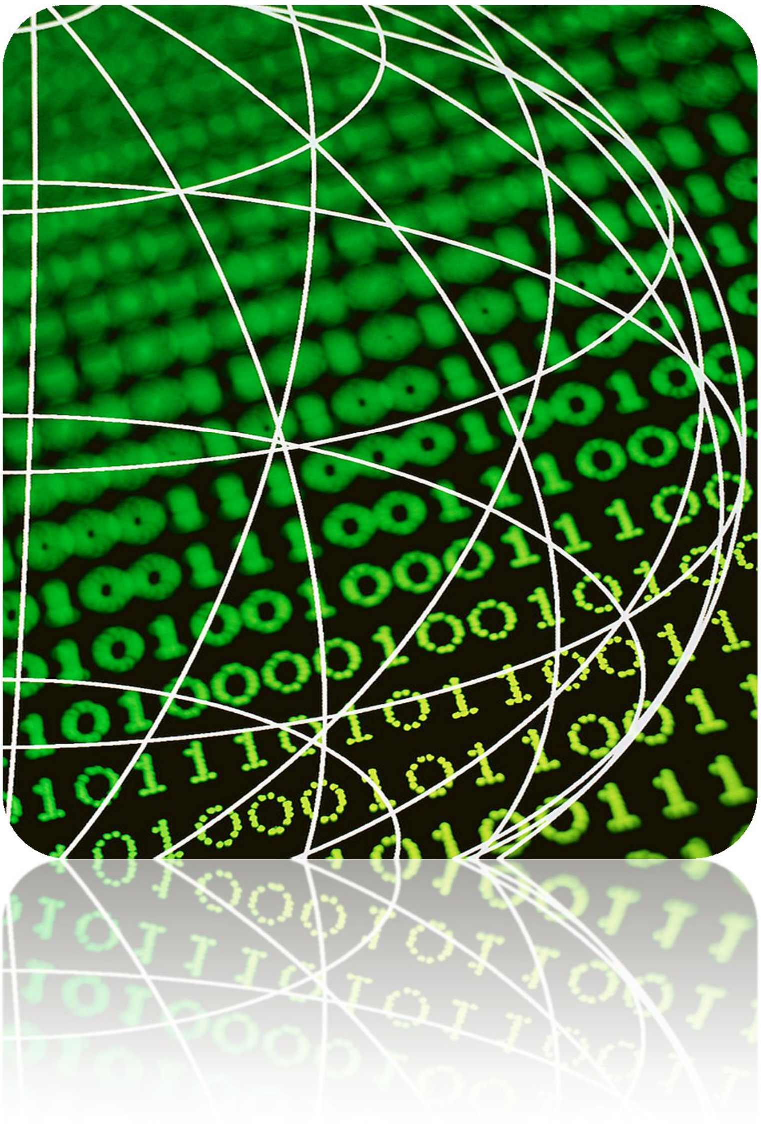
Plugwise unleashed

A document explaining the protocol used by Plugwise products



Author: Maarten Damen (www.maartendamen.com)

Disclaimer:

This document was not written by Plugwise B.V. nor are there any connections between the author and Plugwise B.V. Actually Plugwise B.V. refuses any cooperation on open source products! The information in this document was collected for educational purposes, and to embrace open source support for Plugwise products. The information was collected use merely legal reverse engineering tools (serial port sniffers, debuggers etc.)

Version	Author	Comments
0.1	Maarten Damen	Initial version, open to feedback.

Document index

Introduction

This document describes the protocol used in Plugwise products. The document is still not perfect (there are a few open ends) so any feedback is welcome. There was no cooperation from Plugwise in making this document, instead they refuse any cooperation on open source support.
So don't bother Plugwise about this document please. The creation of this document took me a lot of time, so I would appreciate at least a reference to me/this document when you use it in your project. A donation is off course also very welcome.

Enjoy the document!

