running the API but is instead connected to a network that is accessible from the machine which is running the API. Currently the only Cambrionix unit which connects over a network is EtherSync.

In order to connect to a remote Cambrionix unit you must discover it first by passing 'remote' as the location parameter to cbrx_discover. Once you have the device ID of a remote Cambrionix unit, you then need to provide its password using cbrx_connection_remote_set_password. When the password has been set, you can then call the cbrx_connection_open specifying the device ID of the remote unit as well as 'remote' for the location. Once the connection is open the handle is sufficient for the API to distinguish local or remote and so other calls do not need to specify the location.

There is also an additional call cbrx_connection_remote_apiversion which will return the API version of the remote charger.

U8S and U8S-EXT

The U8S and U8S-EXT chargers have a small difference from other Cambrionix Universal Chargers in that their control or update port has an external connection. All other Cambrionix Universal Chargers have a single host connection that combines the functions of the upstream connection to the host with the control or update port.

In order for the U8S or U8S-EXT to work correctly with the API there must be a USB cable that connects the control or update port to an expansion port on the same board. This is in addition to the USB cable that connects the host port to the machine running the API.

Docks

When there are two Cambrionix chargers that have been built into a product, with the second charger connected to an expansion port of the first charger, this is known as a 'Dock'. For some operations it may be convenient to treat these two chargers as a single unit, that combines the ports of both chargers.

If the application wishes to access the dock as a single unit it should firstly call cbrx_discover with the parameter "docks" to obtain the list of docks available. The application should then call cbrx_connection_open with the ID in question and also specifying "docks".

The dock unit will return the combined total for tags such as 'nrOfPorts' and 'TotalCurrent_mA'. The range of ports is expanded to cover the combined total number of ports for the two chargers. For the charger with its host port directly connected to the computer, its ports will be referenced first followed by those of the charger connected to first charger's expansion port. eg if a U16S has its host

port connected to the host computer and an expansion port connects to a U8S, port 11 will be port 11 on the U16S and port 23 will be port 7 on the U8S.

It is also necessary to install jumpers on the parent and child boards. These jumpers are used to set the 'Group ID'. The parent board must have a 'Group ID' of '8' ie a jumper installed on the pair of pins marked '8'. The child board must have a 'Group ID' of '8+1' ie two jumpers, one jumper on the pair of pins marked '8' and one jumper on the pair of pins marked '1'. No other jumpers should be installed on the 'Group ID' block of pins.

Some tags such as 'Hardware' or 'Firmware' do not combine in a meaningful way and so these tags will return the value for the parent charger. If it is desired to get the values of these tags from the downstream charger then it is possible to open and retrieve them from that charger in the usual manner. Opening a charger does not interfere with access to the dock except as to when settings are changed.

Calls to cbrx_connection_set for a dock will result in the relevant tag being set on both the chargers except for the port specific tags which will be directed to the appropriate charger only.

Dictionaries

For each Cambrionix unit, cbrxd uses two dictionaries:

- the "Get dictionary", containing keys for the tags that can be read
- the "Set dictionary", containing keys for the tags that can be written to or can perform an action

The keys available in each dictionary depend on the number of ports and the feature set(s) supported by the unit.

Feature sets

The following feature sets are available:

Feature set	Description
base	Base level functionality supported by all Cambrionix units
sync	Syncing capability
5V	The unit has a fixed 5V power supply
12V	The unit has a 12v power supply
temperature	The unit has a temperature sensor
PD	The unit implements the USB Power Delivery Specification

All Cambrionix units support the “base” feature set.
The range of possible values for a tag in the “base” feature set can be extended if an additional feature set is also available. For example, “Port.n.flags” can only return a flag “S” (port is in sync mode) on a Cambrionix unit that also implements the sync feature set

The “Hardware” key returns a value for the type of Cambrionix unit.
The “HardwareFlags” key returns a set of flags that indicate the feature sets that are supported by the Cambrionix unit.
These are the extra feature sets cbrxd supports for the various types of Cambrionix unit:

Cambrionix unit type returned by “Hardware”	sync	5V	12V	temperature	PD
PP8C	-	+	+	+	-
PP8S	+	+	+	+	-
PP15C	-	+	+	+	-
PP15S	+	+	+	+	-
PS15	+	+	+	+	-
Series8	-	+	-	-	-
U8C-EXT	-	+	+	+	-
U8C	-	+	-	-	-
U8RA	+	+	-	-	-
U8S-EXT	+	+	+	+	-
U8S	+	+	-	-	-
U10C	-	+	-	-	-
U10S	+	+	-	-	-
U12S	+	+	-	-	-
U16S-NL	+	+	-	-	-
PowerSync 4	- *	-	-	+	+

* It is to be noted that while the PowerSync 4 does not implement the “sync” feature set as such, nevertheless it does have sync capabilities and these are always available. This means that there is no need to switch between charge mode and sync mode.

