Logging Format Standard
Boulder Engineering Studio Internal

Date	*Author*	*Notes*
2016-01-15	Austin Glaser	Initial Draft
2016-03-12	Austin Glaser	Add 'Runtime Error' log type
2016-03-14	Austin Glaser	Modify 'Entry Length' field definition, modify padding length field widths
2016-03-29	Austin Glaser	Add new logo, update title block

Table 0.1: Revision History.

Contents

1 Introduction

1.1 Scope This document specifies a standard data format for logging events to an arbitrary section of nonvolatile memory. It does not specify the type of memory, or the interface protocol. It also does not specify the software API used – for that, consult either a separate document or the software's internal documentation.

1.2 Purpose This data format is intended to be used as a standard across all BES projects where some form of logging is desirable, but which are not high-level enough to include an on board filesystem with high-capacity, persistent storage. It is intended to be simple to implement and consume little memory relative to a 'standard' logging protocol which saves raw text.

It's also intended to be flexible, with the ability to log certain events using as little of 16 bytes of storage space. It also provides the ability to spend more data and record more detail of program state at the time of the event, an arbitrary string, or binary data.

2 Terminology

See table 2.1 for definitions of terms used throughout this document. **TODO: There probably should be more here.**

Term	*Definition*
Entry	A piece of data in the log recording the occurrence of a particular event
Log	A section of memory set aside to store data according to this standard
Logger	A software utility which stores information to a log in accordance with this standard
Log Header	The section of the log allocated to storing log metadata

Table 2.1: Definition of terms used within this document.

All stored numbers shall be in little-endian format. All stored addresses shall be offsets from the beginning of the log memory. All lengths shall be in bytes.

3 Log Header

3.1 Description Each log shall begin with a data section encoding metadata relevant to the log as a whole, referred to as the 'log header'. This log header shall reside at the lowest addresses of the memory set aside for the log, and shall occupy 64 bytes. If the memory used for logging only supports page-sized operation, the log entries shall start at the next page boundary after the end of the header.

The log header shall contain the following information:

- Full git revision hash at the time of compilation
- Build date
- Flags:
  - Dirty (D) – shall be set if the git repository was 'dirty' at build time
  - Not Stable ($\overline{\text{S}}$) – shall be set if the firmware was not pinned at a stable release at build time. Will never be cleared when D is set

- Project ID
- Stable version (if D and $\overline{S}$ flags are both cleared)
- Log data address

The logger shall verify this information at each system reset and, if any has changed, shall update the log header *and remove all currently stored log entries.* **Everyone, is this desireable behavior?**

3.2 Data Format

See table 3.1 for a description of data layout in memory.

The *Project ID* field shall correspond to BES's internal project numbering system. The major and minor firmware version numbers shall only be considered valid if both the *D* and $\overline{S}$ flags are cleared.

The *Git Hash* field shall be the binary encoding of the full 40-character hash associated with a git commit. Since this hash is a hexadecimal number, it can be stored in 20 bytes of memory.

The *Build Timestamp* field shall be a unix epoch timestamp indicating when the software was compiled. It shall be stored in 64-bit form to avoid the 2030 rollover issue.

The *Log Start* field is the address at which valid log data starts. This may be the address immediately following the header (0x0000 0040), or it may be the beginning of the next writeable page of memory.

The *Log End* field is one more than the last valid address for the log memory region. That is, if the last address in the log region is 0x0000 0FFF, *Log End* shall be 0x000 1000.