Plugwise hardware

The MC (Main Controller) also known as “Stick” utilizes an Ember EM250 chipset (Zigbee PRO), more info can be found here:

http://www.ember.com/products_zigbee_chips_e250.html

The MC communicates directly with the NC (Network Controller), also known as Circle+. The NC communicates with other nodes (Circles) in the mesh network topology. As I never opened a Circle I don’t know what hardware it utilizes.

Plugwise communication takes place in the following way:

1) A request is sent to the MC from the PC.
2) The MC sends the request to the NC.
3) The NC may forward the request to other circles.

My though is that the Circle/Circle+ uses the same hardware, I haven’t wrecked/opened one myself though.

Serial interface

Although the Plugwise stick looks like a USB interface, it actually utilizes a serial protocol. A virtual serial port is provided by an onboard FTDI chip.

Communication protocol

The Plugwise communication protocol uses the same sequence for all commands handled by the MC. It can be divided in a few steps (an example follows later):

1) The PC sends a request to the MC.
2) The MC responds to this request with an response code, including a sequence number.
3) The MC responds again with the result of the request.

How does this look? To illustrate this we will use the example command for stick initialization.

1) The stick initialization command is “000A” with no parameters. A CRC checksum is added to each command, this is a 16bit CRC checksum. For more information about this refer to the CRC topic in this guide. After adding the CRC checksum the command looks like this: “000AB43C”. Please note that the initialization command is always has the same checksum as the checksum is generated over the same command without parameters all the time.
2) After the command has been received by the MC, the MC sends an acknowledge to the PC. The acknowledge response looks like this: “00000F5F00C1E2FA”

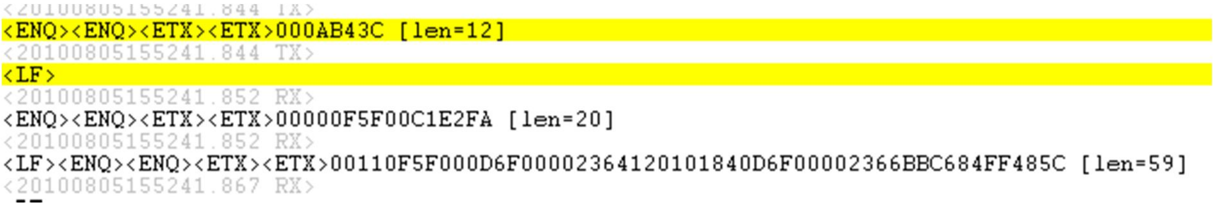
Data	Datatype	Explanation
0000	Integer	This command indicates the

		acknowledge command sent by the MC.
0F5F	Integer	This is the sequence number, each command gets its own sequence number of which you should keep track of in your software. In this case the sequence number is “3935”
00C1	Integer	This is the acknowledge code. 00C1 means the command was successful.
E2FA	Integer	This is the 16bit CRC checksum value.

3) After the acknowledgement has been sent to the PC, the result of the command will be sent over the serial port. In the case of the initialization command it’s response is: “00110F5F000D6F0000236412010184OD6F00002366BBC684FF485C”

   0011 is the command response code, followed by the payload and the command response is finalized by the CRC16 checksum (485C)

Here’s an example from a serial port sniffer:



```
<20100805155241.844 TX>
<ENQ><ENQ><ETX><ETX>000AB43C [len=12]
<20100805155241.844 TX>
<LF>
<20100805155241.852 RX>
<ENQ><ENQ><ETX><ETX>00000F5F00C1E2FA [len=20]
<20100805155241.852 RX>
<LF><ENQ><ENQ><ETX><ETX>00110F5F000D6F0000236412010184OD6F00002366BBC684FF485C [len=59]
<20100805155241.867 RX>
```

Please note that the acknowledge response is ignored for the rest of this document, as it is the same for each and every response.

