## REVISION HISTORY

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>DATE</th>
<th>DESCRIPTION</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>2012-02-01</td>
<td>Add usage notes on LD_LIBRARY_PATH, libeaton_dump_all and libeaton_debug_level.</td>
<td>A. Quette</td>
</tr>
<tr>
<td>0.3</td>
<td>2012-01-17</td>
<td>Complete the &quot;Support information&quot; with an URL, and mention Beta publication.</td>
<td>A. Quette</td>
</tr>
<tr>
<td>0.2</td>
<td>2012-01-09</td>
<td>Clarify wording, complete legal information and add the &quot;Support information&quot; paragraph.</td>
<td>A. Quette</td>
</tr>
<tr>
<td>0.1</td>
<td>2011-11-21</td>
<td>First release of Eaton Software Development Kit.</td>
<td>F. Bohe, A. Quette</td>
</tr>
</tbody>
</table>
# Contents

1. Introduction .............................. 1

2. Legal limitations .......................... 1

3. Support information ....................... 2

4. Eaton SDK ................................ 2
   4.1 Limitation ................................. 2
   4.2 How does it work? ......................... 2
   4.3 Building your application with the SDK ... 3
      4.3.1 Linux build ............................. 4
      4.3.2 Windows build ........................... 4
   4.4 Building SNMP application ................. 4
      4.4.1 Linux build ............................. 5
      4.4.2 Windows build ........................... 5
      With MSVC .................................. 5
      With MinGW ................................. 5
   4.5 Building XML/HTTP application ............. 5
      4.5.1 Linux build ............................. 6
      4.5.2 Windows build ........................... 6
      With MSVC .................................. 6
      With MinGW ................................. 6
   4.6 Building USB HID application ............... 7
      4.6.1 Linux build ............................. 7
      4.6.2 Windows build ........................... 7
      With MSVC .................................. 7
      With MinGW ................................. 8
   4.7 Building USB XCP application .............. 8
      4.7.1 Linux build ............................. 8
      4.7.2 Windows build ........................... 9
      With MSVC .................................. 9
      With MinGW ................................. 9
   4.8 Building serial XCP application .......... 9
      4.8.1 Linux build ............................. 10
      4.8.2 Windows build ........................... 10
      With MSVC .................................. 10
      With MinGW ................................. 10
   4.9 Building SHUT application ................. 11
4.9.1 Linux build ......................................................... 11
4.9.2 Windows build ............................................... 11
   With MSVC ..................................................... 11
   With MinGW ................................................... 12

5 Using the complete NUT framework .................................. 12

6 Using part of the NUT framework ................................ 14

7 Protocol re-implementation ......................................... 14
   7.1 USB and shut .................................................. 14
   7.2 XCP ............................................................ 17
   7.3 SNMP .......................................................... 17

A NUT command and variable naming scheme ....................... 17
   A.1 Variables ....................................................... 17
      A.1.1 device: General unit information ...................... 17
      A.1.2 ups: General unit information ......................... 18
      A.1.3 input: Incoming line/power information .......... 19
      A.1.4 output: Outgoing power/inverter information .... 19
      A.1.5 Three-phase additions ................................ 20
         Phase Count Determination ................................ 20
         DOMAINs .................................................... 20
         Specification (SPEC) ....................................... 20
         CONTEXT .................................................... 20
         Valid CONTEXTs ............................................ 20
         Valid SPECs ................................................ 20
      A.1.6 EXAMPLES .................................................. 21
      A.1.7 battery: Any battery details ......................... 21
      A.1.8 ambient: Conditions from external probe equipment 22
      A.1.9 outlet: Smart outlet management ..................... 22
      A.1.10 driver: Internal driver information ............... 23
      A.1.11 server: Internal server information ............... 23
   A.2 Instant commands ............................................. 23
1 Introduction

The Eaton Software Development Kit ("SDK") allows your applications to connect to certain Eaton devices and retrieve certain internal data. Eaton SDK extensively uses the NUT - Network UPS Tools Opensource framework (supported by Eaton), to do so.

Warning
This version of Eaton SDK is tagged Beta, and is currently only intended for evaluation purpose.

This document provides the documentation for using the Eaton SDK libraries integration, on Linux and Microsoft Windows platforms, in chapter 3.

In case using the SDK libraries is not possible in your project (refer e.g. to the Legal limitations and SDK limitation), you will find alternatives methods on using the NUT framework with your application in the following chapters:

- Chapter 4 describes how to interact with the full NUT framework,
- Chapter 5 explains how to use directly NUT drivers,
- Chapter 6 contains some hints on how to re-implement the device specific protocol.

2 Legal limitations

The Network UPS Tools Opensource framework is a GNU General Public License (GPL) software that provides support for many different UPSs and PDUs. We expressly make the license terms of the GNU GPL applicable to this SDK solution, and by using the SDK you also accept to abide by these license terms; please see below.

GPL software have legal and technical implications, and generally can not be used with proprietary software. You should consider these very seriously before making a choice and using the present SDK.

Warning
Eaton recommends to use this SDK or the other documented approaches only as described in this document or the attached documentation. Eaton does not warrant completeness, correctness and permanent availability of the data and information retrieved by using the SDK (unless required by applicable law). Eaton does not assume any liability (unless required by applicable law), for damages resulting from the use of the SDK, or data or information retrieved by using SDK.

Here are some references that may help you understand these limitations and constraints: GPL FAQ.

Please also respect the following notice, which is compulsory for using the GPL:

Copyright © 2011 - 2012 Eaton Industries France - Frédéric Bohe, Arnaud Quette

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

This program 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 General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/.
3 Support information

In case you become aware of any malfunctions of this solution, we kindly ask you to inform us using the following support form: http://pqtools.eaton.com/form/support/index_soft.php?lang=eng.

In this case, please use Eaton Open Source as Software used, and mention Eaton SDK in your message, with the details of your issue.

We recommend to only obtain this software from original Eaton sources, and to check for updates regularly, so that you can enjoy the latest functionalities.

4 Eaton SDK

Note before using the Eaton SDK, please review section 2 regarding legal limitations. If in doubt, chapter 2 references several documents to guide you through the licensing. chapter 2.

4.1 Limitation

Using the Eaton SDK library approach, an application can only manage one device! If you need to manage more than one device, please contact the Eaton Opensource team, or refer to the other possible approaches detailed in other chapters of this user’s guide as shown in the Introduction chapter.

Please, also note that Eaton SDK is not yet Multi thread safe. Thus, you need to pay attention to have at most one thread that call Eaton SDK functions.

4.2 How does it work?

The SDK consists of several libraries allowing access to Eaton’s UPS/PDU in several ways. Your application may link to one of those libraries to access a UPS/PDU.

