cbus Documentation

Release 0.1-dev

Michael Farrell

December 10, 2014

1 Introduction **3**
1.1 What is CBus? . . . 3
1.2 Clipsal's official interfaces . . . 3
1.3 So where does libcbus come in? . . . 4
1.4 Installing . . . 4

2 Hacking **7**
2.1 Official documentation . . . 7
2.2 CNI / network protocol . . . 7
2.3 Setting up a fake CNI and sniffing the protocol . . . 8
2.4 USB support / 5500PCU . . . 8
2.5 Unit Tests . . . 9

3 CNI Discovery **11**
3.1 Discovery Query . . . 11
3.2 Discovery Reply . . . 11

4 Wiser **13**
4.1 Downloading SWFs . . . 13
4.2 Protocol . . . 14
4.3 Getting a shell . . . 15
4.4 CFTP . . . 16
4.5 Firmware image . . . 19

5 daemons **21**
5.1 cdbusd . . . 21
5.2 dbuspcid . . . 21
5.3 sage . . . 21
5.4 staged . . . 28

6 dump_labels utility **29**
6.1 Invocation . . . 29

7 D-Bus Client API **31**

8 libcbus module index **33**
8.1 cbus Package . . . 33

9 Indices and tables **49**

Python Module Index 51

Project Page / Source repository: https://github.com/micolous/cbus

libcbus is a set of Python libraries for interacting with Clipsal C-Bus.

Contents:

CHAPTER 1

Introduction

Welcome to libcbus!

This is a Python library for interacting with Clipsal CBus units, and provides some additional utility functions, such as:

- A high-level DBus-based API for sharing a CBus PCI with multiple application on a computer.
- A library for parsing information from CBus Toolkit project backup files.
- A “fake PCI” test server for parsing data sent by CBus applications.

1.1 What is CBus?

CBus is a home automation and electrical control system made by Clipsal. It’s also known as Square D in the United States, and sold under other brands worldwide by Schnider Electric.

It uses low voltage wiring for light switches (panels) and other sensors, and centrally-fed dimmer and relay controls for devices (such as lights).

CBus has a unit called the PCI (PC Interface) and the CNI (CBus Network Interface) which interactions with the CBus network via Serial and TCP/IPv4 respectively. These use a common interface described by the Serial Interface Guide, and other public CBus specification documents.

The PCI also has a USB variant which includes a USB to Serial converter.

1.2 Clipsal’s official interfaces

In addition to protocol documentation, Clipsal also provide two systems for interacting with CBus, `libcbm` and C-Gate. Clipsal’s own software (like Toolkit) and hardware (like Wiser) uses this to interact with CBus networks over serial and IPv4.

1.2.1 Comparison to `libcbm`

`libcbm` supports to C-Bus protocol completely, including conforming to the various “protocol certification levels”. It is closed source and written in C, and only will work with ia32 Windows and Linux systems. It is distributed for Linux as a static library in an RPM.

`libcbus` supports only the lighting application (at present). It is open source (LGPL3+) and written in Python, and will work with any Python supported platform.

`libcbus` also includes an abstraction daemon called `cdbusd` which will allow multiple applications to simultaneously use the PCI. This daemon requires D-Bus, which is not available on Windows. Other components of `libcbus` will continue to function.

1.2.2 Comparison to C-Gate

C-Gate is Clipsal's own C-Bus control software. It is a closed source application written in Java, that uses the SerialIO library (also closed source) or sockets to communicate with a PCI.

Toolkit itself uses C-Gate in order to communicate with the PCI. It supports a wide range of operations through it's own protocol, including reprogramming units on the network.

However, the SerialIO library included with C-Gate is only available on 32-bit platforms, and even then only on Windows and ancient versions of Linux.

1.3 So where does libcbus come in?

libcbus primarily provides three ways to communicate with C-Bus, with varying levels of complexity and abstraction:

- A low level API which allows direct encoding and decoding of packets. It exposes parts of the packet as classes with attributes.
- A medium level API which handles access to the C-Bus PCI through the Twisted networking library and PySerial. You can insert your own protocol handler, or work with the lower level API in order to access the library at a level that suits you. There are both server (FakePCI) and client interfaces.
- A high level API which provides access to C-Bus over DBus. This allows multiple applications on your computer to interact with the CBus network in a simple way, and allows you to use other DBus supporting languages (such as bash, C, C#, Perl, Python, Ruby, and Vala) to interact with the network through a single PCI.

libcbus does this using completely open source code (LGPLv3), and works across all Python supported platforms. Platforms that don't support DBus (such as Windows) will be able to use the lower level APIs only, and lack the sharing functionality.

I've tested this primarily with Linux on armel, armhf, amd64 and i386, and Windows on amd64.

1.4 Installing

Note: This section is incomplete.

1.4.1 Linux

Most Linux distributions have D-Bus installed by default. As a result, you should only need to install the Python bindings:

- python-dbus

1.4.2 Mac OS X

In order to run this software, you'll need to first install Xcode (from the App Store) and MacPorts.

You then need to install dbus and the Python bindings for it:

```
# port install dbus +universal
```

1.4.3 Windows

Windows doesn't have a D-Bus, so you can't use `cdbusd`.

However, you can use the libraries with Twisted directly. But most of the applications interface with `cdbusd`.

CHAPTER 2

Hacking

Information about using the hardware and software.

2.1 Official documentation

Official serial protocol documentation is available from Clipsal's website: http://training.clipsal.com/downloads/OpenCBus/OpenCBusProtocolDownloads.html

You should generally implement software in conjunction with reading these guides. This library provides a fairly low level API for parsing and generating CBus packets, and understanding what is happening on a lower level is needed when understanding use of this library.

There is a large amount of documentation in there that says "these items are deprecated and shouldn't be used". I've noticed a lot that C-Gate and Toolkit will interact with the hardware in these "deprecated" ways...

This doesn't mean implement the library to talk this way. You should implement it properly. Just be aware than when working with implementing a fake PCI or parsing out packets that Clipsal's software generated, be aware they'll do strange and undocumented things.

2.1.1 Geoffry Bennett's reverse engineering notes (2001 - 2004)

Geoffry Bennett gave a talk at a LinuxSA meeting in 2001 and at Linux.conf.au 2004 about his experiences with reverse engineering C-Bus. At the time there was no official protocol documentation available.

The Linux.conf.au 2004 notes cover a lot more information, includes some information about dumping and reverse engineering the contents of NVRAM in units, a Perl client library, and an emulator used for older versions of the Clipsal programming software.

2.2 CNI / network protocol

The C-Bus Toolkit software has a CNI (network) interface mode. This is really just the serial protocol over a TCP socket. Note that libcbus does not currently implement a CNI client. There's a further discovery protocol, however this requires special implementation which has not been done.

2.3 Setting up a fake CNI and sniffing the protocol

If you want to see how Toolkit interacts with a Serial PCI, use the `tcp_serial_redirect.py` script from the pySerial example scripts. This can be run even on a non-Windows machine, for dealing with pesky 100 different revisions of PL2303 USB-Serial adapters that require different and conflicting Windows drivers. For example:

```
$ python tcp_serial_redirect.py -p /dev/ttyUSB0 -P 22222
```

Congratulations, you now have turned your computer and a 5500PC into a 5500CN without writing a single line of custom code, and saved about 200$. Even a Beaglebone can be had for less than 200$. ;)

Go into Toolkit, set the Default Interface type to "IP Address (CNI)" with the IP and port of the machine running the serial redirector.

You can then use tools like Wireshark to monitor interactions with the C-Bus PCI, instead of using kernel hacks to sniff serial, other redirects, or wireing up your own serial sniffer device. This will aid if you wish to use undocumented commands, or isolate issues in the Clipsal documentation.

You could also use this with tools like C-Gate to get a higher level interface with the C-Bus PCI.

2.4 USB support / 5500PCU

Clipsal's driver is not digitally signed.

It appears to use the driver `silabser.sys` on Windows, which corresponds to a Silicon Labs CP210X USB-serial bridge. `cbususb.inf` lists the following products:

- `10C4:EA60`: Generic SiLabs CP210X
- `166A:0101`: C-Bus Multi-room Audio Matrix Switcher (560884)
- `166A:0201`: C-Bus Pascal/Programmable Automation Controller (5500PACA)
- `166A:0301`: C-Bus Wireless PC Interface (5800PC). This appears to be an unreleased product.
- `166A:0303`: C-Bus Wired PC Interface (5500PCU)
- `166A:0304`: C-Bus Black & White Touchscreen Mk2 (5000CT2)
- `166A:0305`: C-Bus C-Touch Spectrum Colour Touchscreen (C-5000CT2)
- `166A:0401`: C-Bus Architectural Dimmer (L51xx series)

2.4.1 Linux kernel module

The `cp210x` kernel module in Linux 2.6.30 and later supports this chipset. However, only the generic adapter and 5500PCU device IDs are included with the kernel for versions before 3.2.22 and 3.5-rc6.

Your distribution vendor may backport the patches in to other kernel versions.

To see which devices your kernel supports, run the following command:

```
$ /sbin/modinfo cp210x | grep v166A
```

If the following is returned, you only have support for the 5500PCU:

```
alias:          usb:v166Ap0303d*dc*dsc*dp*ic*isc*ip*
```

If more lines come back, then your kernel supports all the hardware that is known about at this time.

2.5 Unit Tests

There's some basic unit tests that are written that require you have the `nosetests` package (`nose` on pip).

When you run `nosetests`, it will discover all the unit tests in the package and try to run them.

I'm targetting Python 2.6 and 2.7 at this time. I'll have a look into Python 3 support when some more backend libraries that this project requires work on Python 3. Patches still welcome.

When implementing a new application, you should copy all of the examples given in the documentation of that application into some tests for that application. Be careful though, in some instances I have found errors in Clipsal's documentation, so double check to make sure that the examples are correct. If you find errors in Clipsal's documentation, you should email them about it.

CNI Discovery

At the moment this is a rather unorganised set of notes while I'm still figuring out the protocol.

I've started working on a test program for dissecting the protcol in `cbus/discovery_test.py`.

3.1 Discovery Query

A client will broadcast a UDP packet on 255.255.255.255:20050.