Stick initialization

The stick needs to be initialized once a connection has been made. The initialization is as follows:

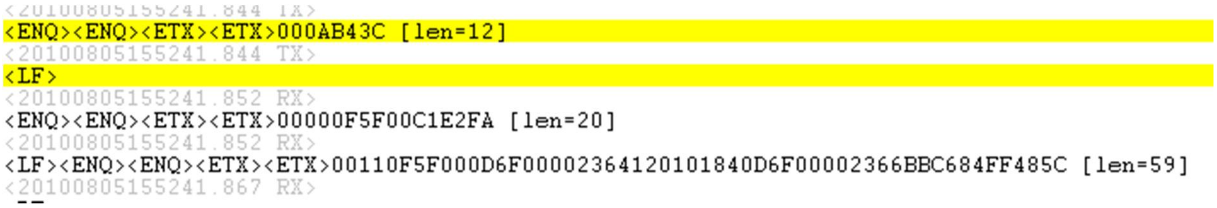
Send -> 000AB43C
Command response -> 00110F5F000D6F0000236412010184OD6F00002366BBC684FF485C

The send request consist of the request code 000A and a CRC checksum value: B43C. The response is described in the table:

Data	Datatype	Explanation
0011	Integer	This is the command response code.
0F5F	Integer	This is the sequence number associated with the request.

000D6F0000236412	Unsigned 64bit integer	This is the mac address of the MC.
01	Boolean	??
01	Boolean	Indicates whether or not the network is online. (Association with Circle+)
840D6F00002366BB	Unsigned 64bit integer	This is the unique network code, not sure how it's generated. (Zigbee PRO?)
C684	Integer	Shorter notation of the network unique ID. This only changes when completely resetting stick+Circle+.
FF	??	Unused, never changes.
485C	Integer	CRC16 checksum.

Here's an example from a serial port sniffer:



```
<20100805155241.844 TX>
<ENQ><ENQ><ETX><ETX>000AB43C [len=12]
<20100805155241.844 TX>
<LF>
<20100805155241.852 RX>
<ENQ><ENQ><ETX><ETX>00000F5F00C1E2FA [len=20]
<20100805155241.852 RX>
<LF><ENQ><ENQ><ETX><ETX>00110F5F000D6F00002364120101840D6F00002366BBC684FF485C [len=59]
<20100805155241.867 RX>
```

Calibration request

Send -> 0026000D6F00002366BB7071
Command response -> 00272CBC000D6F00002366BB3F78BD69B6FF08763CA99962000000000B70

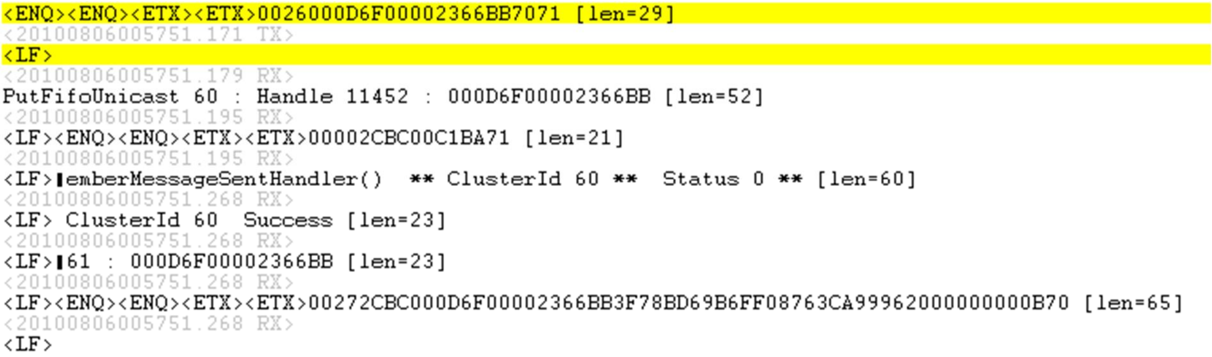
The send request consist of the request code 0026 and a CRC checksum value: 7071. The response is described in the table:

Data	Datatype	Explanation
0027	Integer	This is the command response code.
2CBC	Integer	This is the sequence number associated with the request/response.
000D6F00002366BB	Unsigned 64bit integer	This is the MAC address of the Circle.
3F78BD69	Float	This is the gaina calibration parameter (see PlugwiseData.MDB)
B6FF0876	Float	This is the gainb calibration parameter (see PlugwiseData.MDB)
3CA99962	Float	This is the offtot calibration parameter (see PlugwiseData.MDB)

00000000	Float	This is the offruis calibration parameter (see PlugwiseData.MDB)
0B70	Integer	CRC16 checksum.

All hexadecimal values need to be converted to floats, this will give you the same results as in the Plugwise database file.