To use this library, your application must initialize it first by calling libeaton_init function.

```c
void libeaton_init(char * device)
```

The device parameter depends on the library you are using and is discussed in the relevant in following chapters.

Now you can call function to read or write variables of your device. You can also request your device to execute instant commands. Each of these functions gets the variable name or command name. A list of all existing variables and commands can be found in appendix A. Note that this gives all existing variables and commands, not all available variables and commands, which depend on the device type.

To have the list of available commands and variables of your device you can use the libeaton_dump_all function.

To read a variable, use the libeaton_read function in conjunction with the libeaton_update function:

```c
void libeaton_update(void);
const char *libeaton_read(const char *varname)
```

libeaton_read returns a pointer to a string containing the value of the variable or NULL if an error occurred. It points to a string managed by the library you must not try to free it. libeaton_read returns data which were gathered in the last call to libeaton_update. It’s up to your application to call libeaton_update in order to have fresh data (but remember this may be time consuming, especially for serial connections). You must call libeaton_update at least once before calling libeaton_read or it will fail.

To write a variable, use the libeaton_write function:
int libeaton_write (const char *varname, const char *val)

libeaton_write returns 0 if the variable was successfully written.

To call an instant command, use the libeaton_command function:

int libeaton_command(const char *cmdname, const char *extradata)

libeaton_command return 0 if the command was successfully issued.

To list available commands and variables:

char * libeaton_dump_all()

libeaton_dump_all returns a pointer to a buffer containing the available commands and variables for the current device. This list contains several strings in the form :

type<tab>command or variable name<tab>value

• type is one of VAR_RO, VAR_RW or CMD
• value is only available for variable

This is a sample output:

VAR_RO ups.mfr Eaton
VAR_RW outlet.1.delay.shutdown 10
CMD outlet.1.shutdown.return

Once done with your device you should call libeaton_free function to free internal data.

void libeaton_free()

This is for debugging and support purpose :

int libeaton_debug_level

You can set this variable to a value from 1 to 5 for more or less verbosity of the library (0 is no debug output). If you ask for support you will probably be asked to set this variable to a value of 3 or above.

4.3 Building your application with the SDK

For all environments and all type of connections, you need all of the following files available in your include path: attribute.h, common.h, config.h, libeaton.h, proto.h and timehead.h. You may need to add the -I<path-to-include> option to the build command lines provided in this document to point to those header files.

You may have to link with libeaton shared libraries. To do so the relevant libeaton shared library must be available for your compiler. So you have to add the correct "-L<path-to-library>" to the build command lines provided in this document to point to those libraries.

Depending the type of connection, you may need additional external libraries (net-snmp, neon...) which are described in the relevant chapter.
4.3.1 Linux build

On Linux, building with the SDK has been tested using GCC.
For static link you will need the corresponding ".a" file.
For dynamic link you will need the corresponding ".so" file for compilation and ".so", ".so.0" and ".so.0.0.0" in you libs path for execution.

For dynamic link, you will also need to point the system linker at Eaton SDK library. To do so, just use the environment variable \texttt{LD_LIBRARY_PATH} the following way:

\$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/eaton_so_libraries

For example, if Eaton SDK dynamic libraries reside in /opt/eaton/lib, use:

\$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/eaton/lib

4.3.2 Windows build

On Windows, building with the SDK has been tested with GCC in MinGW environment and CL in MS Visual Studio 10 environment.

4.4 Building SNMP application

You must call libeaton_init with your device’s IP.
This is a sample code displaying the device status:

```
#include "common.h"
#include "libeaton.h"

int main(void)
{
    /* This contains the result of our request to the device */
    const char * status;

    /* init the library */
    libeaton_init("192.168.0.17");

    /* This is the real request to the device to update internal database. This may be time consuming */
    libeaton_update();

    /* Here we get the status of the device from the database */
    status = libeaton_read("ups.status");

    printf("%s\n",status);

    /* Free SDK internals */
    libeaton_free();

    exit(EXIT_SUCCESS);
}
```

Save this code in the file test.c
4.4.1 Linux build

You need net-snmp library available on your system. Make sure “net-snmp-config” command is available. Building statically:

```
gcc test.c -I<path-to-include> <path-to-lib>/libeaton_snmp.a 'net-snmp-config --base-lib-<base-lib>' 
cflags --libs' -o test
```

Building with shared object:

```
gcc test.c -I<path-to-include> -L <path-to-lib> -leaton_snmp 'net-snmp-config --base-lib-<base-lib>' 
cflags --libs' -o test
```

4.4.2 Windows build

**With MSVC**

You will need the libeaton_snmp.lib file. This is how to create it from libeaton_snmp-0.dll.

```
pexports libeaton_snmp-0.dll > libeaton_snmp.def
lib /machine:i386 /def:libeaton_snmp.def
```

Then you can link it to your application.

```
c1 test.c -DWIN32 libeaton_snmp.lib
```

You need libnetsnmp-30.dll, libregex-1.dll and libeaton_snmp-0.dll in your path to run the resulting executable.

**With MinGW**

Building statically:

```
gcc test.c -I<path-to-include> libeaton_snmp.a libnetsnmp.a libregex.a -lws2_32
```

Building with shared object:

```
gcc test.c -I<path-to-include> libeaton_snmp.dll.a libcommon.dll.a
```

You need libnetsnmp-30.dll, libregex-1.dll and libeaton_snmp-0.dll in your path to run the resulting executable.

4.5 Building XML/HTTP application

You must call libeaton_init with your device’s URL.

This is a sample code displaying the device status:

```
#include "common.h"
#include "libeaton.h"

int main(void)
{
    /* This will contain the result of our request to the device */
    const char * status;

    /* init the library */
    libeaton_init("http://192.168.0.17");

    /* This is the real request to the device to update internal database.
    This may be time consuming */
    libeaton_update();
}
```
Eaton Software Development Kit

/* Here we get the status of the device from the database */
status = libeaton_read("ups.status");
printf("%s\n",status);
/* Free SDK internals */
libeaton_free();
exit(EXIT_SUCCESS);

Save this code in the file test.c

4.5.1 Linux build

You need neon library available on your system. Make sure “pkg-config neon” command is available.

Building statically:
gcc test.c -I<path-to-include> <path-to-lib>/libeaton_xmlpdc.a 'pkg-config --cflags --libs neon' -o test

Building with shared object:
gcc test.c -I<path-to-include> -L <path-to-lib> -leaton_xmlpdc 'pkg-config --cflags --libs neon' -o test

4.5.2 Windows build

With MSVC

You will need the libeaton_xmlpdc.lib file. This is how to create it from libeaton_xmlpdc-0.dll.
pexports libeaton_xmlpdc-0.dll > libeaton_xmlpdc.def
lib /machine:i386 /def:libeaton_xmlpdc.def

Then you can link it to your application.
c1 test.c -DWIN32 libeaton_xmlpdc.lib

You need libneon-27.dll, libeaton_xmlpdc-0.dll, libexpat-1.dll, libiconv-2.dll, libintl-8.dll and libz-1.dll in your path to run the resulting executable.