Data structure is as follows:

```
char[4] command  = "CB 80 00 00"  // CBUS_DISCOVERY_QUERY
char[4] unknown1 = "00 00 00 00"
char[4] unknown2 = "01 01 01 0B"
char[4] unknown3 = "01 1D 80 01"
char[3] unknown4 = "02 47 FF"
```

Example packet:

```
cb:80:00:00:00:00:00:00:01:01:01:0b:01:1d:80:01:02:47:ff

0xcb, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x01, 0x01, 0x01, 0x0b, 0x01, 0x1d, 0x80, 0x01,
0x02, 0x47, 0xff
```

3.2 Discovery Reply

Replies are sent back to the querying client on port 20050.

Values are in big-endian format (network byte order):

```
char[4]  magic      = 0xcb, 0x81, 0x00, 0x00   CBUS_DISCOVERY_REPLY
char[4]  unknown1   = 0x00, 0x00, 0x00, 0x01   //  0x20, 0xe8, 0xf5, 0x52
char[4]  unknown2   = 0x81, 0x01, 0x00, 0x01
char     product_id = 0x03 // 0x01
char[4]  unknown4   = 0x81, 0x0b, 0x00, 0x02
uint16   port       = 0x27, 0x11  (10001)
char[4]  unknown5   = 0x81, 0x1d, 0x00, 0x01
char     unknown6   = 0x00 // 0x01     (not a flag for "in use")
char[4]  unknown7   = 0x80, 0x01, 0x00, 0x02
char[2]  unknown8   = 0x66, 0x1e // 0x8c, 0x26  (may be a checksum, but doesn't appear to be used)
```

Product IDs:

- `01`: CNI2
- `02`: Hidden – Toolkit ignores packets with this product ID. May be used for internal development.
- `03`: WISER
- Other values: “unknown”

Example packet data:

```
Recv

Client 1 (172.26.1.81)
CNI2 port 10001, "not accessible" (controlled by a WISER)

0xcb, 0x81, 0x00, 0x00, 0x20, 0xe8, 0xf5, 0x52,
0x81, 0x01, 0x00, 0x01, 0x01, 0x81, 0x0b, 0x00,
0x02, 0x27, 0x11, 0x81, 0x1d, 0x00, 0x01, 0x01,
0x80, 0x01, 0x00, 0x02, 0x8c, 0x26

Client 2 (172.26.1.80)
WISER port 10001

0xcb, 0x81, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
0x81, 0x01, 0x00, 0x01, 0x03, 0x81, 0x0b, 0x00,
0x02, 0x27, 0x11, 0x81, 0x1d, 0x00, 0x01, 0x00,
0x80, 0x01, 0x00, 0x02, 0x66, 0x1e

b = "\xcb\x81\x00\x00\x00\x00\x00\x01\x81\x01\x00\x01\x03\x81\x0b\x00\x02’\x11\x81\x1d\x00\x01\x00\x8
```

Wiser

The Wiser is a rebadged SparkLAN WRTR-501 802.11b/g/draft-n WiFi Router with custom firmware. It runs an embedded Linux system, with an expanded web interface for hosting Flash/XMLSocket based control of CBus.

According to the source code release from Clipsal, this runs Linux 2.6.17.14. The kernel configuration indicates that the board is a `fv13xx` ARM system. Also used by:

- Airlink101 AR680W
- PCi MZK-W04N

XMLSocket is also used by the iPhone version of the control software.

At the moment this is a rather unorganised set of notes while I'm still figuring out the protocol and investigating the inner workings of the system.

Note: In XML outputs in this document, new-line characters and basic formatting whitespace has been added to improve readability. The original data does not contain this, unless otherwise indicated.

Note: This project provides it's own replacement for Wiser's web interface, called *sage*.

4.1 Downloading SWFs

First step is you are directed to the page `/clipsal/resources/wiserui.html`. This in turn loads the SWF `/clipsal/resources/wiserui.swf`.

As this is SWF, there is a cross-domain access policy in place to allow the SWF to connect back to the server on other ports:

```
<cross-domain-policy>
  <allow-access-from domain="*" secure="false" to-ports="8888,8889"/>
</cross-domain-policy>
```

This configuration appears to be one that allows anything to make requests to the Wiser. So you could write your own implementation of the Wiser control UI and have it connect back, or if you use a well-known address for the Wiser, any Flash applet on the internet could!

The resources and API classes are stored in `/clipsal/resources/resources.swf`. This contains things like the cbus_controller class which is used to establish Flash XMLSocket connections.

4.2 Protocol

4.2.1 Discovery and Handshake

After the SWF is started, it loads the configuration file from `/clipsal/resources/local_config.xml`. This looks like:

```
<local_config version="1.0">
  <wiser ip="XXX.XXX.XXX.XXX" port="8888" remote_url="" remote_port="8336"
         remote="0" wan="0"/>
  <client name="Web UI" fullscreen="0" http_auth="0" local_file_access="1"
          local_project="0" local_skin_definition="0"/>
</local_config>
```

Here we see the internal IP address of the Wiser, and the port that is used for XMLSockets requests (`port`). `remote_port` indicates the port used by the CFTP daemon.

4.2.2 Authentication

There is a basic authentication system in place on some of the sockets. This can be established by retrieving the key from `/clipsal/resources/projectorkey.xml`. This file looks like:

```
<cbus_auth_data value="0x12345678"/>
```

This projector key is generated when a project file is first created by PICED. The projector key stays the same for all projects created during that particular run of PICED.

As a result, rebooting Wiser or changing the HTTP password **does not change this key**, and as a result it is not an effective measure for preventing access to the CBus XMLSocket protocol.

Once this projector key is acquired, it may be **reused in perpetuity**. It is only possible to change this key by creating an entirely new project file in PICED, during a different run of PICED, and transferring this to the Wiser.

4.2.3 Connecting

There is now enough information to connect to the XMLSocket service on port 8888 of the Wiser (or “port” in `local_config.xml`).

So to start the connection we need to send some commands off to the server to handshake.

This starts with a command called `<cbus_auth_cmd>`. This has three attributes, required **exactly** in this order:

```
<cbus_auth_cmd value="0x12345678" cbc_version="3.7.0" count="0" />
```

- value is the value of the cbus_auth_data retrieved in the previous step.
- cbc_version is the version of the SWF being used. This is found in wiserui.swf, in the variable “cbc_version”.
- count is the number of times that this session has attempted to authenticate. Set this to 0.

You could also request the project files and skin files in one shot, like this:

```
<cbus_auth_cmd value="0x12345678" cbc_version="3.7.0" count="0" />
<project-file-request />
<skin-file-request />
```

The Wiser responds with a message like this:

```
<ka cbus_connected="1" />
<cbd_version version="Kona_1.24.0" />
<net_status cni_transparent="0" cni="1" cftp="1" cbus="1" ntp="0" />
<cbus_event app="0xdf" name="cbusTimeChanged" time="120103102012.43" dst="0" ntp="0" />
```

4.2.4 Project and Skin

It also returns a `<Touchscreen>` XML which is a form of the project file, and a `<skin>` XML which contains localised strings and resource image references.

This can also be downloaded from `/clipsal/resources/project.xml` and `/clipsal/resources/skin_definition.xml`, so you can just establish a connection without requesting these files over the XMLSocket. Potentially this could be more reliable.

The project file contains all of the programming in use on the Wiser, button assignments and schedules. It can also contain additional metadata about the installation, if the installer has filled this in.

4.2.5 XMLSocket protocol for dummies

Adobe's documentation describes the XMLSocket protocol as sending XML documents in either direction on the TCP socket, terminated by a null character.

It is like a simple version of WebSockets – client and server may send data at any time, there is no synchronous response mechanism, and very easy to implement.

The XML documents sent do not require the typical XML stanzas at the start of the file specifying encoding, and may also contain multiple top-level (document) elements.

There are third-party client and server libraries available for this protocol.

4.3 Getting a shell

There is console access available via a web interface on the Wiser, using `/console.asp`. It appears to be taken from some Belkin or Linksys reference firmware image?

Redirection of output to a file using > doesn't work correctly in the shell. Regular pipes (`|`) do work.

Only `stdout` is displayed, not `stderr`.

4.3.1 NVRAM

You can dump the NVRAM:

```
$ nvram show
...
wan_proto=dhcp
wan_ipaddr=0.0.0.0
wan_netmask=0.0.0.0
wan_gateway=0.0.0.0
wan_winsserv=
...
```

4.4 CFTP

CFTP is a service which acts as a back-door into the device. It runs on port 8336, and is managed by the service **cftp_daemon**.

It has a hard-coded password to access the service. Despite the name, it doesn't actually implement FTP. It is used by Clipsal's programming software in order to manage the device. It appears to have the following functionality:

- Manage port forwards inside of the network when the device is acting as the router for the network. Unknown how this is controlled.
- Reflash the contents of partition 6 of FLASH (label: `clipsal`). Appears to be a gzip-compressed tarball, which gets extracted to `/www/clipsal/resources`.

Communication with the server is done with a simple text-based protocol, with the UNIX newline character indicating the end of command. Do not send DOS or other style linefeeds as this will not work.

If the daemon does not understand your command, it will simply send no response.

4.4.1 Startup process

On startup, the process will:

1. Delete `/tmp/*.tar.gz`.
2. Copy the contents of `/dev/mtblock/6` to `/tmp/test.cta`.
3. Mount a new ramfs to `/www/clipsal/resources/`
4. Extract `settings.conf` from the gzip-compressed tarball `/tmp/test.cta` to `/www/clipsal/resources/`.
5. Read daemon configuration from `settings.conf`.
6. Extract all files from the tarball to `/www/clipsal/resources/`.

4.4.2 Unauthenticated state

Connecting to the service yields a welcome message:

```
200 Welcome
```

PASS

Client command:

```
PASS bloop
```

The server will respond that you are logged in successfully, and transition your connection to the authenticated state:

```
201 Logged in
```

Note: There is no way to change this password. It is hard coded in Wiser's firmware.

Sending other passwords yield no response.

4.4.3 Authenticated state

When in the authenticated state, the network code appears to be far less robust. Sending large commands causes the daemon to crash.

This may be an effective and easy way to disable **cftp_daemon** on the device.

PASS

Client command:

```
PASS bloop
```

Server response:

```
201 Logged in
```

Transitions to the authenticated state. Has no effect in authenticated mode.

Note: There is no way to change this password. It is hard coded in Wiser's firmware.

Sending other passwords yield no response.

VERINFO

Client command:

```
VERINFO
```

Server response:

```
202-HomeGateVersion=4.0.41.0
202-CTCServerVersion=Kona_1.24.0
202-UnitName=EXAMPLE
202 WindowsOSVersion=5.1.2600 Service Pack 2
```

Retrieves information about the version of CFTP running on the Wiser, and the C-Bus network's project name.