Here’s an example from a serial port sniffer:



```
<ENQ><ENQ><ETX><ETX>0026000D6F00002366BB7071 [len=29]
<20100806005751.171 TX>
<LF>
<20100806005751.179 RX>
PutFifoUnicast 60 : Handle 11452 : 000D6F00002366BB [len=52]
<20100806005751.195 RX>
<LF><ENQ><ENQ><ETX><ETX>00002CBC00C1BA71 [len=21]
<20100806005751.195 RX>
<LF>▮emberMessageSentHandler()  ** ClusterId 60 **  Status 0 ** [len=60]
<20100806005751.268 RX>
<LF> ClusterId 60  Success [len=23]
<20100806005751.268 RX>
<LF>▮61 : 000D6F00002366BB [len=23]
<20100806005751.268 RX>
<LF><ENQ><ENQ><ETX><ETX>00272CBC000D6F00002366BB3F78BD69B6FF08763CA99962000000000B70 [len=65]
<20100806005751.268 RX>
<LF>
```

Power information request (current)

This request allows you to get a current (actual) power reading from a specific Circle.
Using a special formula the result translates to a current watt usage.

Send -> 0012000D6F00002366BB338B
Receive -> 001324BD000D6F00002366BB00020013000000AD00000000000A7FCA

The send request consists of the request code ‘0012’, the MAC address of the Circle ‘000D6F00002366BB’ and the CRC checksum value ‘338B’, the response is described in the table:

Data	Datatype	Explanation
0013	Integer	This is the response code for the power information request.
24BD	Integer	This is the sequence number associated with the request/response.
000D6F00002366BB	Unsigned 64bit integer	This is the MAC address of the Circle.
0002	16bit integer	This is the number of pulses based on consumption at a 1 second interval.
0013	16bit integer	This is the number of pulses based on consumption at a 8 second interval.
000000AD	32bit integer	This are the total number of pulses.
0000	Unsigned 16bit integer	?? I suspect this is related to

		production of energy.
0000	Unsigned 16bit integer	?? I suspect this is related to production of energy.
000A	Unsigned 16bit integer	?? I suspect this is related to production of energy.
7FCA	Integer	CRC16 Checksum.

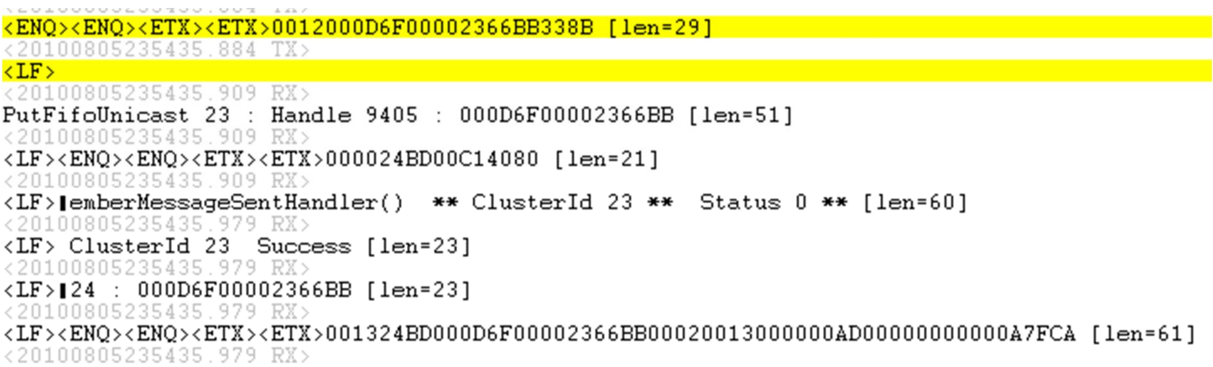
To calculate the amount of watt used, the pulses first need to be corrected based upon the calibration. Here’s a python routine I use to do the correction:

```
    def pulsecorrection(self, pulses, timespansource, timespantarget,
gain_a, \
                      gain_b, offtot, offnoise):
        """
        Corrects pulses based on calibration information, and time elapsed.
        """
        if pulses == 0.0:
            return 0.0

        corrected = 0.0
        value = pulses / timespansource
        out = timespantarget * (((pow(value + offnoise, 2.0) * gain_b)\
                                   + ((value + offnoise) * gain_a)) + \
                                   offtot)
        return out
```