With MinGW

Building statically:
gcc test.c -I<path-to-include> libeaton_xmlpdc.a libneon.a libintl.a libexpat.a libiconv.a -lws2_32

Building with shared object:
gcc test.c -I<path-to-include> libeaton_xmlpdc.dll.a

You need libneon-27.dll, libeaton_xmlpdc-0.dll, libexpat-1.dll, libiconv-2.dll, libintl-8.dll and libz-1.dll in your path to run the resulting executable.
4.6 Building USB HID application

You must call libeaton_init with NULL as a parameter.

This is a sample code displaying the device status:

```c
#include "common.h"
#include "libeaton.h"

int main(void)
{
    /* This will contain the result of our request to the device */
    const char * status;

    /* init the library */
    libeaton_init(NULL);

    /* This is the real request to the device to update internal database.
    This may be time consuming */
    libeaton_update();

    /* Here we get the status of the device from the database */
    status = libeaton_read("ups.status");

    printf("%s\n",status);

    /* Free SDK internals */
    libeaton_free();

    exit(EXIT_SUCCESS);
}
```

Save this code in the file test.c

4.6.1 Linux build

You need libusb library available on your system. Make sure “pkg-config libusb” command is available. Building statically:

```bash
gcc test.c -I<path-to-include> <path-to-lib>/libeaton_usbhid.a 'pkg-config --cflags --libs libusb' -o test
```

Building with shared object:

```bash
gcc test.c -I<path-to-include> -L <path-to-lib> -leaton_usbhid 'pkg-config --cflags --libs libusb' -o test
```

4.6.2 Windows build

You must install a libusb driver for your device. You can either install the device driver (NOT the filter driver) from the official libusb-win32 binaries or use the wdi-simple.exe of the NUT Windows’ package which install it automatically.

With MSVC

You will need the libeaton_usbhid.lib file. This is how to create it from libeaton_usbhid-0.dll.

```bash
pexports libeaton_usbhid-0.dll > libeaton_usbhid.def
lib /machine:i386 /def:libeaton_usbhid.def
```

Then you can link it to your application.
You need libeaton_usbhid-0.dll, libregex-1.dll and libusb0.dll in your path to run the resulting executable.

**With MinGW**

Building statically:

```bash
gcc test.c -I<path-to-include> libeaton_usbhid.lib libusb.a -lws2_32
```

You need libusb0.dll in your path to run the resulting executable.

Building with shared object:

```bash
gcc test.c -I<path-to-include> libeaton_usbhid.dll.a
```

You need libeaton_usbhid-0.dll, libregex-1.dll and libusb0.dll in your path to run the resulting executable.

### 4.7 Building USB XCP application

You must call libeaton_init with NULL as a parameter.

This is a sample code displaying the device status:

```c
#include "common.h"
#include "libeaton.h"

int main(void)
{
    /* This will contains the result of our request to the device */
    const char * status;

    /* init the library */
    libeaton_init(NULL);

    /* This is the real request to the device to update internal database.
    This may be time consuming */
    libeaton_update();

    /* Here we get the status of the device from the database */
    status = libeaton_read("ups.status");
    printf("%s\n", status);

    /* Free SDK internals */
    libeaton_free();

    exit(EXIT_SUCCESS);
}
```

Save this code in the file test.c

#### 4.7.1 Linux build

You need libusb library available on your system. Make sure “pkg-config libusb” command is available. Building statically:

```bash
gcc test.c -I<path-to-include> <path-to-lib>/libeaton_usbxcp.a 'pkg-config --cflags --libs libusb' -o test
```
Building with shared object:

```bash
gcc test.c -I<path-to-include> -L <path-to-lib> -leaton_usbxcp 'pkg-config --cflags --libs libusb' -o test
```

### 4.7.2 Windows build

You must install a libusb driver for your device. You can either install the device driver (NOT the filter driver) from the official libusb-win32 binaries or use the wdi-simple.exe of the NUT Windows’ package which install it automatically.

**With MSVC**

You will need the libeaton_usbxcp.lib file. This is how to create it from libeaton_usbxcp-0.dll.

```bash
pexports libeaton_usbxcp-0.dll > libeaton_usbxcp.def
lib /machine:i386 /def:libeaton_usbxcp.def
```

Then you can link it to your application.

```bash
cl test.c -DWIN32 libeaton_usbxcp.lib
```

You need libeaton_usbxcp-0.dll, libregex-1.dll and libusb0.dll in your path to run the resulting executable.

**With MinGW**

Building statically:

```bash
gcc test.c -I<path-to-include> libeaton_usbxcp.a libusb.a libgnu.a -lws2_32
```

You need libusb0.dll in your path to run the resulting executable.

Building with shared object:

```bash
gcc test.c -I<path-to-include> libeaton_usbxcp.dll.a
```

You need libeaton_usbxcp-0.dll, libregex-1.dll and libusb0.dll in your path to run the resulting executable.

### 4.8 Building serial XCP application

You must call libeaton_init with the name of the serial port device. On Linux, it is something like "/dev/ttyS0" or "/dev/ttyUSB0". On windows, it is something like "COM1".

Note that the initialization phase may be very long (about 30 seconds).

This is a sample code displaying the device status:

```c
#include "common.h"
#include "libeaton.h"

int main(void)
{
    /* This will contains the result of our request to the device */
    const char * status;
    /* init the library */
    libeaton_init("/dev/ttyS0");
    /* This is the real request to the device to update internal database.
    This may be time consuming */
```
libeaton_update();

/* Here we get the status of the device from the database */
status = libeaton_read("ups.status");

printf("%s\n", status);

/* Free SDK internals */
libeaton_free();

exit(EXIT_SUCCESS);
}

Save this code in the file test.c

### 4.8.1 Linux build

Building statically:

```
gcc test.c -I<path-to-include> <path-to-lib>/libeaton_serialxcp.a -o test
```

Building with shared object:

```
gcc test.c -I<path-to-include> -L <path-to-lib> -leaton_serialxcp -o test
```

### 4.8.2 Windows build

#### With MSVC

You will need the libeaton_serialxcp.lib file. This is how to create it from libeaton_serialxcp-0.dll.

```
pexports libeaton_serialxcp-0.dll > libeaton_serialxcp.def
lib /machine:i386 /def:libeaton_serialxcp.def
```

Then you can link it to your application.

```
c1 test.c -DWIN32 libeaton_serialxcp.lib
```

You need libeaton_serialxcp-0.dll in your path to run the resulting executable.