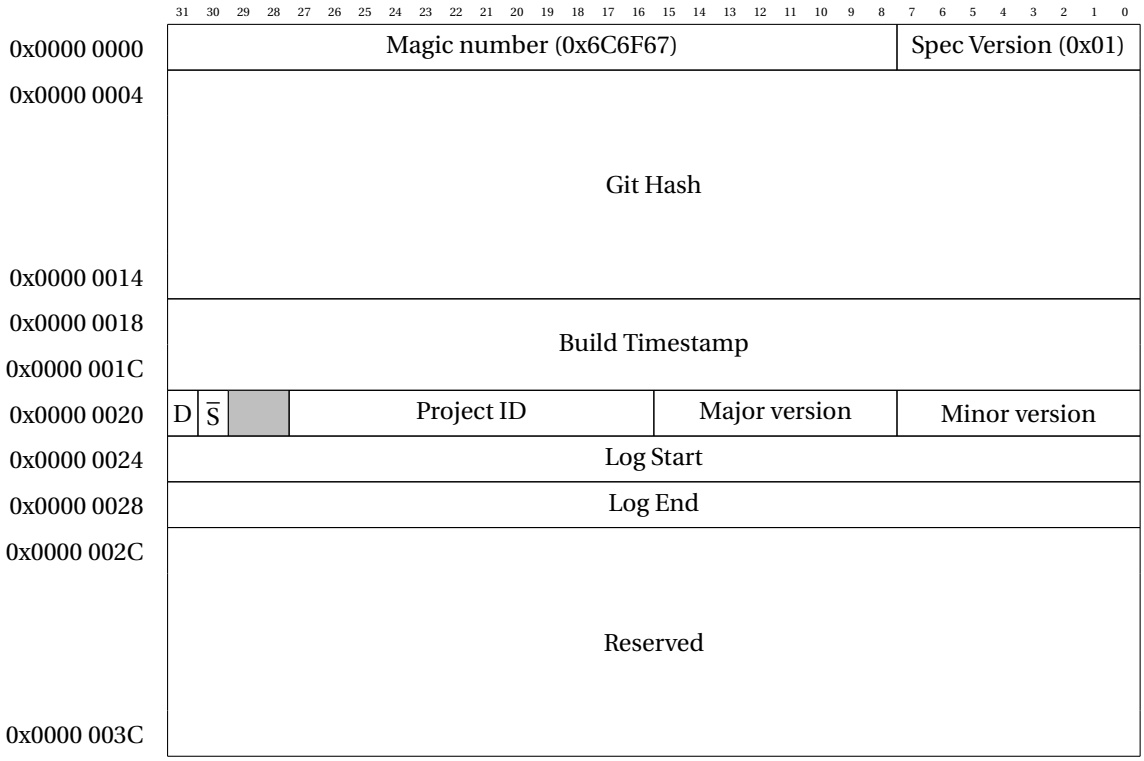

Table 3.1: Data format used to store the log header. Address values are offsets from log region base address.

4 Log Bookkeeping

4.1 Description The log shall begin at the *Log Start*, and shall extend to *Log End* (both stored in the log header, described in section 3).

The log shall function as a circular buffer. Log entries (section 5) shall be inserted into available memory sequentially, aligned to word (4-byte) boundaries.

4.2 Data Format See table 4.1 for a description of how bookkeeping data are stored in memory.

The first 16 bytes of the log shall be reserved for bookkeeping data. This consists of two pointers, *Head* and *Tail*. *Head* shall be the address of the next valid location for insertion. *Tail* shall be the address of the oldest stored log entry, or shall be equal to *Head* if no log entries are stored. When the log is empty, both *Head* and *Tail* shall be equal to *Log Start* + 16. At no time will either head or tail be less than *Log Start* + 16 or greater than or equal to *Log End*.

The bookkeeping data shall always reside at the address pointed to by *Log Start*.

New log entries shall be inserted at *Head*.

When inserting a new log entry, the logger shall first verify that there is space available. Space is available if the space between *Head* and *Tail* (wrapped around the end of the memory region) is greater than *Entry Length*.

If there is no space available, the logger shall remove log entries by moving *Tail* to the start of the first entry where this condition is met.

The logger shall then write the log entry into memory at *Insertion Point*, and shall set *Head* to the next word beyond the end of the new log entry.

In order to avoid memory corruption, the logger shall always perform these steps in the following order:

1. If needed, move *Tail* to remove old log entries and make space for the new one
2. Write the new log entry at *Head*
3. Move *Head* to its new position

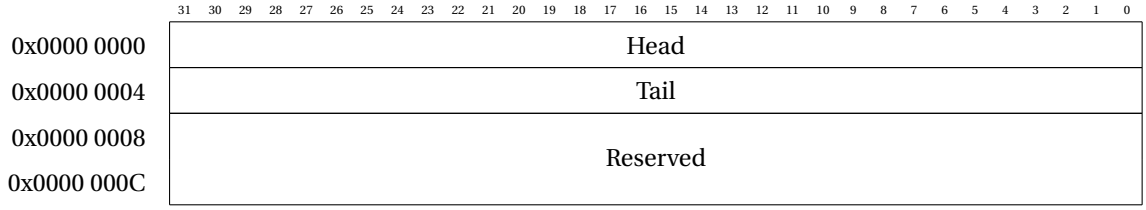

Table 4.1: Format used to record log bookkeeping data. Address values are offsets from *Log Start*.