After the pulses have been corrected you can convert the correct pulses to watt using the following helper functions (python again):

```
    def pulsetowatt(self, pulses):
        """
        Converts pulses to the watt unit.
        """
        return(self.pulsetokwh(pulses) * 1000)

    def pulsetokwh(self, pulses):
        """
        Converts pulses to the kWh unit.
        """
        return (pulses / 3600.0) / 468.9385193;
```

Here’s an example power request from a serial port sniffer:



```
<ENQ><ENQ><ETX><ETX>0012000D6F00002366BB338B [len=29]
<20100805235435.884 TX>
<LF>
<20100805235435.909 RX>
PutFifoUnicast 23 : Handle 9405 : 000D6F00002366BB [len=51]
<20100805235435.909 RX>
<LF><ENQ><ENQ><ETX><ETX>000024BD00C14080 [len=21]
<20100805235435.909 RX>
<LF>▮emberMessageSentHandler()  ** ClusterId 23 **  Status 0 ** [len=60]
<20100805235435.979 RX>
<LF> ClusterId 23  Success [len=23]
<20100805235435.979 RX>
<LF>▮24 : 000D6F00002366BB [len=23]
<20100805235435.979 RX>
<LF><ENQ><ENQ><ETX><ETX>001324BD000D6F00002366BB00020013000000AD00000000000A7FCA [len=61]
<20100805235435.979 RX>
```

Device information request

The device information request can be used to get general information about the Plugwise device. This includes one very important piece of information which is used to obtain power buffer information.

Send -> 0023000D6F00002366BB231B
Receive -> 00240170000D6F00002366BB0A082BBC0005205001850000047300074AA66380012A6E

The send request consists of the request code ‘0023, the MAC address of the Circle ‘000D6F00002366BB’ and the CRC checksum value ‘231B, the response is described in the table:

Data	**Datatype**	**Explanation**
0024	Integer	This is the response code for the device information request.
0170	Integer	This is the sequence number associated with the request/response.
000D6F00002366BB	Unsigned 64bit integer	This is the MAC address of the Circle.
0A	Byte	This is the year of the Circle internal clock, in hexadecimal format (it’s Y2000 based) To get the current date you need to add 2000.
08	Byte	This is the month of the Circle internal clock, in hexadecimal format.
2BBC	Unsigned 16bit integer	These are the amount minutes of the Circle internal clock, in hexadecimal format.
00052050	32bit integer	This is the current log address, this one is important to get power buffer information. Convert this hexadecimal value to integer and use the following math, to get the log address in the same format like in the Plugwise MDB file: (logaddress - 278528) / 32
01	Boolean	This value indicates the power state of the Circle (on or off)
85	Byte	This indicates the amount of herz the module operates on (85 hexadecimal appears to be 50hz, this value never changes)
000004730007	String	This string represents the hardware version of the Circle. In my case: 0000-0473-0007 (cross checked with MDB)