#### With MinGW

Building statically:

```
gcc test.c -I<path-to-include> libeaton_serialxcp.a -lws2_32
```

Building with shared object:

```
gcc test.c -I<path-to-include> libeaton_serialxcp.dll.a
```

You need libeaton_serialxcp-0.dll in your path to run the resulting executable.
4.9 Building SHUT application

You must call libeaton_init with the name of the serial port device. On Linux, it is something like "/dev/ttyS0" or "/dev/ttyUSB0". On windows, it is something like "COM1".

Note that the initialization phase may be very long (about 20 seconds).

This is a sample code displaying the device status:

```c
#include "common.h"
#include "libeaton.h"

int main(void)
{
    /* This will contains the result of our request to the device */
    const char * status;

    /* init the library */
    libeaton_init("/dev/ttyS0");

    /* This is the real request to the device to update internal database.
    This may be time consuming */
    libeaton_update();

    /* Here we get the status of the device from the database */
    status = libeaton_read("ups.status");

    printf("%s\n",status);

    /* Free SDK internals */
    libeaton_free();

    exit(EXIT_SUCCESS);
}
```

Save this code in the file test.c

### 4.9.1 Linux build

**Building statically:**

```
gcc test.c -I<path-to-include> <path-to-lib>/libeaton_shut.a -o test
```

**Building with shared object:**

```
gcc test.c -I<path-to-include> -L <path-to-lib> -leaton_shut -o test
```

### 4.9.2 Windows build

**With MSVC**

You will need the libeaton_shut.lib file. This is how to create it from libeaton_shut-0.dll.

```
pexports libeaton_shut-0.dll > libeaton_shut.def
lib /machine:i386 /def:libeaton_shut.def
```

Then you can link it to your application.

```
c1 test.c -DWIN32 libeaton_shut.lib
```

You need libeaton_shut-0.dll in your path to run the resulting executable.
With MinGW

Building statically:
```
gcc test.c -I<path-to-include> libeaton_shut.a -lws2_32
```

Building with shared object:
```
gcc test.c -I<path-to-include> libeaton_shut.dll.a
```

You need libeaton_shut-0.dll in your path to run the resulting executable.

5 Using the complete NUT framework

**Note**
This approach does not suffer the limitation of the library SDK, as presented in the SDK limitation chapter!

The NUT framework basically consists in a server (upsd) connected to drivers (which are themselves connected to device). Once this framework is running, clients can access device data and send commands via the upsd server.
There is a number of client applications available with NUT (upsc, upsmon...), which can serve as reference implementations. You can also write your own using the upsclient library. This is discussed in the NUT documentation.

**Warning**
NUT libupsclient is licensed under GNU GPL. Refer to the chapter Legal limitations for more information.

As stated in the above link, if you can’t use the upsclient library, you can still directly access the nut server by implementing the client - server protocol which is documented here.

6 Using part of the NUT framework

**Note**
This approach does not suffer the limitation of the library SDK, as presented in the SDK limitation chapter!

As presented in the previous chapter, the NUT framework consists of driver(s) connected to a server. You may choose to not use the server and to connect directly to the driver. For information on how to interface with the NUT drivers, refer to the protocol.

For a Linux minimalistic implementation of this method, you can also refer to the server/sockdebug.c debug program, that can be found in the NUT sources.

7 Protocol re-implementation

You may choose not to use NUT at all and directly re-implement the communication protocol with the device. In this case you may find useful information hereafter.

7.1 USB and shut

Example of specific protocol reimplementation, for USB/HID with an Eaton Protection Station

1) Get the data topology

Using the NUT - Network UPS Tools usbhid-ups driver, launch it with debug level 1:

```
$ /path/to/usbhid-ups -D -a <device name>
```

```
2.632336 Using subdriver: MGE HID 1.27
           x01, Offset: 0, Size: 1, Value: 1
3.432618 Path: UPS.PowerSummary.PresentStatus.ACPresent, Type: Input, ReportID: 0x01 ←
           , Offset: 0, Size: 1, Value: 1
(...) 3.569335 Path: UPS.PowerSummary.RunTimeToEmpty, Type: Feature, ReportID: 0x06, ←
           Offset: 8, Size: 32, Value: 1171
3.569357 Path: UPS.PowerSummary.RunTimeToEmpty, Type: Input, ReportID: 0x06, Offset: ←
           8, Size: 32, Value: 1171
3.671941 Detected a UPS: EATON/Protection Station 650
```
The above is only a small excerpt of data provided by USB/HID devices. It shows the various data that are available for a specific device, and how to request these and extract the result.

You can then switch to debug level 3 (ie "-DDD") to see the raw USB frames:

```
$ /path/to/usbhid-ups -DDD -a <device name>
```

2.623379 Using subdriver: MGE HID 1.27
(...)
3.425409 Report[get]: (4 bytes) => 01 25 00 00
3.425454 Path: UPS.PowerSummary.PresentStatus.ACPresent, Type: Feature, ReportID: 0 ⇐
x01, Offset: 0, Size: 1, Value: 1
(...)

2) Understand how to use these information

```
Report[get]: (4 bytes) => 01 25 00 00
```

This line means that the device send us 4 bytes. The first one is the ReportID (here : 0x01), then we have 3 data bytes.

```
Path: UPS.PowerSummary.PresentStatus.ACPresent, Type: Feature, ReportID: 0x01, Offset: 0, ⇐
Size: 1, Value: 1
```

This line is the interpretation of the above data by NUT. It says that it is the ReportID 0x01 and that at offset 0 (in bits) we have a data of size 1 (in bits) with a value of 1 (0x25 = 00100101). This data HID path is UPS.PowerSummary.PresentStatus.ACPresent. This means that this device is on-line. "Type: Feature" means that this data is a "status" data which is fetched by the application. On the contrary you can have "Type: Input" which are "events" sent asynchronously by the UPS. This tutorial will focus on "Feature" data as it is simpler to implement.

Let’s see another example.

```
Report[buf]: (6 bytes) => 06 64 93 04 00 00
Path: UPS.PowerSummary.RunTimeToEmpty, Type: Input, ReportID: 0x06, Offset: 8, Size: 32, ⇐
Value: 1171
```

This is the reportID 0x06, in which we have at offset 8 (bits, so the third byte of the frame after the reportID byte and the 8 first data bits) 32 bits of data (4 bytes) with a value of 1171 (0x00000493).