5 Log Entry

5.1 Description A log entry is a data structure which encodes the occurrence of a single event. Unlike the other data formats, several fields are optional. At minimum, a log shall include a length, a timestamp, and a type, which will together occupy 16 bytes.

5.2 Data Format See table 5.3 for a description of a log entry's layout in memory.

The *Entry Length* shall be the total length of the log entry (including the *Entry Length* field) in bytes. *Entry Length* shall never be the reserved value 0xFFFF FFFF.

The *Timestamp* shall be either a UTC Unix epoch timestamp (if the *R* bit is cleared) or a relative (system time) timestamp from system reset (if the *R* bit is set). Absolute timestamps are preferred for systems which include an RTC.

The *Entry Type* is a field with standard log types defined in table 5.1. There is also a range of values set aside for arbitrary use. Note that all (non-extended) ASCII characters fall into the user-defined range.

Entry Type	*Field Value*	*Notes*
User defined	0x00 - 0x7F	Can be used to indicate user-defined log types
Reset	0x80	
Exception	0x81	
Runtime Error	0x82	Subtype will indicate the error type
State Change	0x83	Subtype will indicate the *entered* state
Reserved	0x84 - 0xFF	May be used in later versions of this standard

Table 5.1: Entry type codes

The *Entry Subtype* field has meaning dependent on the particular entry type; see table 5.2 for more details.

The bits *PC*, *SP*, *SU*, *DSC*, and *BIN* (collectively known as the '*Field Bits*') indicate which optional fields are present after the mandatory fields. Because these fields are optional, and because some of the fields are variable length, their offsets are non-constant. The logger shall place fields which are present in the order they are listed in table 5.3, leaving no padding where optional fields are not present. If no fields are present, the *Field Bits* shall all be cleared and *Entry Length* shall be 16.

The *BIN* member of the *Field Bits* shall be set if one or more *Binary Data* fields are present in the log entry.

The *Program Counter* field shall be used to record execution state at the time of the entry. Note that this will be the value of the program counter immediately before a call to any log function (usually retrieved by examining the link register).

The *Stack Pointer* field shall be used to record the immediate value of the stack pointer at the time of the entry.

The *Maximum Stack Usage* field shall be used to record the greatest amount of stack allocated up to the time of the entry.

If this is used on a system with multiple threads (and hence multiple stack pointers and program counters), it may be useful to omit the three preceding fields and instead record the information in several *Binary Data* fields.

The *Descriptor String* variable-length field shall begin with a length (*Descriptor String Length*) which indicates the number of bytes in *Descriptor String Body* (including padding). The length shall always be aligned to a word boundary (a multiple of 4), but the highest-order byte of the length (*DPB*) indicate the number of zero padding bytes at the end of the data block.

The binary data section shall be made up of one or more *Binary Data* variable length fields (see description below). The *Binary Data Count* field indicates the number of subsequent *Binary Data* entries.

Each *Binary Data* variable-length field shall begin with a length (*Binary Data Length*) which indicates the number of bytes in *Binary Data Body* (including padding). The length shall always be aligned to a word boundary (a multiple of 4), but the highest-order byte of the length (*BPB*) indicate the number of zero padding bytes at the end of the data block. The *Binary Data Type* field may be left zeroed, or may be used as a user-defined discriminator between different data points.

Entry Type	*Entry Subtype*	*Field Value*	*Notes*
Reset	User defined	0x00-0x7F	
	Unknown Reset	0x80	Can be used where subtype discrimination is not important
	Power Reset	0x81	
	Software Reset	0x82	
	Watchdog Reset	0x83	
	Reserved	0x84-0xFF	May be used in later versions of this standard
Exception	User defined	0x00-0x7F	
	Unknown Exception	0x80	Can be used where subtype discrimination is not important
	Stack Overflow	0x81	
	Hard Fault	0x82	
	Bus Fault	0x83	
	Usage Fault	0x84	
	Reserved	0x85-0xFF	May be used in later versions of this standard
Runtime Error	User defined	0x00-0x7F	
	Unknown Runtime error	0x80	Can be used where subtype discrimination is not important
	Invalid Log Type	0x81	Can be used by logger implementation to signal an invalid log call
	Invalid Log Subtype	0x82	Can be used by logger implementation to signal an invalid log call
	Invalid Argument	0x83	
	Buffer Overflow	0x84	
	Memory Allocation Failure	0x85	
	Reserved	0x86-0xFF	May be used in later versions of this standard
All others	User defined	0x00-0x7F	
	Reserved	0x80-0xFF	May be used in later versions of this standard