The WindowsOSVersion information is a hard-coded string.

HGSTATUS

Client command:

```
HGSTATUS
```

Server response:

```
202-HGRUNNING=False
202-HGLOGGING=False
202 CURRPROJ=C:\HomeGate\Projects\Current\EXAMPLEproj.tar.gz
```

Retrieves the current project name running on the Wiser, and status of "HG"? This is hard coded to always return False to both HGRUNNING and HGLOGGING.

The path is faked by the daemon, with "EXAMPLE" replaced by the project name.

GETFILELIST

Client command:

```
GETFILELIST
```

Server response:

```
202 FILE1=C:\HomeGate\Projects\Current\EXAMPLEproj.tar.gz
```

Retrieves a list of "files" on the device associated with the project. This only returns the project file.

The path is faked by the daemon, with "EXAMPLE" replaced by the project name.

GETPROJ

Client command:

```
GETPROJ
```

Server response:

```
202-Port=8337
202 FILE=C:\HomeGate\Projects\Current\EXAMPLEproj.tar.gz
```

Returns the "project filename" for the contents of flash partition 6. The path information is hard coded and fake, with "EXAMPLE" replaced by the project name.

INSTALL

Client command:

```
INSTALL PROJECT example.tar.gz
```

Server response:

```
202 Port=8337
```

Starts an out of band transfer for overwriting the Wiser's project file.

The server opens up another TCP server on a different port (on Wiser, this is always 8337) in order to accept the file transfer out of band.

4.4.4 Project file transfer

Project file transfer is done on another port (always 8337), and initiated by the `INSTALL` command.

The client immediately sends:

```
FILE example.tar.gz
```

This is then immediately followed by a UNIX newline character, and then the file length as a 32-bit unsigned big-endian integer.

Files must not be bigger than 512kB, or the transfer will be rejected by the Wiser. File names must end in ".tar.gz".

Projects must also not extract to a size greater than about 1 MiB. Wiser stores the contents of this archive in ramfs, so larger archives will use all available RAM on the Wiser, and cannot be freed, leading to Linux's oomkiller to run or processes to fail to dynamically allocate memory. This has the potential in turn to partially brick the Wiser

– **cftp_daemon** will not be able to copy a new project file into RAM temporarily for flashing, and may be permanently stuck in this state. This partial brick state could probably gotten around by writing NULL over the contents of `/dev/mtdblock/6`, then transferring a new project file.

4.5 Firmware image

Firmware image for the device is bundled with the PICED software as `Firmware/firmware_1_24_0.img`. The tool binwalk shows the layout of the firmware image:

```
0x13        uImage header, header size: 64 bytes, header CRC: 0x2781C02C,
            created: Mon Oct  3 11:26:33 2011, image size: 722439 bytes,
            Data Address: 0x40008000, Entry Point: 0x40008000,
            data CRC: 0xF7547123, OS: Linux, CPU: ARM,
            image type: OS Kernel Image, compression type: lzma,
            image name: Linux-2.6.17

0x53        LZMA compressed data, properties: 0x5D,
            dictionary size: 8388608 bytes, uncompressed size: 2015280 bytes

0xC0013     Squashfs filesystem, little endian, version 2.1,
            size: 1736392 bytes, 435 inodes, blocksize: 65536 bytes,
            created: Mon Oct  3 11:27:23 2011
```

Appears to be a uBoot image with some extra headers on the image.

4.5.1 Extracting root filesystem

The version of squashfs used by the root filesystem is very old, and current Linux kernels are incapable of mounting it. It requires an LZMA version of squashfs-2.1 in order to extract it, available from firmware-mod-kit. Their SVN repository contains all the components needed:

```
$ svn co https://firmware-mod-kit.googlecode.com/svn/trunk/src/lzma/
$ svn co https://firmware-mod-kit.googlecode.com/svn/trunk/src/squashfs-2.1-r2/
$ cd squashfs-2.1-r2
$ make
```

Once built, extract the root filesystem with:

```
$ binwalk -D squashfs:squashfs firmware_1_24_0.img
$ ./squashfs-2.1-r2/unsquashfs-lzma C0013.squashfs
```

This will then give an extracted copy of the root filesystem in the directory `squashfs-root`.

4.5.2 Filesystem observations

These are things that need some more investigation:

- NTP client which has 32 hard-coded NTP server IP addresses.

CHAPTER 5

daemons

The cbus.daemon package defines a number of daemons used to interact with the C-Bus network.

5.1 cdbusd

cdbusd is a daemon that is used to allow multiple applications on the same computer to share use of a PC Interface.

The daemon also provides clock services to the C-Bus network.

It is recommended when building C-Bus applications that you use the D-Bus interface to the network.

For more information on the D-Bus client interface, please see *D-Bus Client API*.

5.2 dbuspcid

dbuspcid is a daemon that emulates a PC Interface, allowing you to use it to connect to cdbusd.

This allows you to use applications on other computers to communicate with a virtual PCI, which in turn passes events to cdbusd. It also allows you to use applications that do not support the cdbusd API directly.

5.3 sage

sage is a web interface for lighting controls, similar to *Wiser*, except it doesn't suck:

- It has no dependancy on Flash player or a device-specific app (it uses WebSockets instead).
- It has a very minimalist, touch-friendly UI based on jQuery Mobile. This UI is also used on desktop.
- It works as a "web app" on iOS 4.2 and later (due to patchy WebSockets support).
- It has no requirement for a ethernet-based PCI (serial/USB are fine).
- It doesn't have hard coded backdoors and changing the password actually locks previous users out. (However some browsers don't implement support for HTTP Authentication requests on WebSockets)

It connects to *cdbusd* as it's abstraction layer, and only presents lighting events to clients.

It is made of some parts:

- `saged` which is a backend WebSockets server that translates WebSockets messages into messages in cdbusd, and implements some basic access controls.
- `sageclient.js` which implements the saged WebSockets protocol in JavaScript.
- `sageui.js` which implements the UI of sage itself, which is built on jQuery Mobile.

5.3.1 Running

sage requires that you have a running instance of *cdbusd* first.

Once running, copy the resources in `cbus/sage_root` into a web-accessible folder on the same computer where you will run sage.

Note: If you are serving the files from HTTPS, you may also need a way to make sage's WebSockets service accessible by HTTPS (wss), due to browser security constraints. sage does not support this yet, so the sage files need to be all served via HTTP.

Then start up the **sage** daemon:

```
$ python -m cbus.daemons.saged -H 192.0.2.1 -W
```

Where `192.0.2.1` is your server's LAN IP address. This will run a WebSockets server on port 8080.

Note: Don't expose sage to the internet. Anyone on the internet will be able to use sage if you do this. Always use a VPN to access sage remotely.

You then need to create a project file called `project.json` in your sage_root folder. This file is in JSON format. As an example:

```
{
  "saged": "ws://192.0.2.1:8080/",
  "locations": {
    "0": "Living Areas",
    "1": "Bedrooms",
    "2": "Outside"
  },

  "widgets": {
    "1": {
       "name": "Kitchen",
       "type": "switch",
       "locations": {"0": 1}
    },
    "2": {
      "name": "Patio",
      "type": "switch",
      "locations": {"0": 100, "2": 1}
    }
  }
}
```

The important elements of the configuration are:

- `saged`: The URI of the WebSockets server. When running saged without the `-W` option (in standalone mode), the WebSockets server listens on `/saged`, not the root path..

- `locations`: A dictionary of locations, with the key representing the ID of the location, and the value representing the name to display. Location `0` must exist, and is shown by default.
- `widgets`: A dictionary of group addresses describing what widgets to display.
  - The key is the group address to control.
  - The option `name` sets the name to display in the UI.
  - The option `type` selects the type of widget to use:
    - `switch` displays a simple switch
    - `slider` displays a slider.
    - `switch_slider` displays both a switch and a slider.
  - The option `locations` is a dictionary which sets the locations where the widget should be displayed (indicated by the key), and the order in which it should be shown (indicated by the value, higher values are shown lower in the list).

Note: By default, constraints shown here about widget types are not enforced on the server side, and any group address may be sent commands.

If there are group addresses you wish to restrict access to, use the `-a` option to **sage** to only allow certain group addresses and deny all others (whitelist), or `-d` to deny certain group addresses and allow all others (blacklist).