3) Sample code

This example illustrate the retrieving of the power status, that is to say if the device is operating on battery or from line power.

```
#include <stdio.h>
#include <usb.h> /* libusb header file */
#define EATON_MGE_VENDORID 0x0463

int main()
{
    unsigned char buf[20];
    struct usb_device *dev;
    struct usb_bus *bus;
    usb_dev_handle *usbdev;
    int ret, reportId;
    /* libusb base init */
    usb_init();
    usb_find_busses();
    usb_find_devices();
    /* Enumerate USB busses to find the right device */
    for (bus = usb_busses; bus; bus = bus->next) {
        for (dev = bus->devices; dev; dev = dev->next) {
```
if (dev->descriptor.idVendor == EATON_MGE_VENDORID) {
    /* You may also want to check the device identifier using
     dev->descriptor.idProduct */
    printf("USB UPS found\n");
    /* Now open the device */
    usbdev = usb_open(dev);
    /* And claim the first interface. 
This may need several tries, and (for Linux) a call to
usb_detach_kernel_driver_np()
    to replace the Integrated power management */
    #ifdef WIN32
    usb_set_configuration(usbdev,1);
    #endif
    usb_claim_interface(usbdev, 0);
    /* Send a request to get UPS.PowerSummary.PresentStatus. ←
    ACPIPresent ←
    reportId = 1;
    ret = usb_control_msg(usbdev,
        USB_ENDPOINT_IN + USB_TYPE_CLASS +
        USB_RECIP_INTERFACE,
        0x01, /* HID_REPORT_GET */
        reportId+(0x03<<8), /* HID_REPORT_TYPE_FEATURE */
        0, buf, 128, /* ask for a large ReportSize */
        5000); /* 5 seconds timeout */
    printf("Received %i bytes\n");
    /* Now process the answer, and get the power status */
    if (ret > 0) {
        /* Data bit extraction logic :
        * First byte of report is report ID:
            Bit = bufOffset + 8; */
        /* Offset: 0, Size: 1 (in bit(s)) */
        /* Buf[Bit >> 3] & (1 << (Bit & 7)); */
        if ((buf[1] >> 0 & 01) == 1)
            printf("UPS is online\n");
        else
            printf("UPS is on battery\n");
    }
}
exit (EXIT_SUCCESS);

Note
in order to execute this program, you will need to have proper permissions to access the device. Either run it as root, or preferably use udev (on Linux) or equivalent system to set the proper rights to access the device.

Compile with gcc testusb.c -lusb...
Test by running the program, a first time with AC present:
$ sudo ./a.out
USB UPS found
Received 4 bytes
UPS is online

Then a second time after having removed the main power:

$ sudo ./a.out
USB UPS found
Received 4 bytes
UPS is on battery

---

Note
Another approach to communicate directly with USB devices is to use convenient HID library such as hidapi.

4) SHUT implementation
For SHUT reimplementation, you may have a look in the Simplified SHUT protocol document.
You should also have a look at libshut.c file for a reference implementation.

7.2 XCP
XCP protocol specifications are on line here.
You can find a reference implementation in the NUT source tree in bcmxcp* files.

7.3 SNMP
You can find useful lookup table in the NUT source tree in *-mib.c files.

A NUT command and variable naming scheme
This is a dump of the standard variables and command names used in NUT. Don’t use a name with any of the dstate functions unless it exists here.
If you need a new variable or command name, contact the Development Team first.
Put another way: if you make up a name that’s not in this list and it gets into the tree, and then we come up with a better name later, clients that use the undocumented variable will break when it is changed.

---

Note
"opaque" means programs should not attempt to parse the value for that variable as it may vary greatly from one UPS to the next. These strings are best handled directly by the user.

A.1 Variables

A.1.1 device: General unit information

---

Note
these data will be redundant with some ups.* information during a transition period. The ups.* data will then be removed.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>device.model</td>
<td>Device model</td>
<td>BladeUPS</td>
</tr>
<tr>
<td>device.mfr</td>
<td>Device manufacturer</td>
<td>Eaton</td>
</tr>
<tr>
<td>device.serial</td>
<td>Device serial number (opaque string)</td>
<td>WS9643050926</td>
</tr>
<tr>
<td>device.type</td>
<td>Device type (ups, pdu, scd)</td>
<td>ups</td>
</tr>
</tbody>
</table>

### A.1.2 ups: General unit information

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ups.status</td>
<td>UPS status</td>
<td>OL</td>
</tr>
<tr>
<td>ups.alarm</td>
<td>UPS alarms</td>
<td>OVERHEAT</td>
</tr>
<tr>
<td>ups.time</td>
<td>Internal UPS clock time (opaque string)</td>
<td>12:34</td>
</tr>
<tr>
<td>ups.date</td>
<td>Internal UPS clock date (opaque string)</td>
<td>01-02-03</td>
</tr>
<tr>
<td>ups.model</td>
<td>UPS model</td>
<td>Eaton 5PX 1500</td>
</tr>
<tr>
<td>ups.mfr</td>
<td>UPS manufacturer</td>
<td>EATON</td>
</tr>
<tr>
<td>ups.mfr.date</td>
<td>UPS manufacturing date (opaque string)</td>
<td>10/17/96</td>
</tr>
<tr>
<td>ups.serial</td>
<td>UPS serial number (opaque string)</td>
<td>WS9643050926</td>
</tr>
<tr>
<td>ups.vendorid</td>
<td>Vendor ID for USB devices</td>
<td>0463</td>
</tr>
<tr>
<td>ups.productid</td>
<td>Product ID for USB devices</td>
<td>0001</td>
</tr>
<tr>
<td>ups.firmware</td>
<td>UPS firmware (opaque string)</td>
<td>50.9.D</td>
</tr>
<tr>
<td>ups.firmware.aux</td>
<td>Auxiliary device firmware</td>
<td>4Kx</td>
</tr>
<tr>
<td>ups.temperature</td>
<td>UPS temperature (degrees C)</td>
<td>042.7</td>
</tr>
<tr>
<td>ups.load</td>
<td>Load on UPS (percent)</td>
<td>023.4</td>
</tr>
<tr>
<td>ups.load.high</td>
<td>Load when UPS switches to overload condition (&quot;OVER&quot;) (percent)</td>
<td>100</td>
</tr>
<tr>
<td>ups.id</td>
<td>UPS system identifier (opaque string)</td>
<td>Sierra</td>
</tr>
<tr>
<td>ups.delay.start</td>
<td>Interval to wait before restarting the load (seconds)</td>
<td>0</td>
</tr>
<tr>
<td>ups.delay.reboot</td>
<td>Interval to wait before rebooting the UPS (seconds)</td>
<td>60</td>
</tr>
<tr>
<td>ups.delay.shutdown</td>
<td>Interval to wait after shutdown with delay command (seconds)</td>
<td>20</td>
</tr>
<tr>
<td>ups.timer.start</td>
<td>Time before the load will be started (seconds)</td>
<td>30</td>
</tr>
<tr>
<td>ups.timer.reboot</td>
<td>Time before the load will be rebooted (seconds)</td>
<td>10</td>
</tr>
<tr>
<td>ups.timer.shutdown</td>
<td>Time before the load will be shutdown (seconds)</td>
<td>20</td>
</tr>
<tr>
<td>ups.test.interval</td>
<td>Interval between self tests (seconds)</td>
<td>1209600 (two weeks)</td>
</tr>
<tr>
<td>ups.test.result</td>
<td>Results of last self test (opaque string)</td>
<td>Bad battery pack</td>
</tr>
<tr>
<td>ups.display.language</td>
<td>Language to use on front panel (* opaque)</td>
<td>E</td>
</tr>
<tr>
<td>ups.contacts</td>
<td>UPS external contact sensors (* opaque)</td>
<td>F0</td>
</tr>
<tr>
<td>ups.efficiency</td>
<td>Efficiency of the UPS (ratio of the output current on the input current) (percent)</td>
<td>95</td>
</tr>
<tr>
<td>ups.power</td>
<td>Current value of apparent power (Volt-Amps)</td>
<td>500</td>
</tr>
<tr>
<td>ups.power.nominal</td>
<td>Nominal value of apparent power (Volt-Amps)</td>
<td>500</td>
</tr>
<tr>
<td>ups.realpower</td>
<td>Current value of real power (Watts)</td>
<td>300</td>
</tr>
</tbody>
</table>
### Eaton Software Development Kit

