MPPT100 Log Format

Version 1.15

Table of Contents

1 — Definitions....3

2 — Retrieving a log....4
2.1 — Get log data by index (DataRequest)....4
2.1.a — Response format....5
2.1.b — Data format....5
2.2 — Get information about the log state (InfoRequest)....6
2.2.a — Response format....6

3 — Event Log....7
3.1 — EventEntryHeader format....7
3.2 — Converting entry to human-readable form....7
3.3 — Parsing fields....7

4 — Daily Log....8
4.1 — Daily Log structure....8
4.1.a — GenStar....9
4.1.b — BrightStar....10
4.2 — Daily Log example code....11
4.3 — Daily Log examples, in hex and base64....12
4.3.a — Example 1....12
4.3.b — Example 2....12

5 — Hourly Log....13
5.1 — Hourly Log structure....13

6 — Example code....14
6.1 — GetEntryLength()....14
6.2 — IsIgnorable()....14
6.3 — SeparateAlignedData()....14
6.4 — SeparateUnalignedData()....14

7 — Example supplement objects....15
7.1 — EventSourceNameMap....15
7.2 — EventEntryTypeList....15
7.3 — LogBaseTypeList....15

8 — Get Daily Log data through Modbus....16

9 — Editing this document....17

10 — Version history....17

1 — Definitions

All integers are little endian and unsigned unless otherwise noted.

EventEntry	An entry in the event log. Consists of an `EventEntryHeader` and zero or more `EventFields`, or is a special `EventEntry` (described in section). If non-special, can be interpreted with an `EventEntryType`.
EventEntryHeader	Combination of `LogDataIndicator`, `LogTimestamp`, `EventID`, and `EventSource`: in total 7 bytes. Last two bytes are the combination `(EventSource + (EventID << 4))`.
EventEntryType	An `EventSource`, `EventID`, name, description, and array of `EventFieldType` that describes a particular event.
EventField	A meaningful piece of data in an `EventEntry` which can be interpreted with an `EventFieldType`.
EventFieldType	A name, description, and the name of an `LogBaseType`.
EventID	Indicator of a particular event from a subsystem. Stored with 12 bits.
EventSource	Indicator of which subsystem an event is from. Stored with 4 bits.
Frame	A block of log data, aligned to log address space. Log entries are guaranteed to not cross frame bounds. Currently defined as 2048 bytes for Event Log and Hourly Log, and 512 bytes for Daily Log; so for Event Log, frame 1 begins at address `0x0`, frame 2 begins at `0x800`, and so on.
LogBaseType	A name and function to parse data to a human-readable string.
LogDataIndicator	Single byte; indicates if data is beginning of entry and length of said entry.
LogTimestamp	32-bit integer, 1 second granularity, epoch Jan. 1 2000 00:00:00. Already adjusted for timezone.
LogVersion	Event log, hourly log, daily log are all `0x00010000`, but everything that is needed to support version `0x00020000` is in this document, so a client should accept both version numbers.

2 — Retrieving a log

Log is retrieved by posting a text request to an URL and receiving binary response back in the form of `LogHeader` + data. `LogHeader` is 16 bytes long, data can have varying length.

All log requests are posted to the URL `http://mppt100.url/log`; all previous URLs issued are deprecated.

To select which log to retrieve from, use `LogIdentifier = 0` for event log, `1` for daily log, and `2` for hourly log.

2.1 — Get log data by index (DataRequest)

These parameters should be filled in using parameters received from a previous response (see 2.1.a), or by a frame-aligned request built

Post contents

```
[LogIdentifier], 0, [LastIndex], [BootCount],
[maximum_bytes_to_retrieve]
```

The first three parameters should be copied from a previous response to a `DataRequest` or `InfoRequest`. (The very first request will have to be an `InfoRequest`.)

When performing a second `DataRequest` based on the response of a first `DataRequest`, in other words plugging `LastIndex` from the response into the new request, the data retrieved will be continuous with the data received from the original, first request.

But, if no new data is available, only an up-to-date `LogHeader` is retrieved.

If the data being requested has been overwritten by the logger before the request is received, it will fail. Same if you pick an index that hasn't been written yet.

Indexes aligned by `FrameSize` are guaranteed not to start in the middle of a log entry; for example, if `FrameSize` is 512, then parsing can begin from scratch at indexes 0, 512, 1024, 2048, …

2.1.a — Response format

Response consists of a header, immediately followed by log data.