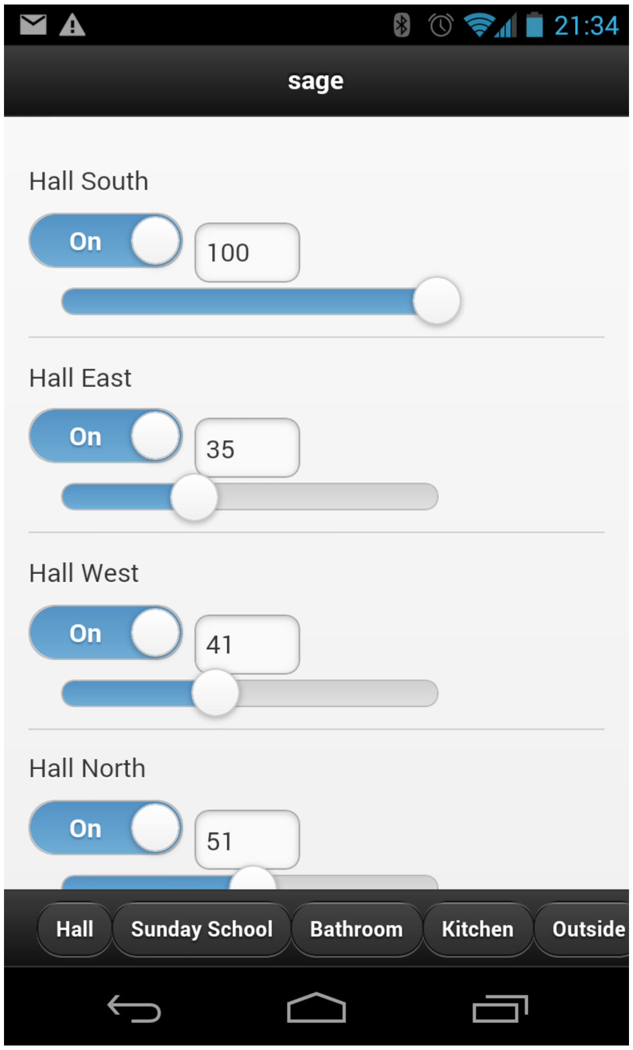
5.3.2 Other options

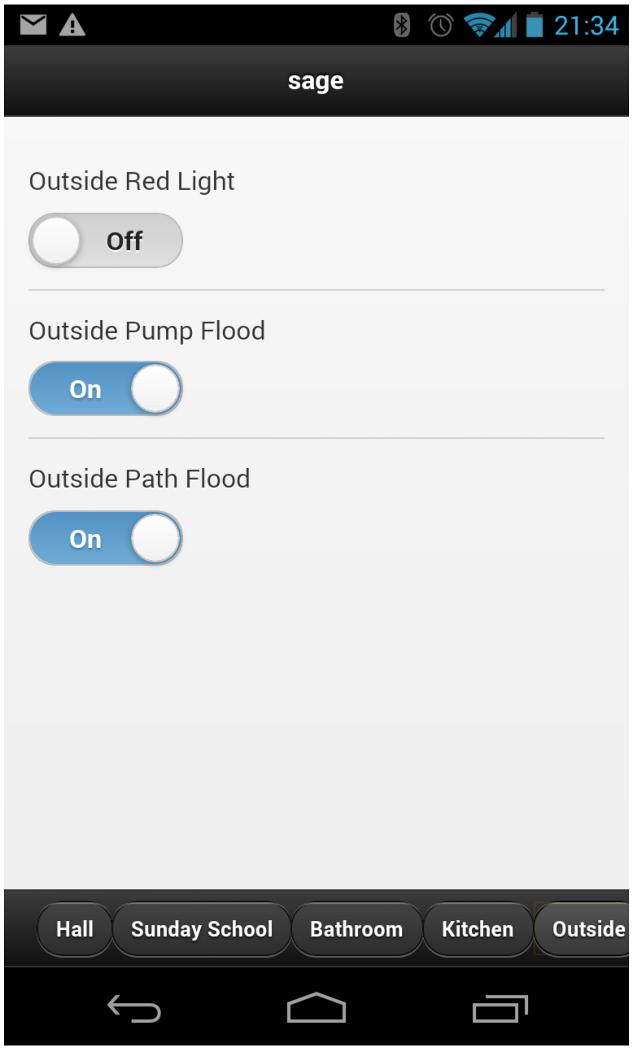
sage's daemon also supports some other options:

- `-H` and `-p` control the listening IP address and port that **sage** uses.
- `-W` turns off sage's own internal web server for static files, and makes the WebSockets resource the root of the web server. Normally, the WebSockets server is located at `/saged`.
- `-r` allows you to specify the root directory of the web server for static resources, if `-W` is not specified.
- `-a` allows specifying a comma-seperated list of group addresses to allow control of through sage, and deny all others.
- `-d` allows specifying a comma-seperated list of group addresses to deny control of through sage, and allow all others.
- `-S` connects to *cdbusd* through the D-Bus Session Bus rather than the System Bus. This is useful for unprivileged testing.
- `-P` allows you to specify a `.htpasswd`-format password file for saged to authenticate users against. This does not work in Chrome due to a bug (#123862).
- `-R` allows you to set the HTTP Basic authentication realm (when using `-P`).

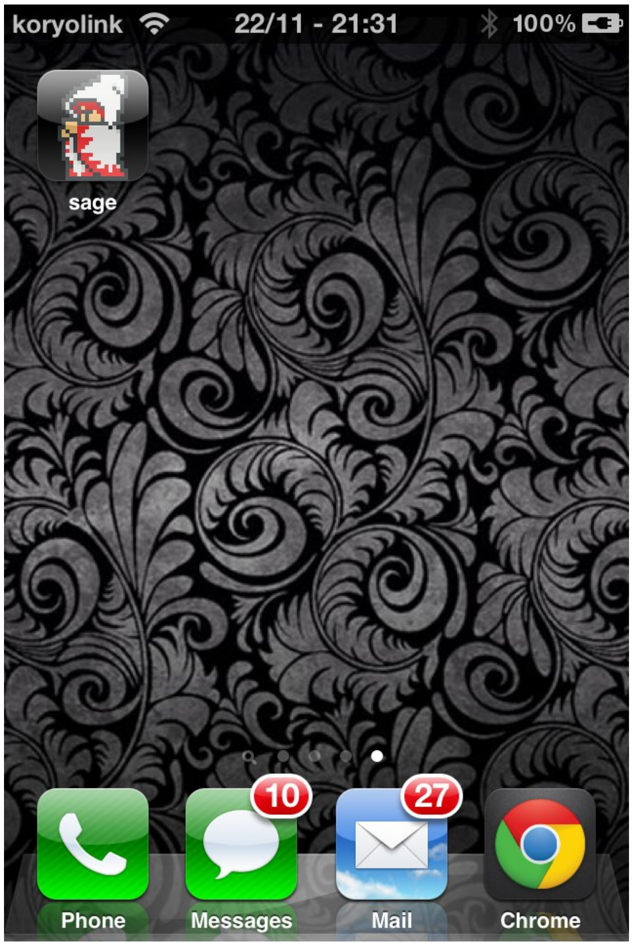
5.3.3 Screenshots

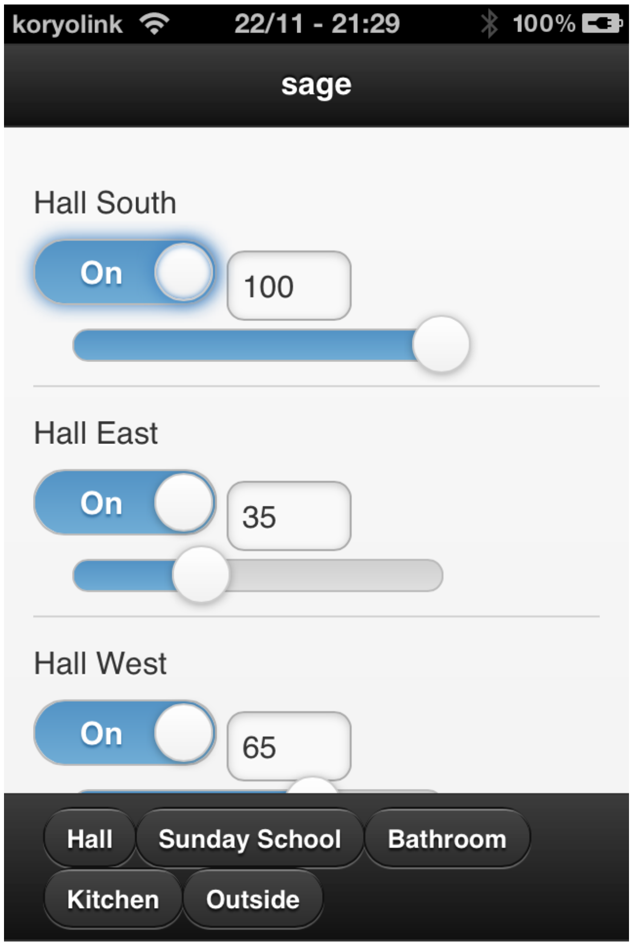
Android (Chrome)

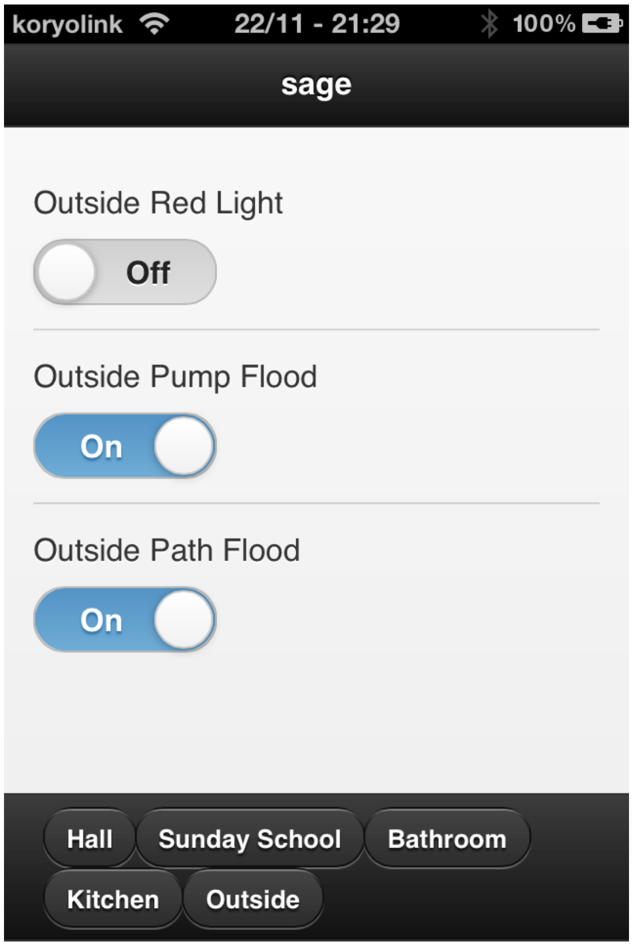



21:34
sage
Outside Red Light
Off
Outside Pump Flood
On
Outside Path Flood
On
Hall
Sunday School
Bathroom
Kitchen
Outside

iOS “Web App”


koryolink
22/11 - 21:31
100%
sage
10
27
Phone
Messages
Mail
Chrome


koryolink
22/11 - 21:29
100%
sage
Hall South
On
100
Hall East
On
35
Hall West
On
65
Hall
Sunday School
Bathroom
Kitchen
Outside



5.4 staged

staged is a scene control daemon. This isn’t finished yet.

It connects to cdbusd, and allows you to create simple scenes that react to messages sent to particular group addresses.

dump_labels utility

dump_labels parses a Toolkit backup file (CBZ) and prints out group and unit address labels into a JSON file.

It will remove all unneeded programming markup from the CBZ, and leave a skeleton of information which can be used in conjunction with the library and other applications.

6.1 Invocation

CHAPTER 7

D-Bus Client API

The recommended way to interact with libcbus is through cdbusd and it's D-Bus API.

It allows many applications, written in any language with D-Bus bindings to interact with the network with one PCI.

It abstracts away the majority of C-Bus protocol details and provides D-Bus methods and events for interacting with the network.

This document describes the interface au.id.micolous.cbus.CBusInterface.