Get Dictionary

Key	Feature set	Description	Example value
SystemTitle	base	The system identification text	cambrionix U8S-EXT 8 Port USB Charge+Sync
Hardware	base	Type of the Cambrionix unit	U8S-EXT
Firmware	base	Firmware version string	1.55
Compiled	base	Timestamp of firmware version	Jul 08 2015 10:43:20
Group	base	Group letter read from PCB jumpers, or "–" if no group jumper was fitted	-
HardwareFlags	Base	Flags indicating whether features are present S = Sync feature set L = 5V feature set E = 12V feature set T = Temperature feature set P = PD feature set	"SLET"
PanelID	base	PanelID number of front panel board, if fitted, or "Absent"/"None"	Absent
Port.n.VID	sync, PD	Vendor ID of the USB device attached to this USB port, if it could be detected. 0 (zero) is returned if it could not be detected	0
Port.n.PID	sync, PD	Product ID of the USB device attached to this USB port, if it could be detected 0 (zero) is returned if it could not be detected	0
Port.n.Manufacturer	sync, PD	Manufacturer as reported by the USB device attached to this USB port, if it could be detected. Empty string is returned if it could not be detected.	""
Port.n.Description	sync, PD	Description as reported by the USB device attached to this USB port, if it could be detected. Empty string is returned if it could not be detected.	""
Port.n.SerialNumber	sync, PD	Serial number as reported by the USB device attached to this USB port, if it could be detected Empty string is returned if it could not be detected.	""
Port.n.USBStrings	sync, PD	A dictionary containing the values for 'Manufacturer', 'Description' and 'SerialNumber' for this USB port	{'SerialNumber': '', 'Description': '', 'Manufacturer': ''}
Port.n.Current_mA	base	Current being delivered to the USB device connected to this USB port in milli-Amperes (mA)	0
Port.n.Voltage_10mV	PD	Voltage being supplied to the port in 10mV	520
Port.n.Flags	base	Port flags separated by spaces. O S B I P C F are mutually exclusive O = USB port Off	R D S

		S = USB port in Sync mode (can only be returned on devices that implement the sync feature set) B = USB port in Biased mode I = USB port in charge mode and Idle P = USB port in charge mode and Profiling C = USB port in charge mode and Charging F = USB port in charge mode and has Finished charging A D are mutually exclusive A = a USB device is Attached to this USB port D = Detached, no USB device is attached E = Errors are present R = system has been Rebooted r = Vbus is being reset during mode change	
Port.n.ProfileID	5V	Profile ID number, or 0 if not charging	0
Port.n.Profiles	5V	List of enabled profiles for this port	1,2,3,4
Port.n.TimeCharging_sec	base	Time in seconds since this USB port started charging an attached device 0 will be returned if the USB port has not started charging an attached device	0
Port.n.TimeCharged_sec	base	Time in seconds since this USB port detected the device has completed charging -1 will be returned if this port has not detected completed charging	-1
Port.n.Energy_Wh	base	Energy the USB device on this USB port has consumed in Watthours (calculated every second)	0.0
Attached	base	A bitfield with one bit set for each port with a device attached, port 1 in bit 0, port 2 in bit 1 and so on	0
nrOfPorts	base	Number of USB ports on the Cambrionix unit	8
TotalCurrent_mA	5V	Total current in mA for all USB ports	0
Uptime_sec	base	Time in seconds the Cambrionix unit has been running since the last reset	151304
FiveVoltRail_V	5V	Current 5V supply voltage in Volt (V)	5.25
TotalPower_W	5V	Total power being consumed on all USB ports in Watts (W)	0.0
FiveVoltRailMin_V	5V	Lowest 5V supply voltage seen in Volt (V)	5.2
FiveVoltRailMax_V	5V	Highest 5V supply voltage seen in Volt (V)	5.25
FiveVoltRail_flags	5V	List of 5V supply rail error flags: UV – undervoltage occurred OV – overvoltage occurred no flags – voltage is acceptable	
TwelveVoltRail_V	12V	Current 12V supply voltage in Volts (V)	12.43
TwelveVoltRailMin_V	12V	Lowest 12V supply voltage seen in Volts (V)	12.31
TwelveVoltRailMax_V	12V	Highest 12V supply voltage seen	12.52
TwelveVoltRail_flags	12V	List of 12V supply rail error flags: UV – undervoltage occurred OV – overvoltage occurred no flags – voltage is acceptable	
InputRail_V	PD	Current input rail supply in Volts (V)	24.03
InputRailMin_V	PD	Lowest input voltage seen in Volts (V)	23.82