`LogVersion`	4 bytes	Log format version. Client should refuse data if it is not the version shown in definitions.
`LastIndex`	8 bytes	Exclusive end address of data retrieved. (For example, if data length is `0x300` bytes and `LastIndex` is `0x2000`, data retrieved is bytes `0x1D00` through `0x1FFF` in logger address space).
`BootCount`	4 bytes	Number of times that MultiWave has rebooted in total.

Using the info from this header format, one can make another request to continue downloading the data directly following that of the response. The request will be refused if there is a chance of discontinuity.

2.1.b — Data format

Data is a bytestream of log entries and unused space. Remember, a request with a frame-aligned index is guaranteed to return data starting with either unused space or the beginning of a log entry, but otherwise there is no such guarantee and the data may start in the middle of a log entry.

Similarly, the last entry in any response is not guaranteed to be complete, until the following request returns no data and just a header (meaning we're caught up with the log). One should store the remainder data of any incomplete entry stored so it can be prepended to the new data.

A bytestream which begins with the beginning of a complete log entry is referred to as **coherent**.

To separate the data of a coherent bytestream into individual entries, take the first byte, which is an `LogDataIndicator`. If the value of the byte is `0x00` or `0xFF` it is unused space and should be ignored; otherwise it is a complete log entry and the value is equal to the length of the entry.

If the length is 1 then the entry has no additional data and it indicates that a buffer overflow happened and some log data was lost, which should be indicated to the user. Otherwise, if the length is less than 7, it is a special entry and can be ignored. Finally, if the length is 7 or more, then it is a regular entry and should be parsed. Repeat until there is no more data; store last entry if incomplete.

For pseudocode showing how to separate data from bytestream into individual entries, see `SeparateAlignedData()` and `SeparateUnalignedData()`.

2.2 — Get information about the log state (InfoRequest)

Post contents

```
[LogIdentifier], 1
```

2.2.a — Response format

`LogVersion`	4 bytes	Log format version. Client should refuse data if it is not the version shown in definitions.
`LastIndex`	8 bytes	Exclusive end index of most recent log data.
`BootCount`	4 bytes	Bootcount of device.
`EarliestIndex`	8 bytes	Exclusive end index of earliest known log data.
`FrameSize`	4 bytes	Size of a frame in bytes.
`TotalNumOfFrames`	4 bytes	Total number of frames in this log's address space.

Using the `EarliestIndex` value in a `DataRequest` will get data starting from as far back as possible. Using the `LastIndex` value in a `DataRequest` will get data that has been written since the `InfoRequest` was made. Using a value in between those two will work, but isn't guaranteed to retrieve a coherent bytestream unless it's aligned to `FrameSize`.

3 — Event Log

A (regular) `EventEntry` consists of an `EventEntryHeader` and zero or more `EventFields`.

Three files, a map of `EventSources` (all numbers mapped to names, all names unique), a list of `EventEntryTypes` (not all source+ID combinations mapped to names, all names unique), and a list of `LogBaseTypes` are required to parse entries to human-readable form.

3.1 — EventEntryHeader format

`length`	1 byte	Length of entire event including all fields.
`timestamp`	4 bytes	See definition for `LogTimestamp`.
`eventSource + (eventID << 4)`	2 bytes	See definitions for `EventSource` and `EventID`.

3.2 — Converting entry to human-readable form

Retrieve items from header. Display `timestamp` in user's preferred form. Display `eventSource` by looking up its name from map. Then, look up `EventEntryType` corresponding to both source and ID. If it cannot be found, display name "Unknown: (`eventID` in decimal)" and show data after header in hexadecimal. If type was found, store it as `curEntryType`, store data after header as `field_byte_array`, and continue as follows:

- Display name `curEntryType.name`
- Display description `curEntryType.description`
- Iterate through array `curEntryType.fieldTypes`, parse each field in turn and display it. If no data is left but `fieldTypes` is still not done, do not error; the entry was probably stored at a lower detail level so not all fields were recorded.

3.3 — Parsing fields

- Display field name `fieldType.name`
- Display field description `fieldType.description`
- Look up field base type by name `fieldType.baseTypeName`
- Call `fieldBaseType.parse(field_byte_array)`; result will be an object of the form `{ parsed:"Display value for this field", bytes_eaten:4 }`
- Display field value `result.parsed`, remove `result.bytes_eaten` bytes from `field_byte_array`

4 — Daily Log

4.1 — Daily Log structure

Combined values are listed from high to low in **Field Description**, with respective # of bits in **Units**.

