cambrionix

Cambrionix Universal Charger API

Cambrionix API

Introduction

This is a description of the Cambrionix API that can be used to control Cambrionix 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.

Revision history

Most recent revision first.

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

Prerequisites

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

FTDI 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 driver installed.

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 language.

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 42434.
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

MacOS installation

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

Windows installation

For Windows a self extracting installer is provided that will set up cbrxd to run as a Windows service.

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, don't separate them from the main binary.
- Main binary will end up in /usr/local/bin
- Documentation will end up in /usr/local/share/cbrxd/doc
- Examples will end up in /usr/local/share/cbrxd/examples
- Python installer for json-rpc will end up in /usr/local/share/cbrxd/python
- Setup scripts will end up in /usr/local/share/cbrxd/setup

- Make sure cbrxd is run by a user that has permission to the /dev/ttyUSB* devices.

- For initial testing cbrxd can be started up from the command line.

```
$ cbrxd
```

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:

```
$ 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
```

Linux installation -- setting up cbrxd to start up automatically

Two 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 cp /usr/local/share/cbrxd/setup/cbrxd.sh /etc/init.d
% sudo update-rc.d cbrxd.sh default
```

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 cp /usr/local/share/cbrxd/setup/cbrxd.service /etc/systemd/system
% sudo systemctl enable cbrxd
% sudo systemctl start cbrxd
```

Command line options for cbrxd

`--version:` Verify the installed version of cbrxd, will not start cbrxd up:
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 42434 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 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(id)
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.

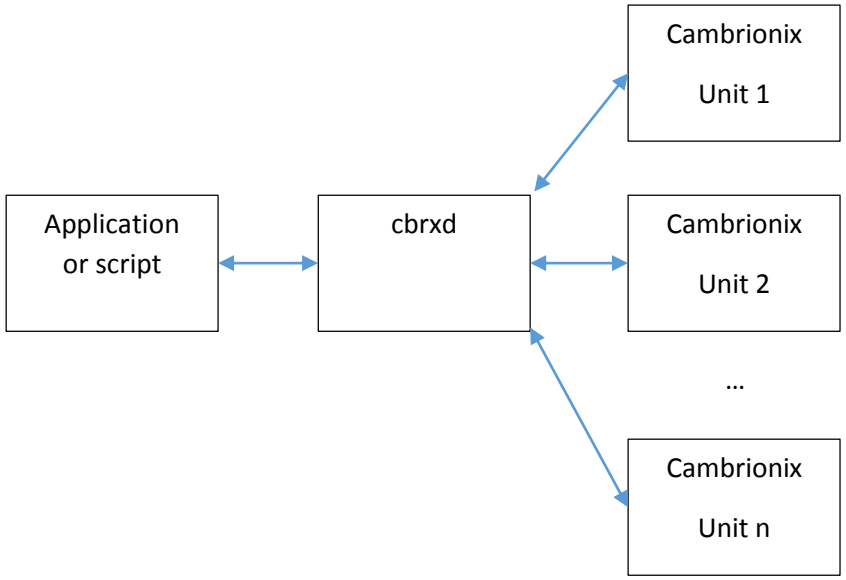
The Python JSON-RPC used causes an exception for an error response with the following mapping:
member code is returned in e.error_code
member message is returned in e.error_message
member data is returned in e.error_data.

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 you can do
  `cbrx_connection_set (connectionHandle, "Port.2.mode", "o")`
- to reset a Cambrionix unit you can do
  `cbrx_connection_set (connectionHandle, "Reset", True)`
- to get the number of USB ports of a Cambrionix unit you can do a "get nrOfPorts" request,
  `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 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 0.

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

Example Python call:

```
cbrxapi.cbrx_apiversion()
```

Example JSONRPC request:

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

}
```

Example successful response:

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

}
```

Discovery

cbrx_discover

Discover Cambrionix units.

Input: "local" for Cambrionix units attached to the local machine

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.

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_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:

a unit ID as returned by a previous call to cbrx_discover

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")
```

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.

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_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.

Example Python call:

```
cbrxapi.cbrx_connection_setdictionary(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 wasn't 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.