InputRailMax_V	PD	Highest input voltage seen in Volts (V)	24.14
InputRail_flags	PD	List of input rail error flags: UV – undervoltage occurred OV – overvoltage occurred no flags – voltage is acceptable	
Temperature_C	temperature	Present PCB temperature in degrees Celsius measured temperatures ≤ 0 °C will return 0 measured temperatures ≥100 °C will return 100	37.7
TemperatureMax_C	temperature	Highest PCB temperature in degrees Celsius measured temperatures ≤ 0 °C will return 0 measured temperatures ≥ 100 °C will return 100	39.9
Temperature_flags	temperature	Temperature error flags: OT – overtemperature event has occurred no flags – temperature is acceptable	
pwm_percent	temperature	Fan speed	0
rebooted	base	A flag indicating if the system has been rebooted since power up True – system has been rebooted False – no reboot has occurred	true
HostPresent	sync, PD	Host is connected to the Cambrionix unit	true
ModeChangeAuto	sync	Mode change from Charge to Sync is automatic	true
FiveVoltRail_Limit_Min_V	5V	Lower limit of the 5V rail that will trigger the error flag	3.5
FiveVoltRail_Limit_Max_V	5V	Upper limit of the 5V rail that will trigger the error flag	5.58
TwelveVoltRail_Limit_Min_V	12v	Lower limit of the 12V rail that will trigger the error flag	9.59
TwelveVoltRail_Limit_Max_V	12v	Upper limit of the 12V rail that will trigger the error flag	14.5
InputRail_Limit_Min_V	PD	Lower limit of the input rail that will trigger the error flag	9.59
InputRail_Limit_Max_V	PD	Upper limit of the input rail that will trigger the error flag	24.7
Temperature_Limit_Max_C	temperature	Upper limit of the acceptable temperature range that will trigger the error flag	65.0
EnabledProfiles	5V	List of global profiles currently enabled	1 2 3 4
Profile.n.enabled	5V	Is global profile n enabled?	false
SecurityArmed	5V	Is security armed?	true / false
Key.<n>	5V	0 if button n has not been pressed since the last time this entry was read 1 if button n has been pressed since the last time this entry was read Double-clicks cannot be detected.	0 / 1
PortInfo.n	base	All available keys and values for this port as a dictionary	
PortsInfo	base	All available information for all ports as a dictionary of dictionaries	
Health	base	All available keys that are not port specific and change dynamically, as a dictionary	
Power	PD	Power variables as a dictionary	

Set Dictionary

Key	Feature set	Description	Possible values
Mode	base	Set all USB Ports mode to	c -- charge mode s – sync mode b – biased o – off Sync mode can only be set on device that implement the “sync” feature set. Biased mode can only be set on devices that implement the “5V” feature set.
Port.n.mode	base	Set USB port N mode to	c -- charge mode s – sync mode b – biased o – off
Port.n.qcmode	PD	Set USB port N quick charge mode to	q – quick charge allowed o – quick charge disallowed
Port.n.led1	base	Set the status of the first LED	0-255 with the LEDs flashing according to the bit pattern represented by the value
Port.n.led2	base	Set the status of the second LED	0-255 with the LEDs flashing according to the bit pattern represented by the value
Port.n.led3	base	Set the status of the third LED	0-255 with the LEDs flashing according to the bit pattern represented by the value
Port.n.leds	base	Set the status of all three LEDs	A 24 bit value consisting of the individual LED settings as 8 bit values shifted and or’ed together ie led1 | (led2 << 8) | (led3 << 16)
Port.n.profiles	5V	Set the list of enabled profiles	A comma separated list of profiles to enable e.g. 1,2,3