#### A.1.3 input: Incoming line/power information

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ups.realpower.nominal</td>
<td>Nominal value of real power (Watts)</td>
<td>300</td>
</tr>
<tr>
<td>ups.beeper.status</td>
<td>UPS beeper status (enabled, disabled or muted)</td>
<td>enabled</td>
</tr>
<tr>
<td>ups.type</td>
<td>UPS type (*opaque)</td>
<td>offline</td>
</tr>
<tr>
<td>ups.watchdog.status</td>
<td>UPS watchdog status (enabled or disabled)</td>
<td>disabled</td>
</tr>
<tr>
<td>ups.start.auto</td>
<td>UPS starts when mains is (re)applied</td>
<td>yes</td>
</tr>
<tr>
<td>ups.start.battery</td>
<td>Allow to start UPS from battery</td>
<td>yes</td>
</tr>
<tr>
<td>ups.start.reboot</td>
<td>UPS coldstarts from battery (enabled or disabled)</td>
<td>yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>input.voltage</td>
<td>Input voltage</td>
<td>121.5</td>
</tr>
<tr>
<td>input.voltage.maximum</td>
<td>Maximum incoming voltage seen</td>
<td>130</td>
</tr>
<tr>
<td>input.voltage.minimum</td>
<td>Minimum incoming voltage seen</td>
<td>100</td>
</tr>
<tr>
<td>input.voltage.nominal</td>
<td>Nominal input voltage</td>
<td>120</td>
</tr>
<tr>
<td>input.voltage.extended</td>
<td>Extended input voltage range</td>
<td>no</td>
</tr>
<tr>
<td>input.transfer.reason</td>
<td>Reason for last transfer to battery (*opaque)</td>
<td>T</td>
</tr>
<tr>
<td>input.transfer.low</td>
<td>Low voltage transfer point</td>
<td>91</td>
</tr>
<tr>
<td>input.transfer.high</td>
<td>High voltage transfer point</td>
<td>132</td>
</tr>
<tr>
<td>input.transfer.low.min</td>
<td>Smallest settable low voltage transfer point</td>
<td>85</td>
</tr>
<tr>
<td>input.transfer.low.max</td>
<td>Greatest settable low voltage transfer point</td>
<td>95</td>
</tr>
<tr>
<td>input.transfer.high.min</td>
<td>Smallest settable high voltage transfer point</td>
<td>131</td>
</tr>
<tr>
<td>input.transfer.high.max</td>
<td>Greatest settable high voltage transfer point</td>
<td>136</td>
</tr>
<tr>
<td>input.sensitivity</td>
<td>Input power sensitivity</td>
<td>H (high)</td>
</tr>
<tr>
<td>input.quality</td>
<td>Input power quality (*opaque)</td>
<td>FF</td>
</tr>
<tr>
<td>input.current</td>
<td>Input current (A)</td>
<td>4.25</td>
</tr>
<tr>
<td>input.current.nominal</td>
<td>Nominal input current (A)</td>
<td>5.0</td>
</tr>
<tr>
<td>input.frequency</td>
<td>Input line frequency (Hz)</td>
<td>60.00</td>
</tr>
<tr>
<td>input.frequency.nominal</td>
<td>Nominal input line frequency (Hz)</td>
<td>60</td>
</tr>
<tr>
<td>input.frequency.low</td>
<td>Input line frequency low (Hz)</td>
<td>47</td>
</tr>
<tr>
<td>input.frequency.high</td>
<td>Input line frequency high (Hz)</td>
<td>63</td>
</tr>
<tr>
<td>input.frequency.extended</td>
<td>Extended input frequency range</td>
<td>no</td>
</tr>
<tr>
<td>input.transfer.boost.low</td>
<td>Low voltage boosting transfer point</td>
<td>190</td>
</tr>
<tr>
<td>input.transfer.boost.high</td>
<td>High voltage boosting transfer point</td>
<td>210</td>
</tr>
<tr>
<td>input.transfer.trim.low</td>
<td>Low voltage trimming transfer point</td>
<td>230</td>
</tr>
<tr>
<td>input.transfer.trim.high</td>
<td>High voltage trimming transfer point</td>
<td>240</td>
</tr>
</tbody>
</table>

### A.1.4 output: Outgoing power/inverter information

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>output.voltage</td>
<td>Output voltage (V)</td>
<td>120.9</td>
</tr>
<tr>
<td>output.voltage.nominal</td>
<td>Nominal output voltage (V)</td>
<td>120</td>
</tr>
<tr>
<td>output.frequency</td>
<td>Output frequency (Hz)</td>
<td>59.9</td>
</tr>
<tr>
<td>output.frequency.nominal</td>
<td>Nominal output frequency (Hz)</td>
<td>60</td>
</tr>
<tr>
<td>output.current</td>
<td>Output current (A)</td>
<td>4.25</td>
</tr>
</tbody>
</table>
### A.1.5 Three-phase additions

The additions for three-phase measurements would produce a very long table due to all the combinations that are possible, so these additions are broken down to their base components.

**Phase Count Determination**

input.phases (3 for three-phase, absent or 1 for 1phase) output.phases (as for input.phases)

**DOMAINs**

Any input or output is considered a valid DOMAIN.

input (should really be called input.mains, but keep this for compat) input.bypass input.servicebypass

output (should really be called output.load, but keep this for compat) output.bypass output.inverter output.servicebypass

**Specification (SPEC)**

Voltage, current, frequency, etc are considered to be a specification of the measurement.