Included in this document is examples of syntax for use with mdbus2, a command-line D-Bus testing tool. Types are given in D-Bus notation.

lighting_group_on(*group_addr*)

Turns on lights for the given group addresses.

Parameters **group_addr** (*ay*) – Group addresses to turn on lights for, up to 9.

Returns Single-byte string with code for the confirmation event.

Return type s

```
# Turn on lights for GA 1 and 2.
mdbus2 -s au.id.micolous.cbus.CBusService / au.id.micolous.cbus.CBusInterface.lighting_group_on
```

lighting_group_off(*group_addr*)

Turns off lights for the given group addresses.

Parameters **group_addr** (*ay*) – Group addresses to turn off lights for, up to 9.

Returns Single-byte string with code for the confirmation event.

Return type s

```
# Turn off lights for GA 3 and 4.
mdbus2 -s au.id.micolous.cbus.CBusService / au.id.micolous.cbus.CBusInterface.lighting_group_off
```

lighting_group_ramp(*group_addr*, *duration*, *level*)

Ramps (fades) a group address to a specified lighting level.

Note: C-Bus only supports a limited number of fade durations, in decreasing accuracy up to 17 minutes (1020 seconds). Durations longer than this will throw an error.

A duration of 0 will ramp "instantly" to the given level.

Parameters

- **group_addr** (*y*) – The group address to ramp.
- **duration** (*n*) – Duration, in seconds, that the ramp should occur over.

- **level** (*d*) – An amount between 0.0 and 1.0 indicating the brightness to set.

Returns Single-byte string with code for the confirmation event.

Return type s

```
# Fades GA 5 to 45% brightness over 12 seconds.
mdbus2 -s au.id.micolous.cbus.CBusService / au.id.micolous.cbus.CBusInterface.lighting_group_ram
```

`lighting_group_terminate_ramp`(*group_addr*)
Stops ramping a group address at the current point.

Parameters **group_addr** (*y*) – Group address to stop ramping of.

Returns Single-byte string with code for the confirmation event.

Return type s

```
# Stops ramping on GA 5
mdbus2 -s au.id.micolous.cbus.CBusService / au.id.micolous.cbus.CBusInterface.lighting_group_ter
```

CHAPTER 8

libcbus module index

8.1 cbus Package

This is the package where all C-Bus modules are defined.

8.1.1 Common functions

cbus.common defines various common helper utilities used by the library, and constants required to communicate with the C-Bus network.

The majority of the functionality shouldn't be needed by your own application, however it is used internally within the protocol encoders and decoders.

`cbus.common.`**`add_cbus_checksum`**`(`*i*`)`

Appends a C-Bus checksum to a given message.

Parameters **i** (*str*) – The C-Bus message to append a checksum to. Must not be in base16 format.

Returns The C-Bus message with the checksum appended to it.

Return type str

`cbus.common.`**`cbus_checksum`**`(`*i*, *b16=False*`)`

Calculates the checksum of a C-Bus command string.

Fun fact: C-Bus toolkit and C-Gate do not use commands with checksums.

Parameters

- **i** (*str*) – The C-Bus data to calculate the checksum of.
- **b16** (*bool*) – Indicates that the input is in base16 (network) format, and that the return should be in base16 format.

Returns The checksum value of the given input

Return type int (if b16=False), str (if b16=True)

`cbus.common.`**`duration_to_ramp_rate`**`(`*seconds*`)`

Converts a given duration into a ramp rate code.

Parameters **seconds** (*int*) – The number of seconds that the ramp is over.

Returns The ramp rate code for the duration given.

Return type int

Throws ValueError If the given duration is too long and cannot be represented.

`cbus.common.`**`get_real_cbus_checksum`**(*i*)
Calculates the supposedly correct cbus checksum for a message.

Assumes input of a base16 encoded message with the checksum ignored.

`cbus.common.`**`ramp_rate_to_duration`**(*rate*)
Converts a given ramp rate code into a duration in seconds.

Parameters **rate** (*int*) – The ramp rate code to convert.

Returns The number of seconds the ramp runs over.

Return type int

Throws KeyError If the given ramp rate code is invalid.

`cbus.common.`**`validate_cbus_checksum`**(*i*)
Verifies a C-Bus checksum from a given message.

Parameters **i** (*str*) – The C-Bus message to verify the checksum of. Must be in base16 format.

Returns True if the checksum is correct, False otherwise.

Return type bool

`cbus.common.`**`validate_ga`**(*group_addr*)
Validates a given group address to verify that it is valid.

Parameters **group_addr** (*int*) – Input group address to validate.

Returns True if the given group address is valid, False otherwise.

Return type bool

`cbus.common.`**`validate_ramp_rate`**(*duration*)
Validates the given ramp rate.

Parameters **duration** (*int*) – A duration, in seconds, to check if it is within the allowed duration constraints.

Returns True if the duration is within range, False otherwise.

Return type bool

8.1.2 Protocol

C-Bus uses it's own protocol in order to send messages over the C-Bus PHY.

This is reflected in the PC Interface protocol.

This package contains classes needed in order to operate with the protocol.

`base_packet`: Base Packet

class `cbus.protocol.base_packet.`**`BasePacket`**(*checksum=True*, *flags=None*, *destination_address_type=None*, *rc=None*, *dp=None*, *priority_class=None*)
Bases: `object`

`confirmation` = **None**

`source_address` = **None**

class `cbus.protocol.base_packet.`**`SpecialPacket`**

Bases: `cbus.protocol.base_packet.BasePacket`

`checksum` = **False**

`destination_address_type` = **None**

`dp` = **None**

`priority_class` = **None**

`rc` = **None**

class `cbus.protocol.base_packet.`**`SpecialClientPacket`**

Bases: `cbus.protocol.base_packet.SpecialPacket`

Client -> PCI communications have some special packets, which we make subclasses of SpecialClientPacket to make them entirely seperate from normal packets.

These have non-standard methods for serialisation.

class `cbus.protocol.base_packet.`**`SpecialServerPacket`**

Bases: `cbus.protocol.base_packet.SpecialPacket`

PCI -> Client has some special packets that we make subclasses of this, because they're different to regular packets.

These have non-standard serialisation methods.

`dm_packet`: Device Management Packet

class `cbus.protocol.dm_packet.`**`DeviceManagementPacket`**(*checksum=True*, *priority_class=2*, *parameter=None*, *value=None*)

Bases: `cbus.protocol.base_packet.BasePacket`

classmethod **`decode_packet`**(*data*, *checksum*, *flags*, *destination_address_type*, *rc*, *dp*, *priority_class*)

`encode`(*source_addr=None*)

`parameter` = **None**

`value` = **None**

`packet` Module

`cbus.protocol.packet.`**`decode_packet`**(*data*, *checksum=True*, *strict=True*, *server_packet=True*)

Decodes a packet from or send to the PCI.

Returns a tuple, the packet that was parsed and the remainder that was unparsed (in the case of some special commands.

If no packet was able to be parsed, the first element of the tuple will be None. However there may be some circumstances where there is still a remainder to be parsed (cancel request).

`pm_packet` Module

class `cbus.protocol.pm_packet.`**`PointToMultipointPacket`**(*checksum=True*, *priority_class=0*, *application=None*, *status_request=False*)

Bases: `cbus.protocol.base_packet.BasePacket`

application = None

classmethod decode_packet (*data*, *checksum*, *flags*, *destination_address_type*, *rc*, *dp*, *priority_class*)

encode (*source_addr=None*)

group_address = None

level_request = False

status_request = False

pp_packet Module

class cbus.protocol.pp_packet.**PointToPointPacket** (*checksum=True*, *priority_class=0*, *unit_address=0*, *bridge_address=0*, *hops=None*)

Bases: cbus.protocol.base_packet.BasePacket

classmethod decode_packet (*data*, *checksum*, *flags*, *destination_address_type*, *rc*, *dp*, *priority_class*)

encode (*source_addr=None*)

reset_packet: PCI Reset Packet

class cbus.protocol.reset_packet.**ResetPacket**

Bases: cbus.protocol.base_packet.SpecialClientPacket

encode (*source_addr=None*)

scs_packet Module

class cbus.protocol.scs_packet.**SmartConnectShortcutPacket**

Bases: cbus.protocol.base_packet.SpecialClientPacket

encode (*source_addr=None*)

pciprotocol Module

class cbus.protocol.pciprotocol.**PCIProtocol**

Bases: twisted.protocols.basic.LineReceiver

Implements a twisted protocol for communicating with a CBus PCI over serial or TCP.

clock_datetime (*when=None*)

Sends the system's local time to the CBus network.

Parameters **when** (*datetime.datetime*) – The time and date to send to the CBus network. Defaults to current local time.

connectionMade ()

Called by twisted a connection is made to the PCI. This will perform a reset of the PCI to establish the correct communications protocol.

decode_cbus_event (*line*)

Decodes a CBus event and calls an event handler appropriate to the event.

Do not override this.

Parameters **line** (*str*) – CBus event data

Returns Remaining unparsed data (str) or None on error.

Return type str or NoneType

`delimiter` **= '\r\n'**

`identify`(*unit_address*, *attribute*)
Sends an IDENTIFY command to the given unit_address.

Parameters

- **unit_address** (*int*) – Unit address to send the packet to
- **attribute** (*int*) – Attribute ID to retrieve information for. See s7.2 of Serial Interface Guide for acceptable codes.

Returns Single-byte string with code for the confirmation event.

Return type string

`lighting_group_off`(*group_addr*)
Turns off the lights for the given group_id.

Parameters **group_addr** (*int, or iterable of ints of length <= 9.*) – Group address(es) to turn the lights off for, up to 9.

Returns Single-byte string with code for the confirmation event.

Return type string

`lighting_group_on`(*group_addr*)
Turns on the lights for the given group_id.

Parameters **group_addr** (*int, or iterable of ints of length <= 9.*) – Group address(es) to turn the lights on for, up to 9.

Returns Single-byte string with code for the confirmation event.

Return type string

`lighting_group_ramp`(*group_addr*, *duration*, *level=1.0*)
Ramps (fades) a group address to a specified lighting level.

Note: CBus only supports a limited number of fade durations, in decreasing accuracy up to 17 minutes (1020 seconds). Durations longer than this will throw an error.

A duration of 0 will ramp "instantly" to the given level.

Parameters

- **group_addr** (*int*) – The group address to ramp.
- **duration** (*int*) – Duration, in seconds, that the ramp should occur over.
- **level** (*float*) – An amount between 0.0 and 1.0 indicating the brightness to set.