Table 5.2: Entry subtype codes

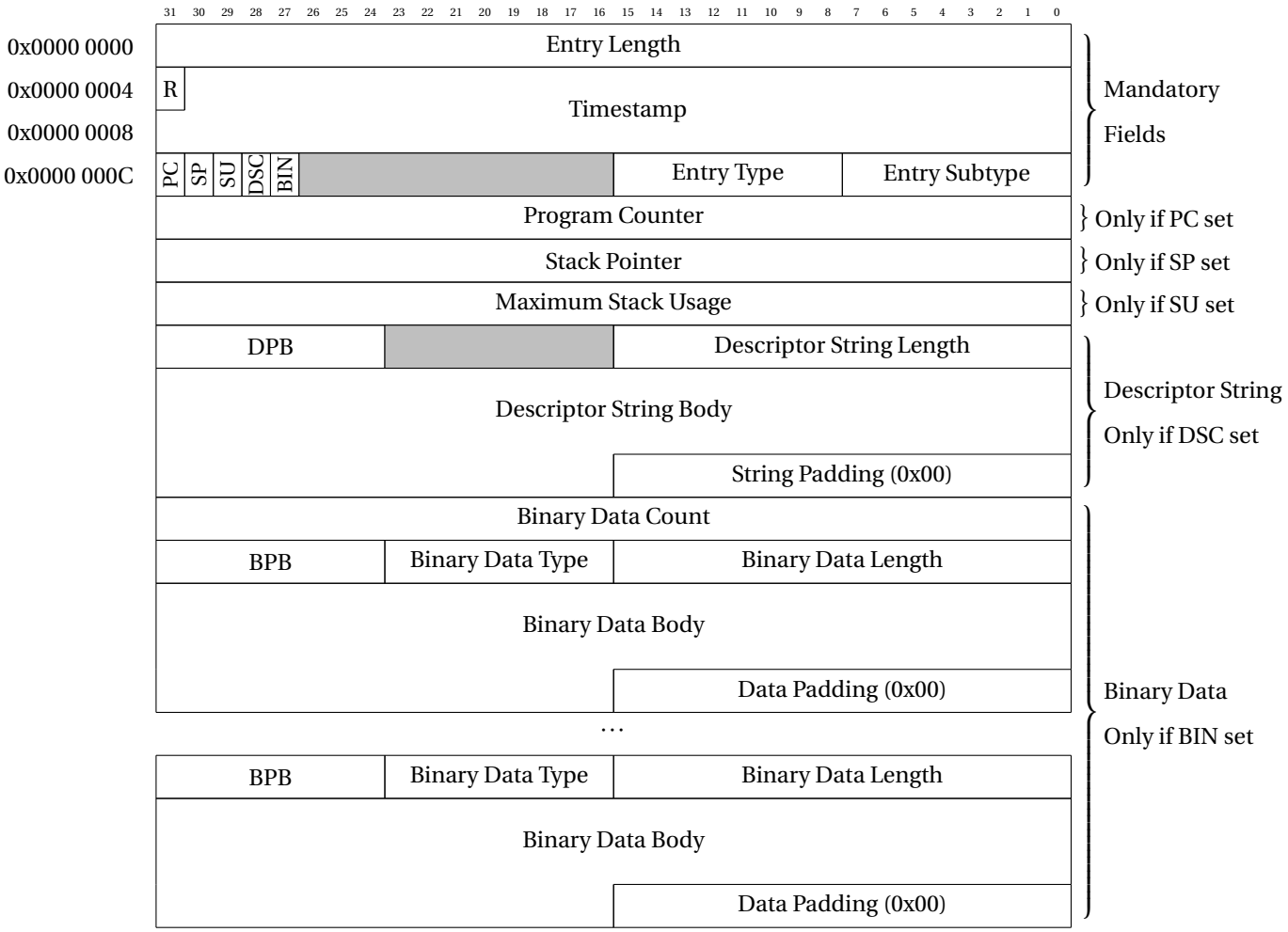

Table 5.3: Format used to store a log entry.