With this notation, the old 1phase naming scheme becomes DOMAIN.SPEC Example: `input.current`

**CONTEXT**

When in three-phase mode, we need some way to specify the target for most measurements in more detail. We call this the CONTEXT.

With this notation, the naming scheme becomes DOMAIN.CONTEXT.SPEC when in three-phase mode. Example: `input.L1.current`

**Valid CONTEXTs**

```
L1-L2  \   
L2-L3  \   
L3-L1  for voltage measurements
L1-N   /   
L2-N   /   
L3-N   /   
L1    \   
L2    for current and power measurements
L3    /   
N     - for current measurement
```

**Valid SPECS**

Valid with/without context (ie. per phase or aggregated/averaged)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>Current (A)</td>
</tr>
<tr>
<td>current.maximum</td>
<td>Maximum seen current (A)</td>
</tr>
<tr>
<td>current.minimum</td>
<td>Minimum seen current (A)</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>peakcurrent</td>
<td>Peak current</td>
</tr>
<tr>
<td>voltage</td>
<td>Voltage (V)</td>
</tr>
<tr>
<td>voltage.nominal</td>
<td>Nominal voltage (V)</td>
</tr>
<tr>
<td>voltage.maximum</td>
<td>Maximum seen voltage (V)</td>
</tr>
<tr>
<td>voltage.minimum</td>
<td>Minimum seen voltage (V)</td>
</tr>
<tr>
<td>power</td>
<td>Apparent power (VA)</td>
</tr>
<tr>
<td>power.maximum</td>
<td>Maximum seen apparent power (VA)</td>
</tr>
<tr>
<td>power.minimum</td>
<td>Maximum seen apparent power (VA)</td>
</tr>
<tr>
<td>power.percent</td>
<td>Percentage of apparent power related to maximum load</td>
</tr>
<tr>
<td>power.maximum.percent</td>
<td>Max seen percentage of apparent power</td>
</tr>
<tr>
<td>power.minimum.percent</td>
<td>Min seen percentage of apparent power</td>
</tr>
<tr>
<td>realpower</td>
<td>Real power (W)</td>
</tr>
<tr>
<td>powerfactor</td>
<td>Power Factor (dimensionless value between 0.00 and 1.00)</td>
</tr>
<tr>
<td>crestfactor</td>
<td>Crest Factor (dimensionless value greater or equal to 1)</td>
</tr>
</tbody>
</table>

Valid without context (ie. aggregation of all phases):

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequency</td>
<td>Frequency (Hz)</td>
</tr>
<tr>
<td>frequency.nominal</td>
<td>Nominal frequency (Hz)</td>
</tr>
</tbody>
</table>

**A.1.6 EXAMPLES**

Partial Three phase - Three phase example:

```plaintext
input.phases: 3
input.frequency: 50.0
input.L1.current: 133.0
input.bypass.L1-L2.voltage: 398.3
output.phases: 3
output.L1.power: 35700
output.powerfactor: 0.82
```

Partial Three phase - One phase example:

```plaintext
input.phases: 3
input.L2.current: 48.2
input.N.current: 3.4
input.L3-L1.voltage: 405.4
input.frequency: 50.1
output.phases: 1
output.current: 244.2
output.voltage: 120
output.frequency.nominal: 60.0
```

**A.1.7 battery: Any battery details**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>battery.charge</td>
<td>Battery charge (percent)</td>
<td>100.0</td>
</tr>
<tr>
<td>battery.charge.low</td>
<td>Remaining battery level when UPS switches to LB (percent)</td>
<td>20</td>
</tr>
<tr>
<td>battery.charge.restart</td>
<td>Minimum battery level for UPS restart after power-off</td>
<td>20</td>
</tr>
</tbody>
</table>
### battery

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>battery.charge.warning</td>
<td>Battery level when UPS switches to &quot;Warning&quot; state (percent)</td>
<td>50</td>
</tr>
<tr>
<td>battery.voltage</td>
<td>Battery voltage (V)</td>
<td>24.84</td>
</tr>
<tr>
<td>battery.capacity</td>
<td>Battery capacity (Ah)</td>
<td>7.2</td>
</tr>
<tr>
<td>battery.current</td>
<td>Battery current (A)</td>
<td>1.19</td>
</tr>
<tr>
<td>battery.temperature</td>
<td>Battery temperature (degrees C)</td>
<td>050.7</td>
</tr>
<tr>
<td>battery.voltage.nominal</td>
<td>Nominal battery voltage (V)</td>
<td>024</td>
</tr>
<tr>
<td>battery.runtime</td>
<td>Battery runtime (seconds) Remaining battery runtime</td>
<td>1080</td>
</tr>
<tr>
<td>battery.runtime.low</td>
<td>when UPS switches to LB (seconds)</td>
<td>180</td>
</tr>
<tr>
<td>battery.alarm.threshold</td>
<td>Battery alarm threshold</td>
<td>0 (immediate)</td>
</tr>
<tr>
<td>battery.mfr.date</td>
<td>Battery change date (opaque string)</td>
<td>11/14/00</td>
</tr>
<tr>
<td>battery.packs</td>
<td>Number of battery packs</td>
<td>001</td>
</tr>
<tr>
<td>battery.packs.bad</td>
<td>Number of bad battery packs</td>
<td>000</td>
</tr>
<tr>
<td>battery.type</td>
<td>Battery chemistry (opaque string)</td>
<td>PbAc</td>
</tr>
<tr>
<td>battery.protection</td>
<td>Prevent deep discharge of battery</td>
<td>yes</td>
</tr>
<tr>
<td>battery.energysave</td>
<td>Switch off when running on battery and no/low load</td>
<td>no</td>
</tr>
</tbody>
</table>

#### A.1.8 ambient: Conditions from external probe equipment

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ambient.temperature</td>
<td>Ambient temperature (degrees C)</td>
<td>25.40</td>
</tr>
<tr>
<td>ambient.temperature.alarm</td>
<td>Temperature alarm (enabled/disabled)</td>
<td>enabled</td>
</tr>
<tr>
<td>ambient.temperature.high</td>
<td>Temperature threshold high (degrees C)</td>
<td>40</td>
</tr>
<tr>
<td>ambient.temperature.low</td>
<td>Temperature threshold low (degrees C)</td>
<td>5</td>
</tr>
<tr>
<td>ambient.temperature.maximum</td>
<td>Maximum temperature seen (degrees C)</td>
<td>37.6</td>
</tr>
<tr>
<td>ambient.temperature.minimum</td>
<td>Minimum temperature seen (degrees C)</td>
<td>18.1</td>
</tr>
<tr>
<td>ambient.humidity</td>
<td>Ambient relative humidity (percent)</td>
<td>038.8</td>
</tr>
<tr>
<td>ambient.humidity.alarm</td>
<td>Relative humidity alarm (enabled/disabled)</td>
<td>enabled</td>
</tr>
<tr>
<td>ambient.humidity.high</td>
<td>Relative humidity threshold high (percent)</td>
<td>80</td>
</tr>
<tr>
<td>ambient.humidity.low</td>
<td>Relative humidity threshold high (percent)</td>
<td>10</td>
</tr>
<tr>
<td>ambient.humidity.maximum</td>
<td>Maximum relative humidity seen (percent)</td>
<td>60</td>
</tr>
<tr>
<td>ambient.humidity.minimum</td>
<td>Minimum relative humidity seen (percent)</td>
<td>13</td>
</tr>
</tbody>
</table>