ClearLCD	5V	Clear the LCD	true
LCDText.<row>.<column>	5V	Write the string on the LCD at (row, column). Row and column are zero based.	String
RemoteControl	base	Enabled / disable controlling of the unit controls. This will allow the LEDs or LCD to be updated or panel button pushes to be detected.	true / false
SecurityArmed	5V	Enable / disable security feature. If the security is enabled, removal of a device from a port will sound an alarm and flash lights.	true / false
Beep	5V	Beep for the number of milliseconds passed in	Integer
ClearRebootFlag	base	Clear the reboot flag	true
ClearErrorFlags	base	Clear all error flags	true
Reboot	base	Reboot the system now	true
FiveVoltRail.OverVoltage	5V	Force the behaviour of a 5V overvoltage condition	true
FiveVoltRail.UnderVoltage	5V	Force the behaviour of a 5V undervoltage condition	true
TwelveVoltRail.OverVoltage	12v	Force the behaviour of a 12V overvoltage condition	true
TwelveVoltRail.UnderVoltage	12v	Force the behaviour of a 12V undervoltage condition	true
InputRail.OverVoltage	PD	Force the behaviour of an input rail overvoltage condition	true
TwelveVoltRail.UnderVoltage	PD	Force the behaviour of an input rail undervoltage condition	true
Temperature.OverTemperature	temperature	Force the behaviour of an overtemperature condition	true
ProfileEnable.n	5V	Enable or disable the global profile n	true / false
Power	PD	Set the maximum allowable power draw for the unit as a whole, in mW	200000
Port.n.power	PD	Set the maximum allowable power draw from a port, in mW	45000

Socket connections

When using the Python wrapper that provides the cbrxapi module, each time a call is made to the API, a socket is created. This socket is then used to send the command and receive the response before being closed.

If you are writing your own program, in whichever language you choose, you may wish to consider creating a single socket at the start of your communication with the API and keeping this socket open until you wish to stop using the API. Keeping the socket open for the lifetime of your communication with the API will reduce the load on the system and lead to shorter communication cycles with the API.

If you do choose to manage your own socket connections to the API, either as a long lived singleton, or else created on a per use basis, it is important that you do not close the socket before receiving the response from the final command. Closing the socket without waiting to receive the response may lead to the requested operation not being completed, this is especially important on set and close operations.

The API will only accept connections from the local machine.

Timeouts

If there is no activity on an open handle for more than 120s, the handle will be deleted. Subsequent calls attempting to use a deleted handle will fail with CBRXAPI_ERRORCODE_INVALIDHANDLE. Software using the API must be able to cope with this situation and respond accordingly. Software may simply call cbrx_connection_open again in order to obtain a fresh handle.

Controlling the LEDs

The API allows control of the LEDs that are present on some chargers or can be attached to other chargers. By default these LEDs are controlled automatically by the charger firmware to indicate the state that a port is in. In order for the LEDs to be controlled by the API this automatic control must be disabled and this is done by setting the RemoteControl key to be ‘True’.

result = cbrxapi.cbrx_connection_set(handle, "RemoteControl", True)

If you wish to return control of the LEDs to the automatic control then you simply set RemoteControl to be ‘False’.

result = cbrxapi.cbrx_connection_set(handle, "RemoteControl", False)

Control of an LED is achieved by providing an 8 bit value which is interpreted in binary as a pattern that is continuously cycled through. So by setting the value 11110000b, the LED will flash slowly. The LED will be lit where there is a ‘1’ and unlit where there is a ‘0’. Alternatively setting the value 10101010b will make the LED flash fast. The pattern need not be symmetrical so 10010000b will produce two short flashes close together with a longer pause before the cycle repeats.

Any value set for an LED while RemoteControl is False will be overwritten and so have no effect.

Limitations

The API provides a means of controlling most of the features of Cambrionix Universal devices, however there are some limitations.

The API does not currently support:

- automatic logging
- changing profiles
- updating of device firmware

Remote access

The API will only accept connections from the local machine. This is for security reasons. If you wish to access the Cambrionix API from another machine this can be done by creating an ssh tunnel with port forwarding. The technical details for implementing this are outside the scope of this document but there are many tutorials on the internet that explain how to do this.

Troubleshooting

Symptom	Cause	Resolution
jsonrpc.RPCTransportError: [Errno 10061] No connection could be made because the target machine actively refused it	Cambrionix API service/daemon is not running	Start the Cambrionix API service/daemon
jsonrpc.RPCTransportError: [Errno 111] Connection refused	Cambrionix API service/daemon is not running	Start the Cambrionix API service/daemon
avahi_client_new returned Nil for client with Error -26 cbrxd: browser.c:581: avahi_service_browser_new: Assertion `client' failed.	Avahi daemon is not running	Install and start the Avahi daemon
jsonrpc.RPCFault: <RPCFault -10001: 'ID not found' (None)>	The ID passed to cbrx_connection_open is not known to the API, possibly due to the Cambrionix charger having been disconnected	Re-run cbrx_discover and use one of the IDs returned.
jsonrpc.RPCFault: <RPCFault -10002: 'No handling thread' (None)>	The API cannot connect to a remote device such as EtherSync, possibly due to an incorrect password	Set the correct password for the remote device using cbrx_connection_remote_set_password
jsonrpc.RPCFault: <RPCFault -10003: 'Key not found' (None)>	Incorrect or misspelled key used in get request	Check the spelling, including the case of all letters, and check that the key is applicable to the Cambrionix charger in use.
jsonrpc.RPCFault: <RPCFault -10004: 'Error setting value' (None)>	Incorrect or misspelled key used in set request	Check the spelling, including the case of all letters, and check that the key is applicable to the Cambrionix charger in use.
jsonrpc.RPCFault: <RPCFault -10005: 'Invalid handle' (None)>	The handle used was not valid	Use the handle returned by cbrx_connection_open. If there has been no activity for 2 minutes the handle will have been automatically closed, in which case call cbrx_connection_open to receive a new handle.