So, “Tb_max_min” has maximum as top (most significant) 8 bits, minimum as bottom (least significant) 8 bits.

The first entry is the “flags” field, which is variable length in 16-bit words; bits 0-14 indicate which fields appear in the log entry, bit 15 indicates whether another flag word continues. So, if bit 15 is set, read another 15 bits of flags, then check bit 31 to see if flags continue, etc.

There is no versioning system built into the log. From version 1.12 of the spec onwards, we won’t resize any existing fields, and any new fields will be in ascending order of bits: therefore, a parser which doesn’t recognize a flag bit should just ignore it, and should throw away unrecognized data at the end of an entry. There is no requirement to signal to the user “this entry was partially unparseable”.

Note: For the "Time_in_regulation” field, the “Rest” value is not used on the MPPT100 and will always be recorded as 0; this should not be shown to the user. The user should only see values for “Eq”, “Absorb”, and “Float”.

Note: All signed integers (Int64, Int32, Int16, Int8) are represented in two’s complement form. So, for an Int32, “0x80000000” means “-2,147,483,648”, “0xFFFFFFFF” means “-1”, “0x0” means 0, and “0x7FFFFFF” means “ 2,147,483,647”.

Note: “Control_reset” flag has no associated data, and the presence of the flag alone indicates that the control has reset in between records.

4.1.a — GenStar

Bytes	Flag bit	Field name	Field description	Units	Scaling or Range
2*F	*	Flags	See above		
4	*	Timestamp	Timestamp in seconds; epoch at Jan. 1 2000	seconds	0 to (2^{32}-1)
2	*	Vb_min	Battery voltage — minimum	V	Float16
2	*	Vb_max	Battery voltage — maximum	V	Float16
2	0	Varray_max	Array voltage — maximum	V	Float16
4	1	Net_batt_Ah	Battery net amp-hours	Ah	Float32
4	2	Charge_kWhr	Charge kilowatt-hours	kWhr	Float32
4	3	Charge_Ah	Charge amp-hours	Ah	Float32
4	4	Load0_Ah	Load 0 amp-hours	Ah	Float32
6	5	Time_in_regulation	Time in Eq, Absorb, Float and Rest (**see note**)	minutes[4]	0 to (2^{12}-1)
2	6	Tb_max_min	Batt. temperature maximum and minimum	C[2]	Int8
4	7	Net_batt_system_Ah	Battery net amp-hours (system)	Ah	Float32
4	8	Charge_system_kWhr	Charge kilowatt-hours (system)	kWhr	Float32
4	9	Charge_system_Ah	Charge amp-hours (system)	Ah	Float32
4	10	Load_system_Ah	Load amp-hours (system)	Ah	Float32
8	11	Alarm_system	Daily alarm reports for system	Bitfield	
8	12	Fault_system	Daily fault reports for system	Bitfield	
2	13	Fault_charge	Daily fault reports for charge	Bitfield	
2	14	Fault_load0	Daily fault reports for load 0	Bitfield	
4	15	Fault_loadSummary	Daily fault reports for all loads	Bitfield	
2	16	Fault_powerSupply	Daily fault reports for power supply	Bitfield	
2	17	Fault_powerStage	Daily fault reports for power stage	Bitfield	
4	18	Fault_block	Daily fault reports for blockbus	Bitfield	
4	19	Shunt0_Ah	Shunt 0 (shown as 1A) amp-hours	Ah	Int32
4	20	Shunt1_Ah	Shunt 1 (shown as 1B) amp-hours	Ah	Int32
4	21	Shunt2_Ah	Shunt 2 (shown as 2A) amp-hours	Ah	Int32
4	22	Shunt3_Ah	Shunt 3 (shown as 2B) amp-hours	Ah	Int32
4	23	Shunt4_Ah	Shunt 4 (shown as 3A) amp-hours	Ah	Int32
4	24	Shunt5_Ah	Shunt 5 (shown as 3B) amp-hours	Ah	Int32
2	25	SOC_min	Battery state of charge, minimum	0.0-1.0	Float16
2	26	SOC_max	Battery state of charge, maximum	0.0-1.0	Float16
0	27	Control_reset	Flag only: this is first log written since reset	N/A	N/A