#### A.1.9 outlet: Smart outlet management

**Note**

n stands for the outlet index. For more information, refer to the NUT outlets management and PDU notes chapter of the user manual. A special case is "outlet.0" which is equivalent to "outlet", and represent the whole set of outlets of the device.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>outlet.n.id</td>
<td>Outlet system identifier (opaque string)</td>
<td>1</td>
</tr>
<tr>
<td>outlet.n.desc</td>
<td>Outlet description (opaque string)</td>
<td>Main outlet</td>
</tr>
<tr>
<td>outlet.n.switch</td>
<td>Outlet switch control (on/off)</td>
<td>on</td>
</tr>
<tr>
<td>outlet.n.status</td>
<td>Outlet switch status (on/off)</td>
<td>on</td>
</tr>
<tr>
<td>outlet.n.switchable</td>
<td>Outlet switch ability (yes/no)</td>
<td>yes</td>
</tr>
<tr>
<td>outlet.n.autoswitch.charge.low</td>
<td>Remaining battery level to power off this outlet (percent)</td>
<td>80</td>
</tr>
<tr>
<td>outlet.n.delay.shutdown</td>
<td>Interval to wait before shutting down this outlet (seconds)</td>
<td>180</td>
</tr>
<tr>
<td>outlet.n.delay.start</td>
<td>Interval to wait before restarting this outlet (seconds)</td>
<td>120</td>
</tr>
<tr>
<td>outlet.n.current</td>
<td>Current (A)</td>
<td>0.19</td>
</tr>
<tr>
<td>outlet.n.current.maximum</td>
<td>Maximum seen current (A)</td>
<td>0.56</td>
</tr>
<tr>
<td>outlet.n.realpower</td>
<td>Current value of real power (W)</td>
<td>28</td>
</tr>
<tr>
<td>outlet.n.voltage</td>
<td>Voltage (V)</td>
<td>247.0</td>
</tr>
<tr>
<td>outlet.n.powerfactor</td>
<td>Power Factor (dimensionless value between 0 and 1)</td>
<td>0.85</td>
</tr>
<tr>
<td>outlet.n.crestfactor</td>
<td>Crest Factor (dimensionless, equal to or greater than 1)</td>
<td>1.41</td>
</tr>
<tr>
<td>outlet.n.power</td>
<td>Apparent power (VA)</td>
<td>46</td>
</tr>
</tbody>
</table>

### A.1.10 driver: Internal driver information

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver.name</td>
<td>Driver name</td>
<td>usbhid-ups</td>
</tr>
<tr>
<td>driver.version</td>
<td>Driver version (NUT release)</td>
<td>X.Y.Z</td>
</tr>
<tr>
<td>driver.version.internal</td>
<td>Internal driver version (if tracked separately)</td>
<td>1.23.45</td>
</tr>
<tr>
<td>driver.parameter.xxx</td>
<td>Parameter xxx (ups.conf or cmdline -x) setting</td>
<td>(varies)</td>
</tr>
<tr>
<td>driver.flag.xxx</td>
<td>Flag xxx (ups.conf or cmdline -x) status</td>
<td>enabled (or absent)</td>
</tr>
</tbody>
</table>

### A.1.11 server: Internal server information

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>server.version</td>
<td>Server version</td>
<td>X.Y.Z</td>
</tr>
</tbody>
</table>

### A.2 Instant commands

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>load.off</td>
<td>Turn off the load immediately</td>
</tr>
<tr>
<td>load.on</td>
<td>Turn on the load immediately</td>
</tr>
<tr>
<td>shutdown.return</td>
<td>Turn off the load possibly after a delay and return when power is back</td>
</tr>
<tr>
<td>shutdown.stayoff</td>
<td>Turn off the load possibly after a delay and remain off even if power returns</td>
</tr>
<tr>
<td>shutdown.stop</td>
<td>Stop a shutdown in progress</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>shutdown.reboot</td>
<td>Shut down the load briefly while rebooting the UPS</td>
</tr>
<tr>
<td>shutdown.reboot.graceful</td>
<td>After a delay, shut down the load briefly while rebooting the UPS</td>
</tr>
<tr>
<td>test.panel.start</td>
<td>Start testing the UPS panel</td>
</tr>
<tr>
<td>test.panel.stop</td>
<td>Stop a UPS panel test</td>
</tr>
<tr>
<td>test.failure.start</td>
<td>Start a simulated power failure</td>
</tr>
<tr>
<td>test.failure.stop</td>
<td>Stop simulating a power failure</td>
</tr>
<tr>
<td>test.battery.start</td>
<td>Start a battery test</td>
</tr>
<tr>
<td>test.battery.start.quick</td>
<td>Start a &quot;quick&quot; battery test</td>
</tr>
<tr>
<td>test.battery.start.deep</td>
<td>Start a &quot;deep&quot; battery test</td>
</tr>
<tr>
<td>test.battery.stop</td>
<td>Stop the battery test</td>
</tr>
<tr>
<td>calibrate.start</td>
<td>Start runtime calibration</td>
</tr>
<tr>
<td>calibrate.stop</td>
<td>Stop runtime calibration</td>
</tr>
<tr>
<td>bypass.start</td>
<td>Put the UPS in bypass mode</td>
</tr>
<tr>
<td>bypass.stop</td>
<td>Take the UPS out of bypass mode</td>
</tr>
<tr>
<td>reset.input.minmax</td>
<td>Reset minimum and maximum input voltage status</td>
</tr>
<tr>
<td>reset.watchdog</td>
<td>Reset watchdog timer (forced reboot of load)</td>
</tr>
<tr>
<td>beeper.enable</td>
<td>Enable UPS beeper/buzzer</td>
</tr>
<tr>
<td>beeper.disable</td>
<td>Disable UPS beeper/buzzer</td>
</tr>
<tr>
<td>beeper.mute</td>
<td>Temporarily mute UPS beeper/buzzer</td>
</tr>
<tr>
<td>beeper.toggle</td>
<td>Toggle UPS beeper/buzzer</td>
</tr>
</tbody>
</table>