Error codes

CBRXAPI specific errors

CBRXAPI_ERRORCODE_DROPPED = -10007

Socket connection to remote has been dropped.
The socket connection to a remote Cambrionix unit has been dropped. To continue communication, a socket must be re-established by calling cbrx_connection_open again.

CBRXAPI _ERRORCODE_TIMEOUT = -10006

Timeout on communication.
An operation towards a Cambrionix unit took too long to complete. It may have been disconnected or just slow to respond. It is worth retrying the operation.

CBRXAPI _ERRORCODE_INVALIDHANDLE = -10005

Invalid handle.

The handle passed in to a function is not valid or no longer valid. This could happen either by passing in an incorrect value or if the handle has already been closed (i.e. by cbrxd_closeandlock being called), or the unit has been disconnected from the computer.

CBRXAPI _ERRORCODE_ERRORSETTINGVALUE = -10004

Could not set value.

The (key, value) pair was not acceptable. This could mean the tag does not exist or is misspelled, the value is of the wrong type or the value passed is invalid or out of range.

CBRXAPI _ERRORCODE_KEYNOTFOUND = -10003

Key not found.

A key that is passed in cannot be found. It may be misspelled or not exist in the dictionary for this unit.

CBRXAPI _ERRORCODE_NOHANDLINGTHREAD = -10002

Unable to start handling thread.

The cbrxd needs to open a connection to the Cambrionix unit which will have an internal handling thread. If cbrxd fails to create a handling thread it will not be able to communicate.

For a remote connection this may mean that the ssh connection could not be established due to a bad password or the host key changing.

CBRXAPI _ERRORCODE_IDNOTFOUND = -10001

ID not found.

The unit ID passed in does not represent a Cambrionix unit or it has been disconnected since discovery was last run.

Revision history

Most recent revision first.

0.20
Rebrand

0.19
Add troubleshooting section for some common issues.

0.18
Add PortInfo.n, PortsInfo and Health keys to speed up returning information from API, bump API version to 1.7

0.17
Add HardwareFlags key to the Get dictionary to help check on feature sets available.

0.16
Add keys to dictionaries to manipulate enabled profiles and bump API version to 1.6

0.15
Add description of new cbrx_connection_getdetail command

0.14
Add in support for PowerSync 4 with PD feature set

0.13
Correct python example for reboot
Add in need to install python-setuptools

0.12
Add options and description for ‘Docks’

0.11
Add section on controlling LEDs

0.10
Add commands to allow specifying remote devices explicitly and to clear the list of remote devices

0.9
Add section on use of sockets
Add section on handle timeouts
Add support for EtherSync
Fixed incorrect value for default listening port
Improved response times
Added USB event driven updates

0.8
Add reference to minimum supported firmware level
Fixed current apiversion returned
Add leds, USBStrings and Attached commands

0.7
Remove listed requirement for libgtk from Linux install section
Add install instructions for OS X
Fix typo mistake “Reset” -> “Reboot” in one place
Add mode command to control all ports at once
Fix typo in Minimal Example Python code
Remove erroneous params in cbrx_apiversion JSON example

0.6
API now allows multiple requests in a single TCP connection.

0.5pre11
New keys added to Get Dictionary:
Key.1, Key.2, Key.3, SecurityArmed
New keys added to Set Dictionary:
SecurityArmed

0.5pre10
New keys added to Set Dictionary:
RemoteControl, Beep, ClearLCD, LCDText.row.column, Port.n.led1, Port.n.led2, Port.n.led3

0.5pre9
Linux now supports Port.n.PID and Port.n.VID
Windows installer available
cbrx_connection_id_to_os_reference call added
Unit id is now based on the serial number of the Cambrionix unit
New keys added to Get Dictionary for properties of an attached USB device:
Port.n.SerialNumber, Port.n.Manufacturer, Port.n.Description

0.5pre8 Initial public revision