4.1.b — BrightStar

Bytes	Flag bit	Field name	Field description	Units	Scaling or Range
2*F	*	Flags	See above		
4	*	Timestamp	Timestamp in seconds; epoch at Jan. 1 2000	seconds	0 to (2^{32}-1)
2	*	Vb_min	Battery voltage — minimum	V	Float16
2	*	Vb_max	Battery voltage — maximum	V	Float16
2	0	Varray_max	Array voltage — maximum	V	Float16
4	1	Net_batt_Ah	Battery net amp-hours	Ah	Float32
4	2	Charge_kWhr	Charge kilowatt-hours	kWhr	Float32
4	3	Charge_Ah	Charge amp-hours	Ah	Float32
4	4	Load0_Ah	Load 0 amp-hours	Ah	Float32
4	5	Load1_Ah	Load 1 amp-hours	Ah	Float32
4	6	Load2_Ah	Load 2 amp-hours	Ah	Float32
4	7	Load3_Ah	Load 3 amp-hours	Ah	Float32
6	8	Time_in_regulation	Time in Eq, Absorb, Float and Rest (**see note**)	minutes[4]	0 to (2^{12}-1)
2	9	Tb_max_min	Batt. temperature maximum and minimum	C[2]	Int8
4	10	Net_batt_system_Ah	Battery net amp-hours (system)	Ah	Float32
4	11	Charge_system_kWhr	Charge kilowatt-hours (system)	kWhr	Float32
4	12	Charge_system_Ah	Charge amp-hours (system)	Ah	Float32
4	13	Load_system_Ah	Load amp-hours (system)	Ah	Float32
8	14	Alarm_system	Daily alarm reports for system	Bitfield	
8	15	Fault_system	Daily fault reports for system	Bitfield	
2	16	Fault_charge	Daily fault reports for charge	Bitfield	
2	17	Fault_load0	Daily fault reports for load 0	Bitfield	
2	18	Fault_load1	Daily fault reports for load 1	Bitfield	
2	19	Fault_load2	Daily fault reports for load 2	Bitfield	
2	20	Fault_load3	Daily fault reports for load 3	Bitfield	
2	21	Fault_powerSupply	Daily fault reports for power supply	Bitfield	
2	22	Fault_powerStage	Daily fault reports for power stage	Bitfield	
4	23	Fault_block	Daily fault reports for blockbus	Bitfield	
4	24	Shunt0_Ah	Shunt 0 (shown as 1A) amp-hours	Ah	Int32
4	25	Shunt1_Ah	Shunt 1 (shown as 1B) amp-hours	Ah	Int32
4	26	Shunt2_Ah	Shunt 2 (shown as 2A) amp-hours	Ah	Int32
4	27	Shunt3_Ah	Shunt 3 (shown as 2B) amp-hours	Ah	Int32
4	28	Shunt4_Ah	Shunt 4 (shown as 3A) amp-hours	Ah	Int32
4	29	Shunt5_Ah	Shunt 5 (shown as 3B) amp-hours	Ah	Int32
2	30	SOC_min	Battery state of charge, minimum	0.0-1.0	Float16
2	31	SOC_max	Battery state of charge, maximum	0.0-1.0	Float16
0	32	Control_reset	Flag only: this is first log written since reset	N/A	N/A

4.2 — Daily Log example code

```
maxBatteryTemp(Uint16 Tb_max_min) {
      return (Tb_max_min >> 8) & 0xFF;
}

minBatteryTemp(Uint16 Tb_max_min) {
      return (Tb_max_min >> 0) & 0xFF;
}

timeInEq(Uint48 Time_in_regulation) {
      return (Time_in_regulation >> 36) & 0xFFF;
}

timeInAbsorb(Uint48 Time_in_regulation) {
      return (Time_in_regulation >> 24) & 0xFFF;
}

timeInFloat(Uint48 Time_in_regulation) {
      return (Time_in_regulation >> 12) & 0xFFF;
}

timeInRest(Uint48 Time_in_regulation) {
      return (Time_in_regulation >> 0) & 0xFFF;
}
```

4.3 — Daily Log examples, in hex and base64

4.3.a — Example 1

```
3D 1F C3 C5 00 9A F3 A7 29 FC 49 FF 49 D1 49 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 80 80 01 24 00 20 20 00 00 00 40 00 00
00 00 00 00 00 08 00 40 00 00 05

PR/DxQCa86cp/En/SdFJAAAAAAAAAAAAAAAAAAAAAAAAAAAAICAASQAICAAAABAAAAAAAAAAg
AQAAABQ==
```