Example Python call:

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

Example JSONRPC request:

```
{     "jsonrpc": "2.0",
      "method": "cbrx_connection_closeandlock",
      "params": ["COM20"],
      "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 wasn't 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.

Example Python call:

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

Example JSONRPC request

```
{     "jsonrpc": "2.0",
      "method": "cbrx_connection_unlock",
      "params": ["COM20"],
      "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"

      }

}
```

Dictionaries

For each Cambrionix unit, cbrxd can return 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 returned depend on 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
12v	the unit has a 12v power supply
temp	the unit has a temperature sensor

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 device that also implements the sync feature set

The “Hardware” key returns a value for the type of Cambrionix unit.
These are the extra feature sets cbrxd supports for the various types of Cambrionix unit:

Cambrionix unit type returned by “Hardware”	**sync**	**12v**	**temp**
PP8C	-	+	+
PP8S	+	+	+
PP15C	-	+	+
PP15S	+	+	+
Series8	-	-	-
U8C-EXT	-	+	+
U8C	-	-	-
U8RA	+	-	-
U8S-EXT	+	+	+
U8S	+	-	-
U10C	-	-	-
U10S	+	-	-
U12S	+	-	-
U16S-NL	+	-	-

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	-
PanelID	base	PanelID number of front panel board, if fitted, or "Absent"/"None"	Absent
Port.n.VID	sync	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	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	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	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	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.Current_mA	base	Current being delivered to the USB device connected to this USB port in milli-amperes (mA)	0
Port.n.Flags	base	Port flags separated by spaces. O S B I P C F are mutually exclusive O = USB port Off 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	R D S

		A = a USB device is Attached to this USB port D = Detached, no USB device attached E = Errors are present R = system has been Rebooted r = Vbus is being reset during mode change	
Port.n.ProfileID	base	Profile ID number, or 0 if not charging	0
Port.n.TimeCharging_sec	base	Time in seconds the USB port has been charging for 0 will be returned if the USB port is not charging	0
Port.n.TimeCharged_sec	base	Time in seconds the USB port has completed charging -1 will be returned if the port has not 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
nrOfPorts	base	Number of USB ports on the Cambrionix unit	8
TotalCurrent_mA	base	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	base	Current 5V supply voltage in Volt (V)	5.25
TotalPower_W	base	Total power being consumed on all USB ports in Watts (W)	0.0
FiveVoltRailMin_V	base	Lowest 5V supply voltage seen in Volt (V)	5.2
FiveVoltRailMax_V	base	Highest 5V supply voltage seen in Volt (V)	5.25
FiveVoltRail_flags	base	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 Volt (V)	12.43
TwelveVoltRailMin_V	12v	Lowest 12V supply voltage seen in Volt (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	
Temperature_C	temp	Present PCB temperature in degrees Celsius measured temperatures ≤ 0 °C will return 0 measured temperatures ≥100 °C Celsius will return 100	37.7
TemperatureMax_C	temp	Highest PCB temperature in degrees Celsius measured temperatures ≤ 0 °C will return 0 measured temperatures ≥ 100 °C Celsius will return 100	39.9
Temperature_flags	temp	Temperature error flags: OT – overtemperature event has occurred no flags – temperature is acceptable	
pwm_percent	temp	Fan speed	0
rebooted	base	A flag indicating if the system has been rebooted since power up	true

		True – system has been rebooted False – no reboot has occurred	
HostPresent	sync	Host is connected to the Cambrionix unit	true
ModeChangeAuto	sync	Mode change from Charge to Sync is automatic	true
FiveVoltRail_Limit_Min_V	base	Lower limit of the 5V rail that will trigger the error flag	3.5
FiveVoltRail_Limit_Max_V	base	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
Temperature_Limit_Max_C	temp	Upper limit of the acceptable temperature range that will trigger the error flag	65.0
Profile.n.enabled	base	Is global profile n enabled?	false
SecurityArmed	base	Is security armed?	true / false
Key.<n>	base	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

Set Dictionary

Key	Feature set	Description	Possible values
Port.<N>.mode	base	Set USB port N 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.
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
ClearLCD	base	Clear the LCD	true
LCDText.<row>.<column>	base	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	base	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	base	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	base	Force the behaviour of a 5V overvoltage condition	true
FiveVoltRail.UnderVoltage	base	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
Temperature.OverTemperature	temp	Force the behaviour of an overtemperature condition	true

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:

- remote/network attached units
- automatic logging
- changing profiles
- updating of device firmware

Error codes

CBRXAPI specific errors

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.

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.