Returns Single-byte string with code for the confirmation event.

Return type string

`lighting_group_terminate_ramp`(*group_addr*)
Stops ramping a group address at the current point.

Parameters **group_addr** (*int*) – Group address to stop ramping of.

Returns Single-byte string with code for the confirmation event.

Return type string

`lineReceived`(*line*)
Called by LineReciever when a new line has been recieved on the PCI connection.

Do not override this.

Parameters **line** (*str*) – Raw CBus event data

`on_clock_request`(*source_addr*)
Event called when a unit requests time from the network.

Parameters **source_addr** (*int*) – Source address of the unit requesting time.

`on_clock_update`(*source_addr*, *variable*, *val*)
Event called when a unit sends time to the network.

Parameters **source_addr** (*int*) – Source address of the unit requesting time.

`on_confirmation`(*code*, *success*)
Event called when a command confirmation event was recieved.

Parameters

- **code** (*str*) – A single byte matching the command that this is a response to.
- **success** (*bool*) – True if the command was successful, False otherwise.

`on_lighting_group_off`(*source_addr*, *group_addr*)
Event called when a lighting application "off" request is recieved.

Parameters

- **source_addr** (*int*) – Source address of the unit that generated this event.
- **group_addr** (*int*) – Group address being turned off.

`on_lighting_group_on`(*source_addr*, *group_addr*)
Event called when a lighting application "on" request is recieved.

Parameters

- **source_addr** (*int*) – Source address of the unit that generated this event.
- **group_addr** (*int*) – Group address being turned on.

`on_lighting_group_ramp`(*source_addr*, *group_addr*, *duration*, *level*)
Event called when a lighting application ramp (fade) request is recieved.

Parameters

- **source_addr** (*int*) – Source address of the unit that generated this event.
- **group_addr** (*int*) – Group address being ramped.
- **duration** (*int*) – Duration, in seconds, that the ramp is occurring over.
- **level** (*float*) – Target brightness of the ramp (0.0 - 1.0).

`on_lighting_group_terminate_ramp`(*source_addr*, *group_addr*)
Event called when a lighting application "terminate ramp" request is recieved.

Parameters

- **source_addr** (*int*) – Source address of the unit that generated this event.
- **group_addr** (*int*) – Group address stopping ramping.

`on_lighting_label_text` (*source_addr*, *group_addr*, *flavour*, *language_code*, *label*)
Event called when a group address' label text is updated.

Parameters

- **source_addr** (*int*) – Source address of the unit that generated this event.
- **group_addr** (*int*) – Group address to relabel.
- **flavour** (*int*) – "Flavour" of the label to update. This is a value between 0 and 3.
- **language_code** (*int*) – Language code for the label.
- **event_bytes** (*str*) – Label text, or an empty string to delete the label.

`on_mmi` (*application*, *bytes*)
Event called when a MMI was recieved.

Parameters

- **application** (*int*) – Application that this MMI concerns.
- **bytes** (*str*) – MMI data

`on_pci_cannot_accept_data` ()
Event called whenever the PCI cannot accept the supplied data. Common reasons for this occurring:

- The checksum is incorrect.
- The buffer in the PCI is full.

Unfortunately the PCI does not tell us which requests these are associated with.

This error can occur if data is being sent to the PCI too quickly, or if the cable connecting the PCI to the computer is faulty.

While the PCI can operate at 9600 baud, this only applies to data it sends, not to data it recieves.

`on_pci_power_up` ()
If Power-up Notification (PUN) is enabled on the PCI, this event is fired.

This event may be fired multiple times in quick succession, as the PCI will send the event twice.

`on_reset` ()
Event called when the PCI has been hard reset.

`pci_reset` ()
Performs a full reset of the PCI.

`pciserverprotocol` Module

class `cbus.protocol.pciserverprotocol.`**`PCIServerProtocol`**
Bases: `twisted.protocols.basic.LineReceiver`

Implements a twisted protocol listening to CBus PCI commands over TCP or serial.

This presently only implements a subset of the protocol used by PCIProtocol.

`application_addr1` = **255**

`application_addr2` = **255**

`basic_mode` = **True**

`checksum` = **False**

`connect` = **False**

`connectionMade`()
: Called by twisted a connection is made to the PCI.

  This doesn't get fired in normal serial connections, however we'll send a power up notification (PUN).

  Serial Interface User Guide s4.3.3.4, page 33

`decode_cbus_event`(*line*)
: Decodes a CBus event and calls an event handler appropriate to the event.

  Do not override this.

  Parameters **line** (*str*) – CBus event data

  Returns Remaining unparsed data (str) or None on error.

  Return type str or NoneType

`delimiter` = **'\r\n'**

`idmon` = **False**

`lighting_group_off`(*source_addr*, *group_addr*)
: Turns off the lights for the given group_id.

  Parameters

  - **source_addr** (*int*) – Source address of the event.
  - **group_id** (*int*) – Group address to turn the lights on for.

  Returns Single-byte string with code for the confirmation event.

  Return type string

`lighting_group_on`(*source_addr*, *group_addr*)
: Turns on the lights for the given group_id.

  Parameters

  - **source_addr** (*int*) – Source address of the event.
  - **group_id** (*int*) – Group address to turn the lights on for.

  Returns Single-byte string with code for the confirmation event.

  Return type string

`lighting_group_ramp`(*source_addr*, *group_addr*, *duration*, *level=1.0*)
: Ramps (fades) a group address to a specified lighting level.

  Note: CBus only supports a limited number of fade durations, in decreasing accuracy up to 17 minutes (1020 seconds). Durations longer than this will throw an error.

  A duration of 0 will ramp "instantly" to the given level.

  Parameters

  - **source_addr** (*int*) – Source address of the event.
  - **group_id** (*int*) – The group address to ramp.
  - **duration** (*int*) – Duration, in seconds, that the ramp should occur over.
  - **level** (*float*) – An amount between 0.0 and 1.0 indicating the brightness to set.

  Returns Single-byte string with code for the confirmation event.

  Return type string

`lighting_group_terminate_ramp`(*source_addr*, *group_addr*)
Stops ramping a group address at the current point.

Parameters

- **source_addr** (*int*) – Source address of the event.
- **group_addr** (*int*) – Group address to stop ramping of.

Returns Single-byte string with code for the confirmation event.

Return type string

`lineReceived`(*line*)
Called by LineReciever when a new line has been recieved on the PCI connection.

Do not override this.

Parameters **line** (*str*) – Raw CBus event data

`local_echo` = **True**

`monitor` = **False**

`on_clock_request`()
Event called when a clock application “request time” is recieved.

`on_clock_update`(*variable*, *val*)
Event called when a clock application “update time” is recieved.

Parameters

- **variable** (*datetime.date or datetime.time*) – Clock variable to update.
- **val** – Clock value

`on_lighting_group_off`(*group_addr*)
Event called when a lighting application “off” request is recieved.

Parameters **group_addr** (*int*) – Group address being turned off.

`on_lighting_group_on`(*group_addr*)
Event called when a lighting application “on” request is recieved.

Parameters **group_addr** (*int*) – Group address being turned on.

`on_lighting_group_ramp`(*group_addr*, *duration*, *level*)
Event called when a lighting application ramp (fade) request is recieved.

Parameters

- **group_addr** (*int*) – Group address being ramped.
- **duration** (*int*) – Duration, in seconds, that the ramp is occurring over.
- **level** (*float*) – Target brightness of the ramp (0.0 - 1.0).

`on_lighting_group_terminate_ramp`(*group_addr*)
Event called when a lighting application “terminate ramp” request is recieved.

Parameters **group_addr** (*int*) – Group address ramp being terminated.

`on_reset`()
Event called when the PCI has been hard reset.

`send_confirmation`(*code*, *ok=True*)

`send_error`()

Applications

Running ontop of the C-Bus protocols are applications.

This package provides encoders and decoders for application-level messages on the C-Bus network.

Application messages inside of C-Bus packets are called “Specific Application Language”, or SALs for short. A packet may contain many SALs for a single application, up to the MTU of the C-Bus network.

Clock and Timekeeping Application

The Clock and Timekeeping Application is used to provide access to date and time information to CBus units.

This is used for example in conjunction with programmable units that act differently depending on the time or date, and with touchscreen units that may display the time on their screens.

Please refer to this document in conjunction with the Clock and Timekeeping Application Guide published by Clipsal.

class `cbus.protocol.application.clock.`**`ClockApplication`**

Bases: `object`

This class is called in the cbus.protocol.applications.APPLICATIONS dict in order to describe how to decode clock and timekeeping application events recieved from the network.

Do not call this class directly.

classmethod **`decode_sal`**(*data*, *packet*)

Decodes a clock and timekeeping application packet and returns it’s SAL(s).

class `cbus.protocol.application.clock.`**`ClockSAL`**(*packet=None*)

Bases: `object`

Base type for clock and timekeeping application SALs.

This should not be called directly by your code!

Use one of the subclasses of cbus.protocol.clock.ClockSAL instead.

classmethod **`decode`**(*data*, *packet*)

Decodes a clock broadcast application packet and returns it’s SAL(s).

Parameters

- **data** (*str*) – SAL data to be parsed.
- **packet** (*cbus.protocol.base_packet.BasePacket*) – The packet that this data is associated with.

Returns The SAL messages contained within the given data.

Return type list of cbus.protocol.application.clock.ClockSAL

`encode`()

Encodes the SAL into a format for sending over the C-Bus network.

class `cbus.protocol.application.clock.`**`ClockUpdateSAL`**(*packet*, *variable*, *val*)

Bases: `cbus.protocol.application.clock.ClockSAL`

Clock update event SAL.

Informs the network of the current time.

Creates a new SAL Clock update message.

Parameters

- **packet** (*cbus.protocol.base_packet.BasePacket*) – The packet that this SAL is to be included in.
- **variable** (*int*) – The variable being updated.
- **val** (*date or time*) – The value of that variable. Dates are represented in native date format, and times are represented in native time format.

classmethod `decode` (*data*, *packet*, *command_code*)
Do not call this method directly – use ClockSAL.decode

`encode` ()

`is_date`

`is_time`

class `cbus.protocol.application.clock.ClockRequestSAL` (*packet*)
Bases: `cbus.protocol.application.clock.ClockSAL`

Clock request event SAL.

Requests network time.

Creates a new SAL Clock request message.