Length of entry: `3D` (61 in decimal)

Flags: `1F C3 C5 00`. Flags are read as little endian 16-bit words, and if the highest bit of a flag word is set, then another word should be read. So, flags can be interpreted as:

```
0000 0000 1100 0101 1100 0011 0001 1111
   0    0    C    5    C    3    1    F
```

Therefore, the included fields are: Timestamp (always), Vb_min (always), Vb_max (always), Varray_max (0), Net_batt_Ah (1), Charge_kWhr (2), Charge_Ah (3), Load0_Ah (4), Time_in_regulation (8), Tb_max_min (9), Alarm_system (14), Fault_system (15), Fault_load0 (17), Fault_powerSupply (21), Fault_powerStage (22).

Timestamp: `9A F3 A7 29` (698872730)
Vb_min: `FC 49` (11.97)
Vb_max: `FF 49` (11.99)
Varray_max: `D1 49` (11.63)
Tb_max_min: `80 80` (0.0, 0.0)
Alarm_system: `01 24 00 20 20 00 00 00` (Alarms 0, 10, 13, 29, 37)
Fault_system: `40 00 00 00 00 00 00 00` (Fault 6)
Fault_load: `08 00` (Fault 3)
Fault_powerSupply: `40 00` (Fault 6)
Fault_powerStage: `00 05` (Faults 8 and 10)

(all values not listed are 0)

4.3.b — Example 2

```
0B 00 00 9A F3 A7 29 FC 49 FF 49

CwAAmvOnKfxJ/0k=
```

Length of entry: `0B` (11 in decimal)

Flags: `00 00`. This is a minimal entry with no optional fields. It just contains Timestamp, Vb_min, and Vb_max. The values are the same as Example 1.

5 — Hourly Log

5.1 — Hourly Log structure

LogBaseType	Name of field
uint32	Timestamp
float16	Vb_min_hourly
float16	Vb_max_hourly
float16	Batt_soc_min_hourly
float16	Batt_soc_max_hourly
float32	Ah_net_hourly
float32	Wh_AC_out_hourly

6 — Example code

Example Python-esque pseudocode for methods in this documentation.

6.1 — GetEntryLength()

```
unsigned GetEntryLength(byte LogDataIndicator):
  if (LogDataIndicator == 0x0) or (LogDataIndicator == 0xFF):
    return 1
  else:
    return LogDataIndicator # entry length includes this byte
```

6.2 — IsIgnorable()

```
bool IsIgnorable(byte LogDataIndicator):
  # 0x00 and 0xFF are unused space.
  # Entry length < 7 means a special event, which can be ignored aside from
  # buffer overflow.
  if LogDataIndicator != 1 and LogDataIndicator < 7:
    return true
  elif LogDataIndicator == 0xFF:
    return true
  else:
    return false
```

6.3 — SeparateAlignedData()

```
array<array<byte>> SeparateAlignedData(array<byte> data):
  global array<byte> remainder # place to put incomplete entry
  array<array<byte>> separated_entries = []
  while data.length > 0:
    if IsIgnorable(data[0]):
      data = data[GetEntryLength(data[0]):] # python slice notation
      continue
    current_entry_length = GetEntryLength(data[0])
    if current_entry_length > data.length:
      # last entry is not complete; save in remainder
      remainder = data
      return separated_entries
    separated_entries.append(data[0:current_entry_length])
    data = data[current_entry_length:]
  remainder = [] # loop completed without spillover, so no remainder
  return separated_entries
```

6.4 — SeparateUnalignedData()

```
array<array<byte>> SeparateUnalignedData(array<byte> data):
  global array<byte> remainder # possible incomplete entry is here
  data = remainder + data # prepend remainder to data!!
  return SeparateAlignedData(data) # can reuse this code
```

7 — Example supplement objects

7.1 — EventSourceNameMap

```
{
  { source:0,  name:"RESERVED0"  }, { source:1,  name:"BATTERY"    },
  { source:2,  name:"FAULT"      }, { source:3,  name:"ALARM"      },
  { source:4,  name:"NETWORK"    }, { source:5,  name:"AGC"        },
  { source:6,  name:"COMMAND"    }, { source:7,  name:"CONFIG"     },
  { source:8,  name:"RESERVED8"  }, { source:9,  name:"RESERVED9"  },
  { source:10, name:"RESERVED10" }, { source:11, name:"RESERVED11" },
  { source:12, name:"RESERVED12" }, { source:13, name:"TESTING"    },
  { source:14, name:"SYSTEM"     }, { source:15, name:"RESERVED15" }
}
```