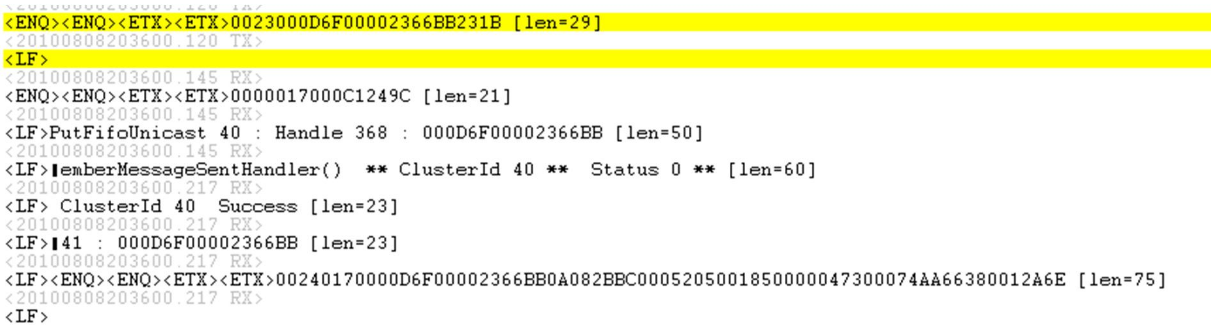
4AA66380	32bit integer	This value represents the firmware version of the Circle. The value is a timestamp (Unix Epoch)
01	Byte	??

The following functions can be helpful:

```
    def logaddresstoint(self, logaddress):
        """
        Converts plugwise log address to integer.
        """
        return (logaddress - 278528) / 32

    def deviceinforesponse(self, response):
        """
        Handles plugwise general device information response.
        """
        if len(response) != 70 or response.startswith(self.DEVINFORESPONSE)
== False:
            print "invalid device information response"
        else:
            #print response
            macaddress  = response[8:24]
            year        = self.hextoint(response[24:26]) + 0x7d0
            month       = self.hextoint(response[26:28])
            minutes     = self.hextoint(response[28:32])

            logaddress  =
self.logaddresstoint(self.hextoint(response[32:40]))
            powerstate  = self.hextoint(response[40:42])
            #herz        = self.determinehz(response[42:44])
            hwversion   = "%s-%s-%s" % (response[44:48], response[48:52],\
                                        response[52:56])

            firmware     =
datetime.datetime.utcfromtimestamp(self.hextoint(response[56:64]))

            for device in self.devices:
                if device.address == macaddress:
                    device = device

            status = False
            if powerstate == 1:
                status = True
            elif powerstate == 0:
                status = False

            args = [device.id, status]
            self.router.sendcommand("update_status", args, "database")

            print logaddress

#            device =
PlugwiseDevices.selectBy(address=macaddress).limit(1)[0]
#
            # update buffer information
            if (device.lastlogaddress < logaddress):
```

```
            if device.lastlogaddress == None:
                lastlogaddress = 0
            else:
                lastlogaddress = device.lastlogaddress

            for i in range(lastlogaddress+1, logaddress+1):
                self.get_powerbuffer(str(device.address), i)

        self.waitreply = False
```

Here’s an example device information request from a serial port sniffer:



```
<20100808203600.120 TX>
<ENQ><ENQ><ETX><ETX>0023000D6F00002366BB231B [len=29]
<20100808203600.120 TX>
<LF>
<20100808203600.145 RX>
<ENQ><ENQ><ETX><ETX>0000017000C1249C [len=21]
<20100808203600.145 RX>
<LF>PutFifoUnicast 40 : Handle 368 : 000D6F00002366BB [len=50]
<20100808203600.145 RX>
<LF>▮emberMessageSentHandler()  ** ClusterId 40 **  Status 0 ** [len=60]
<20100808203600.217 RX>
<LF> ClusterId 40  Success [len=23]
<20100808203600.217 RX>
<LF>▮41 : 000D6F00002366BB [len=23]
<20100808203600.217 RX>
<LF><ENQ><ENQ><ETX><ETX>00240170000D6F00002366BB0A082BBC0005205001850000047300074AA66380012A6E [len=75]
<20100808203600.217 RX>
<LF>
```

Power buffer information

The Plugwise Circle holds an internal buffer with information about power usage in the past. Of course we can read this historic power information.

Send -> 0048000D6F00002366BB00044020167E
Receive ->
0049016C000D6F00002366BB0000338C0000001D0000338D0000001D0000338E000000220000338F0000001A00044020B020

The send request consists of the request code ‘0048, the MAC address of the Circle ‘000D6F00002366BB’, the log address.. and the CRC checksum value ‘167E.