Parameters **packet** (*cbus.protocol.base_packet.BasePacket*) – The packet that this SAL is to be included in.

classmethod `decode` (*data*, *packet*, *command_code*)
Do not call this method directly – use ClockSAL.decode

`encode` ()

Enable Control Application

The Enable Control Application is used to set network variables on CBus units.

This can change the behaviour of certain elements of the network, or allow some reprogramming of devices.

Please refer to this document in conjunction with the Enable Control Application Guide published by Clipsal.

class `cbus.protocol.application.enable.EnableApplication`
Bases: `object`

This class is called in the cbus.protocol.applications.APPLICATIONS dict in order to describe how to decode enable broadcast application events recieved from the network.

Do not call this class directly.

classmethod `decode_sal` (*data*, *packet*)
Decodes a enable broadcast application packet and returns it's SAL(s).

class `cbus.protocol.application.enable.EnableSAL` (*packet=None*)
Bases: `object`

Base type for enable control application SALs.

This should not be called directly by your code!

Use one of the subclasses of cbus.protocol.enable.EnableSAL instead.

classmethod **`decode`**(*data*, *packet*)
Decodes a enable control application packet and returns it's SAL(s).

Parameters

- **data** (*str*) – SAL data to be parsed.
- **packet** (*cbus.protocol.base_packet.BasePacket*) – The packet that this data is associated with.

Returns The SAL messages contained within the given data.

Return type list of cbus.protocol.application.enable.EnableSAL

`encode`()
Encodes the SAL into a format for sending over the C-Bus network.

class `cbus.protocol.application.enable.`**`EnableSetNetworkVariableSAL`**(*packet*, *variable*, *value*)
Bases: `cbus.protocol.application.enable.EnableSAL`

Enable control Set Network Variable SAL.

Sets a network variable.

Creates a new SAL Enable Control Set Network Variable

Parameters

- **packet** (*cbus.protocol.base_packet.BasePacket*) – The packet that this SAL is to be included in.
- **variable** (*int*) – The variable ID being changed
- **value** (*int*) – The value of the network variable

classmethod **`decode`**(*data*, *packet*, *command_code*)
Do not call this method directly – use EnableSAL.decode

`encode`()

Lighting Application

The lighting application is the most commonly used application on the C-Bus network.

It is used for turning lights on and off, and setting lights to a particular brightness.

Sometimes the lighting application is used to control other, non-lighting loads, such as exhaust fans.

Please refer to this document in conjunction with the Lighting Application Guide published by Clipsal.

class `cbus.protocol.application.lighting.`**`LightingApplication`**
Bases: `object`

This class is called in the cbus.protocol.applications.APPLICATIONS dict in order to describe how to decode lighting application events recieved from the network.

Do not call this class directly.

classmethod **`decode_sal`**(*data*, *packet*)
Decodes a lighting application packet and returns it's SAL(s).

class `cbus.protocol.application.lighting.`**`LightingSAL`**(*packet=None*, *group_address=None*)
Bases: `object`

Base type for lighting application SALs.

This should not be called directly by your code!

Use one of the subclasses of cbus.protocol.lighting.LightingSAL instead.

classmethod **`decode`**(*data*, *packet*)
Decodes a lighting application packet and returns it's SAL(s).

Parameters

- **data** (*str*) – SAL data to be parsed.
- **packet** (*cbus.protocol.base_packet.BasePacket*) – The packet that this data is associated with.

Returns The SAL messages contained within the given data.

Return type list of cbus.protocol.application.lighting.LightingSAL

`encode`()
Encodes the SAL into a format for sending over the C-Bus network.

class `cbus.protocol.application.lighting.`**`LightingRampSAL`**(*packet*, *group_address*, *duration*, *level*)
Bases: `cbus.protocol.application.lighting.LightingSAL`

Lighting Ramp (fade) event SAL

Instructs the given group address to fade to a lighting level (brightness) over a given duration.

Creates a new SAL Lighting Ramp message.

Parameters

- **packet** (*cbus.protocol.base_packet.BasePacket*) – The packet that this SAL is to be included in.
- **group_address** (*int*) – The group address to ramp.
- **duration** (*int*) – The duration to ramp over, in seconds.
- **level** (*float*) – The level to ramp to, with 0.0 indicating off, and 1.0 indicating full brightness.

classmethod **`decode`**(*data*, *packet*, *command_code*, *group_address*)
Do not call this method directly – use LightingSAL.decode

`encode`()

class `cbus.protocol.application.lighting.`**`LightingOnSAL`**(*packet*, *group_address*)
Bases: `cbus.protocol.application.lighting.LightingSAL`

Lighting on event SAL

Instructs a given group address to turn it's load on.

Creates a new SAL Lighting On message.

Parameters

- **packet** (*cbus.protocol.base_packet.BasePacket*) – The packet that this SAL is to be included in.
- **group_address** (*int*) – The group address to turn on.

classmethod **`decode`**(*data*, *packet*, *command_code*, *group_address*)
Do not call this method directly – use LightingSAL.decode

`encode`()

class `cbus.protocol.application.lighting.`**`LightingOffSAL`**(*packet*, *group_address*)

Bases: `cbus.protocol.application.lighting.LightingSAL`

Lighting off event SAL

Instructs a given group address to turn it's load off.

Creates a new SAL Lighting Off message.

Parameters

- **packet** (*cbus.protocol.base_packet.BasePacket*) – The packet that this SAL is to be included in.
- **group_address** (*int*) – The group address to turn off.

classmethod **`decode`**(*data*, *packet*, *command_code*, *group_address*)

Do not call this method directly – use LightingSAL.decode

`encode`()

class `cbus.protocol.application.lighting.`**`LightingTerminateRampSAL`**(*packet*, *group_address*)

Bases: `cbus.protocol.application.lighting.LightingSAL`

Lighting terminate ramp event SAL

Instructs the given group address to discontinue any ramp operations in progress, and use the brightness that they are currently at.

Creates a new SAL Lighting Terminate Ramp message.

Parameters

- **packet** (*cbus.protocol.base_packet.BasePacket*) – The packet that this SAL is to be included in.
- **group_address** (*int*) – The group address to stop ramping.

classmethod **`decode`**(*data*, *packet*, *command_code*, *group_address*)

Do not call this method directly – use LightingSAL.decode

`encode`()

Temperature Broadcast Application

The Temperature Broadcast application is used to notify units on the C-Bus network of changes in temperature. It is used to allow temperature control systems to react to changes in environmental conditions.

This has been replaced by the Measurement application (not get implemented by `libcbus`).

Please refer to this document in conjunction with the Temperature Broadcast Application Guide published by Clipsal.

class `cbus.protocol.application.temperature.`**`TemperatureApplication`**

Bases: `object`

This class is called in the cbus.protocol.applications.APPLICATIONS dict in order to describe how to decode temperature broadcast application events recieved from the network.

Do not call this class directly.

classmethod **`decode_sal`**(*data*, *packet*)

Decodes a temperature broadcast application packet and returns it's SAL(s).

class `cbus.protocol.application.temperature.`**`TemperatureSAL`**(*packet=None*, *group_address=None*)

Bases: `object`

Base type for temperature broadcast application SALs.

This should not be called directly by your code!

Use one of the subclasses of cbus.protocol.temperature.TemperatureSAL instead.

classmethod **`decode`**(*data*, *packet*)

Decodes a temperature broadcast application packet and returns it's SAL(s).

Parameters

- **data** (*str*) – SAL data to be parsed.
- **packet** (*cbus.protocol.base_packet.BasePacket*) – The packet that this data is associated with.

Returns The SAL messages contained within the given data.

Return type list of cbus.protocol.application.temperature.TemperatureSAL

`encode`()

Encodes the SAL into a format for sending over the C-Bus network.

class `cbus.protocol.application.temperature.`**`TemperatureBroadcastSAL`**(*packet*, *group_address*, *tempera-ture*)

Bases: `cbus.protocol.application.temperature.TemperatureSAL`

Temperature broadcast event SAL.

Informs the network of the current temperature being sensed at a location.

Creates a new SAL Temperature Broadcast message.

Parameters

- **packet** (*cbus.protocol.base_packet.BasePacket*) – The packet that this SAL is to be included in.
- **group_address** (*int*) – The group address that is reporting the temperature.
- **temperature** (*float*) – The temperature, in degrees celcius, between 0.0 and 63.75.

classmethod **`decode`**(*data*, *packet*, *command_code*, *group_address*)

Do not call this method directly – use TemperatureSAL.decode

`encode`()

8.1.3 toolkit Package

`cbz` Module

cbus/toolkit/cbz.py Library for reading CBus Toolkit CBZ files with lxml.

This library is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library. If not, see <http://www.gnu.org/licenses/>.

class `cbus.toolkit.cbz.`**`CBZ`**(*fh*)

Bases: `object`

Opens the file as a CBZ.

fh can either be a string poining to a file name or a file-like object.

exception `cbus.toolkit.cbz.`**`CBZException`**

Bases: `exceptions.Exception`

dump_labels utility

dump_labels parses a Toolkit backup file (CBZ) and prints out group and unit address labels into a JSON file.

It will remove all unneeded programming markup from the CBZ, and leave a skeleton of information which can be used in conjunction with the library and other applications.