7.2 — EventEntryTypeList

```
{
  {
    source:2, // FAULT
    id:7,
    name:"BHVD",
    description:"Battery high voltage disconnect",
    fieldTypes:
    {
      { name:"V", description:"Voltage reading", baseTypeName:"uint32" },
      { name:"q", description:"Another field",   baseTypeName:"float" },
      ...
    }
  },
  ...
}
```

7.3 — LogBaseTypeList

```
{
  {
    name:"uint32",
    parse:(b) => {
      if (b.length < 4) throw "Insufficient length"
      x = (b[0]) | (b[1] << 8) | (b[2] << 16) | (b[3] << 24)
      return { parsed:x.toString(), eaten: 4 }
    }
  },
  ...
}
```

8 — Get Daily Log data through Modbus

Note: the offsets here are out of date due to Daily Log entries changing size for MPPT100. They will be updated soon, but the principles here remain the same.

The MPPT100 can retrieve up to 107 days' worth of Daily Log data over Modbus. This data is represented as a circular buffer where the oldest data is over-written by the newest data.

- The logged data is mapped from 0x4000-0x4FE1 inclusive.
- The data consists of up to 107 blocks of data.
- Each block is 76 bytes (38 modbus variables.)
- Ignore blocks with timestamps of 0x000000 or 0xFFFFFF.
- The index of the earliest entry in the log can be found at MBVAR_Elog_index (0x220D). The address of the earliest entry can be calculated: (ELog_index * 38 + 16384)

PDU Addr	Logical Addr	Variable name	Variable description
0x4000-4025	16384-16421	logger[0]	
…	…		
0x4FBC-4FE1	20412-20449	logger[106]	

Aside from the Modbus protocol itself, data is in little endian format; when a value is stored in more than one Modbus variable (larger than two bytes), the Modbus variable with the lowest address should be read as the least significant value; as well, the "packed" structure format should be read as little endian.

But, the two bytes **within** each individual Modbus variable are still transmitted in big endian order, in accordance with the protocol.

So for the log entry starting at 0x4EB2, or index 99:

PDU Addr	Relative Addr	Variable description
0x4EB2	0x0	Least significant two bytes of timestamp
...	...	
0x4EB8	0x6	Least significant two bytes of the AC1_kWhr floating point value
0x4EB9	0x7	Most significant two bytes of the AC1_kWhr floating point value
...	...	
0x4ED7	0x25	End of system alarm value

9 — Editing this document

This file is a hybrid PDF, which has embedded within it the original ODF markup. It can be opened and edited in LibreOffice without any additional conversion steps.

10 — Version history

Major version changes are breaking changes to format.

The latest additions to the document are colored blue.

1.15	2023-10-05	Daily log: summarize load fault reporting on GenStar, remove Load1-3_Ah
1.14	2022-10-26	Daily log: add fields SOC min, SOC max, Control Reset.
1.13	2022-08-03	Corrected descriptions of Shunt Ah and Batt. Temp fields: they are both signed integers, not unsigned. Add note about how signed integers are represented.
1.12	2022-05-18	Add Shunt 0 – Shunt 6 fields to daily log; remove “rest” value; add clarifications on parsing.
1.11	2022-03-03	Elaborated the daily log examples.
1.10	2022-03-02	Add daily log examples. Change “low endian” to “little endian”.
1.9	2021-10-18	Daily log variable length.
1.8	2021-09-16	Changed references to MPPT100, updated Daily Log again.
1.7	2020-10-08	Updated Daily Log to include all faults. Changed some wording, fixed pseudo-hexadecimal. Un-bumped LogVersion since nothing is broken yet.
1.6	2020-09-17	Hide wraps field from client, LogVersion bumped.
1.5	2020-09-01	Add Blockbus fault field to daily log.
1.4	2020-04-01	Update Daily Log description, add info on retrieving daily log from Modbus.
1.3	2019-08-20	Add timestamp to Hourly Log, more detail in daily log format.
1.2	2019-06-25	Add information for daily and hourly log.
1.1	2019-04-26	Fixed bug in SeparateFrameRequestResponse() pseudocode.
1.0	2018-08-22	Original.