The log address can be calculated using the same formula as described in the last request (Device information) but then reversed..

Logaddress = (logaddress + 278528) * 32

Each response contains four hours of information (4 buffers), the response is described in the table:

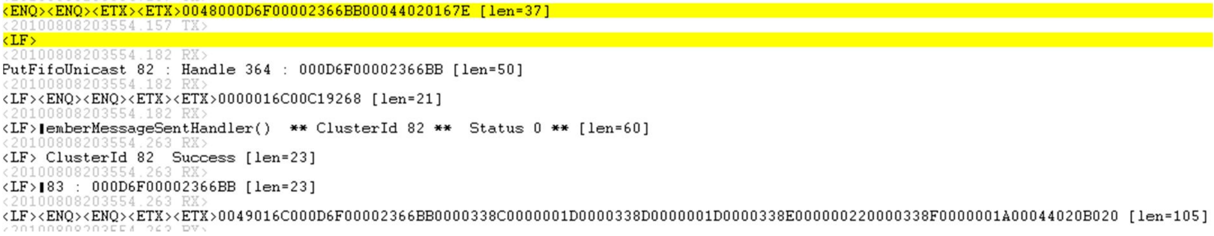
Data	**Datatype**	**Explanation**
0049	Integer	This is the response code for the power buffer information request.
016C	Integer	This is the sequence number associated with the request/response.
000D6F00002366BB	Unsigned 64bit integer	This is the MAC address of the Circle.

0000338C	32bit integer	This is the first logdate. See the helpful functions how to convert this logdate to a feasible value.
0000001D	32bit integer	This is the pulses information for the associated hour, use the methods previously described to convert this to a feasible value (in kWh)
0000338D	32bit integer	This is the second logdate. See the helpful functions how to convert this logdate to a feasible value.
0000001D	32bit integer	This is the pulses information for the associated hour, use the methods previously described to convert this to a feasible value (in kWh)
0000338E	32bit integer	This is the third logdate. See the helpful functions how to convert this logdate to a feasible value.
00000022	32bit integer	This is the pulses information for the associated hour, use the methods previously described to convert this to a feasible value (in kWh)
0000338F	32bit integer	This is the fourth logdate. See the helpful functions how to convert this logdate to a feasible value.
0000001A	32bit integer	This is the pulses information for the associated hour, use the methods previously described to convert this to a feasible value (in kWh)
00044020	32bit integer	This is the log address associated with the buffer information.
B020	Integer	CRC16 Checksum.

The following functions can be helpful:

```
    def clockinfotodatetime(self, year, month, minutes):
        """
        Converts plugwise device date and time information to pythonic
datetime.
        """
        time = datetime.datetime(2000, 1, 1, 0, 0, 0, tzinfo=tzutc())

        if year > 2000:
            year = year - 2000
        time += relativedelta(months=+month-1, years=+year,
minutes=+minutes, hours=-1)
        return time
```

Here’s an example power buffer information request from a serial port sniffer:



```
<ENQ><ENQ><ETX><ETX>0048000D6F00002366BB00044020167E [len=37]
<20100808203554.157 TX>
<LF>
<20100808203554.182 RX>
PutFifoUnicast 82 : Handle 364 : 000D6F00002366BB [len=50]
<20100808203554.182 RX>
<LF><ENQ><ENQ><ETX><ETX>0000016C00C19268 [len=21]
<20100808203554.182 RX>
<LF>▮emberMessageSentHandler()  ** ClusterId 82 **  Status 0 ** [len=60]
<20100808203554.263 RX>
<LF> ClusterId 82  Success [len=23]
<20100808203554.263 RX>
<LF>▮83 : 000D6F00002366BB [len=23]
<20100808203554.263 RX>
<LF><ENQ><ENQ><ETX><ETX>0049016C000D6F00002366BB0000338C0000001D0000338D0000001D0000338E000000220000338F0000001A00044020B020 [len=105]
<20100808203554.263 RX>
```