Invocation

CHAPTER 9

Indices and tables

- *search*

Python Module Index

C

cbus.common, 33
cbus.protocol.application.clock, 42
cbus.protocol.application.enable, 43
cbus.protocol.application.lighting, 44
cbus.protocol.application.temperature, 46
cbus.protocol.base_packet, 34
cbus.protocol.dm_packet, 35
cbus.protocol.packet, 35
cbus.protocol.pciprotocol, 36
cbus.protocol.pciserverprotocol, 39
cbus.protocol.pm_packet, 35
cbus.protocol.pp_packet, 36
cbus.protocol.reset_packet, 36
cbus.protocol.scs_packet, 36
cbus.toolkit.cbz, 47

Index

A

add_cbus_checksum() (in module cbus.common), 33
application (cbus.protocol.pm_packet.PointToMultipointPacket attribute), 35
application_addr1 (cbus.protocol.pciserverprotocol.PCIServerProtocol attribute), 39
application_addr2 (cbus.protocol.pciserverprotocol.PCIServerProtocol attribute), 39

B

BasePacket (class in cbus.protocol.base_packet), 34
basic_mode (cbus.protocol.pciserverprotocol.PCIServerProtocol attribute), 39

C

cbus.common (module), 33
cbus.protocol.application.clock (module), 42
cbus.protocol.application.enable (module), 43
cbus.protocol.application.lighting (module), 44
cbus.protocol.application.temperature (module), 46
cbus.protocol.base_packet (module), 34
cbus.protocol.dm_packet (module), 35
cbus.protocol.packet (module), 35
cbus.protocol.pciprotocol (module), 36
cbus.protocol.pciserverprotocol (module), 39
cbus.protocol.pm_packet (module), 35
cbus.protocol.pp_packet (module), 36
cbus.protocol.reset_packet (module), 36
cbus.protocol.scs_packet (module), 36
cbus.toolkit.cbz (module), 47
cbus_checksum() (in module cbus.common), 33
CBZ (class in cbus.toolkit.cbz), 48
CBZException, 48
checksum (cbus.protocol.base_packet.SpecialPacket attribute), 35
checksum (cbus.protocol.pciserverprotocol.PCIServerProtocol attribute), 39
clock_datetime() (cbus.protocol.pciprotocol.PCIProtocol method), 36
ClockApplication (class in cbus.protocol.application.clock), 42
ClockRequestSAL (class in cbus.protocol.application.clock), 43
ClockSAL (class in cbus.protocol.application.clock), 42
ClockUpdateSAL (class in cbus.protocol.application.clock), 42
confirmation (cbus.protocol.base_packet.BasePacket attribute), 34
connect (cbus.protocol.pciserverprotocol.PCIServerProtocol attribute), 39
connectionMade() (cbus.protocol.pciprotocol.PCIProtocol method), 36
connectionMade() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 39

D

decode() (cbus.protocol.application.clock.ClockRequestSAL class method), 43
decode() (cbus.protocol.application.clock.ClockSAL class method), 42
decode() (cbus.protocol.application.clock.ClockUpdateSAL class method), 43
decode() (cbus.protocol.application.enable.EnableSAL class method), 43
decode() (cbus.protocol.application.enable.EnableSetNetworkVariableSAL class method), 44
decode() (cbus.protocol.application.lighting.LightingOffSAL class method), 46
decode() (cbus.protocol.application.lighting.LightingOnSAL class method), 45
decode() (cbus.protocol.application.lighting.LightingRampSAL class method), 45
decode() (cbus.protocol.application.lighting.LightingSAL class method), 45
decode() (cbus.protocol.application.lighting.LightingTerminateRampSAL class method), 46
decode() (cbus.protocol.application.temperature.TemperatureBroadcastSAL class method), 47
decode() (cbus.protocol.application.temperature.TemperatureSAL class method), 47

decode_cbus_event() (cbus.protocol.pciprotocol.PCIProtocol method), 36
decode_cbus_event() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 40
decode_packet() (cbus.protocol.dm_packet.DeviceManagementPacket class method), 35
decode_packet() (cbus.protocol.pm_packet.PointToMultipointPacket class method), 36
decode_packet() (cbus.protocol.pp_packet.PointToPointPacket class method), 36
decode_packet() (in module cbus.protocol.packet), 35
decode_sal() (cbus.protocol.application.clock.ClockApplication class method), 42
decode_sal() (cbus.protocol.application.enable.EnableApplication class method), 43
decode_sal() (cbus.protocol.application.lighting.LightingApplication class method), 44
decode_sal() (cbus.protocol.application.temperature.TemperatureApplication class method), 46
delimiter (cbus.protocol.pciprotocol.PCIProtocol attribute), 37
delimiter (cbus.protocol.pciserverprotocol.PCIServerProtocol attribute), 40
destination_address_type (cbus.protocol.base_packet.SpecialPacket attribute), 35
DeviceManagementPacket (class in cbus.protocol.dm_packet), 35
dp (cbus.protocol.base_packet.SpecialPacket attribute), 35
duration_to_ramp_rate() (in module cbus.common), 33

E

EnableApplication (class in cbus.protocol.application.enable), 43
EnableSAL (class in cbus.protocol.application.enable), 43
EnableSetNetworkVariableSAL (class in cbus.protocol.application.enable), 44
encode() (cbus.protocol.application.clock.ClockRequestSAL method), 43
encode() (cbus.protocol.application.clock.ClockSAL method), 42
encode() (cbus.protocol.application.clock.ClockUpdateSAL method), 43
encode() (cbus.protocol.application.enable.EnableSAL method), 44
encode() (cbus.protocol.application.enable.EnableSetNetworkVariableSAL method), 44
encode() (cbus.protocol.application.lighting.LightingOffSAL method), 46
encode() (cbus.protocol.application.lighting.LightingOnSAL method), 45
encode() (cbus.protocol.application.lighting.LightingRampSAL method), 45
encode() (cbus.protocol.application.lighting.LightingSAL method), 45
encode() (cbus.protocol.application.lighting.LightingTerminateRampSAL method), 46
encode() (cbus.protocol.application.temperature.TemperatureBroadcastSAL method), 47
encode() (cbus.protocol.application.temperature.TemperatureSAL method), 47
encode() (cbus.protocol.dm_packet.DeviceManagementPacket method), 35
encode() (cbus.protocol.pm_packet.PointToMultipointPacket method), 36
encode() (cbus.protocol.pp_packet.PointToPointPacket method), 36
encode() (cbus.protocol.reset_packet.ResetPacket method), 36
encode() (cbus.protocol.scs_packet.SmartConnectShortcutPacket method), 36

G

get_real_cbus_checksum() (in module cbus.common), 34
group_address (cbus.protocol.pm_packet.PointToMultipointPacket attribute), 36

I

identify() (cbus.protocol.pciprotocol.PCIProtocol method), 37
idmon (cbus.protocol.pciserverprotocol.PCIServerProtocol attribute), 40
is_date (cbus.protocol.application.clock.ClockUpdateSAL attribute), 43
is_time (cbus.protocol.application.clock.ClockUpdateSAL attribute), 43

L

level_request (cbus.protocol.pm_packet.PointToMultipointPacket attribute), 36
lighting_group_off(), 31
lighting_group_off() (cbus.protocol.pciprotocol.PCIProtocol method), 37
lighting_group_off() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 40
lighting_group_on(), 31
lighting_group_on() (cbus.protocol.pciprotocol.PCIProtocol method), 37
lighting_group_on() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 40
lighting_group_ramp(), 31
lighting_group_ramp() (cbus.protocol.pciprotocol.PCIProtocol method), 37
lighting_group_ramp() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 40

lighting_group_terminate_ramp(), 32
lighting_group_terminate_ramp() (cbus.protocol.pciprotocol.PCIProtocol method), 37
lighting_group_terminate_ramp() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 40
LightingApplication (class in cbus.protocol.application.lighting), 44
LightingOffSAL (class in cbus.protocol.application.lighting), 45
LightingOnSAL (class in cbus.protocol.application.lighting), 45
LightingRampSAL (class in cbus.protocol.application.lighting), 45
LightingSAL (class in cbus.protocol.application.lighting), 44
LightingTerminateRampSAL (class in cbus.protocol.application.lighting), 46
lineReceived() (cbus.protocol.pciprotocol.PCIProtocol method), 38
lineReceived() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 41
local_echo (cbus.protocol.pciserverprotocol.PCIServerProtocol attribute), 41

M

monitor (cbus.protocol.pciserverprotocol.PCIServerProtocol attribute), 41

O

on_clock_request() (cbus.protocol.pciprotocol.PCIProtocol method), 38
on_clock_request() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 41
on_clock_update() (cbus.protocol.pciprotocol.PCIProtocol method), 38
on_clock_update() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 41
on_confirmation() (cbus.protocol.pciprotocol.PCIProtocol method), 38
on_lighting_group_off() (cbus.protocol.pciprotocol.PCIProtocol method), 38
on_lighting_group_off() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 41
on_lighting_group_on() (cbus.protocol.pciprotocol.PCIProtocol method), 38
on_lighting_group_on() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 41
on_lighting_group_ramp() (cbus.protocol.pciprotocol.PCIProtocol method), 38
on_lighting_group_ramp() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 41
on_lighting_group_terminate_ramp() (cbus.protocol.pciprotocol.PCIProtocol method), 38
on_lighting_group_terminate_ramp() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 41
on_lighting_label_text() (cbus.protocol.pciprotocol.PCIProtocol method), 38
on_mmi() (cbus.protocol.pciprotocol.PCIProtocol method), 39
on_pci_cannot_accept_data() (cbus.protocol.pciprotocol.PCIProtocol method), 39
on_pci_power_up() (cbus.protocol.pciprotocol.PCIProtocol method), 39
on_reset() (cbus.protocol.pciprotocol.PCIProtocol method), 39
on_reset() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 41

P

parameter (cbus.protocol.dm_packet.DeviceManagementPacket attribute), 35
pci_reset() (cbus.protocol.pciprotocol.PCIProtocol method), 39
PCIProtocol (class in cbus.protocol.pciprotocol), 36
PCIServerProtocol (class in cbus.protocol.pciserverprotocol), 39
PointToMultipointPacket (class in cbus.protocol.pm_packet), 35
PointToPointPacket (class in cbus.protocol.pp_packet), 36
priority_class (cbus.protocol.base_packet.SpecialPacket attribute), 35

R

ramp_rate_to_duration() (in module cbus.common), 34
rc (cbus.protocol.base_packet.SpecialPacket attribute), 35
ResetPacket (class in cbus.protocol.reset_packet), 36

S

send_confirmation() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 41
send_error() (cbus.protocol.pciserverprotocol.PCIServerProtocol method), 41
SmartConnectShortcutPacket (class in cbus.protocol.scs_packet), 36
source_address (cbus.protocol.base_packet.BasePacket attribute), 34
SpecialClientPacket (class in cbus.protocol.base_packet), 35
SpecialPacket (class in cbus.protocol.base_packet), 34

SpecialServerPacket (class in cbus.protocol.base_packet), 35
status_request (cbus.protocol.pm_packet.PointToMultipointPacket attribute), 36

T

TemperatureApplication (class in cbus.protocol.application.temperature), 46
TemperatureBroadcastSAL (class in cbus.protocol.application.temperature), 47
TemperatureSAL (class in cbus.protocol.application.temperature), 46

V

validate_cbus_checksum() (in module cbus.common), 34
validate_ga() (in module cbus.common), 34
validate_ramp_rate() (in module cbus.common), 34
value (cbus.protocol.dm_packet.DeviceManagementPacket attribute), 35