Multiscale Electrophysiology Data Format, Version 1.1

(MED 1.1)

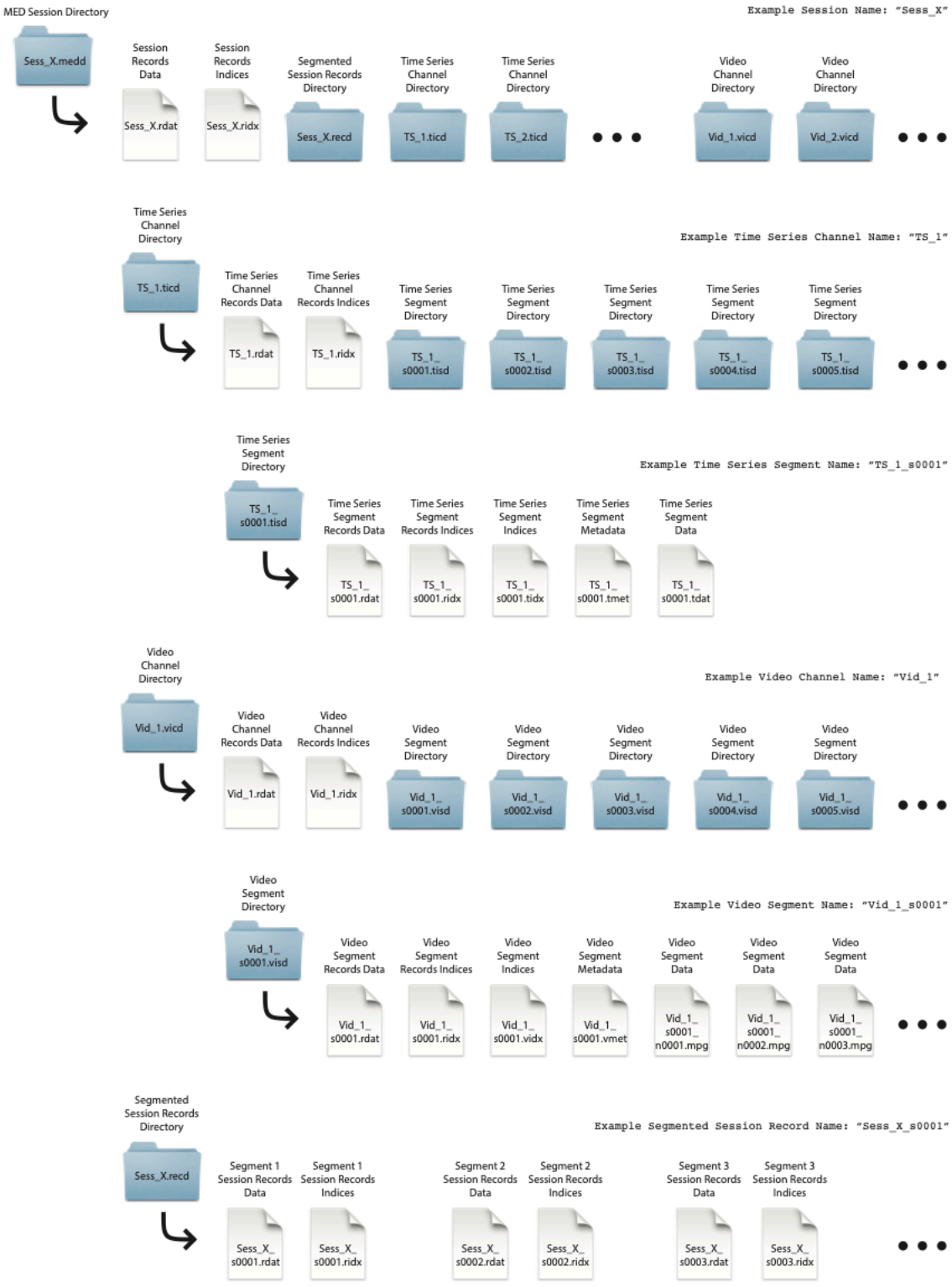
Library Naming & Tags:

- Format versions will be named as: "MED $F_{maj}.F_{min}$." where "F_{maj}" is the major format version, "F_{min}" is the minor format version.
- Libraries will be named as: "MED $F_{maj}.F_{min}.L$" where "L" is the library version for the major format version. There will be no sub-versioning of the library versions.
- MED major format versions are restricted to one digit (and thus capped at 9 under this schema). MED minor format versions start at zero, and are not restricted to one digit.
- A minor format version is guaranteed to be cross compatible with all versions in it's major category.
- Minor format versions may add fields to the format in protected regions, but no preexisting fields will be removed or moved.
- *Library Tags:* All functions, constants, macros, and data types defined in the library are tagged with the suffix "_mF_{maj}L" (or "MED format F_{maj}, library L".
- Minor format versions are not included in the tags. This is to keep them concise, and because all minor format versions are cross compatible with all versions in their major category.
- Library version numbering restarts at zero with the release of a new major format version.
- Minor Format version numbering restarts at one with the release of a new major format version.
- Cross compatibility will be maintained between major MED format versions only if practical.
- In this document, the tag "_m1x" refers to MED format version 1, any library version.
- The full library name, & thus the minor format versions it supports, will be given in the header comments.

Examples:

- "MED 1.0.1 C API" is the C API for MED format 1.0, library version 1. This API will work on all MED 1 format files, but not necessarily MED 2-n format files. The tag for this library is "_m11".
- "MED 1.1.2 C API" is the C API for MED format 1.1, library version 2. This API will work on all MED 1.0 and 1.1 format files. The tag for this library is "_m12".
- The tag "_m213" indicates "MED format version 2.0-x, library version 13". This library will work on all MED 2 format files, but not necessarily MED 1 or MED 3-n format files.
- The tags allow different versions of the MED libraries to used within the same code. One such use might be in code for a MED 1 to MED 2 converter.

Figure 1: MED Data Hierarchy & Naming Conventions



MED Data Hierarchy (See Figure 1)

- Each collection of recorded channels is called a “Session”. A session is a directory at the top level of the hierarchy.
- A session directory is not required, MED channels or segments can be acquired and used independently.
- Channel Directories: Channels are any data stream. Currently time-series and video data are supported, but other channel types may be incorporated in the future.
- All channels are divided into segments. All channels are required to have at least one segment.
- Every level of the hierarchy may have records associated with that level.
- Each Session Directory contains:
    - Record Data File *(if present, a session Record Indices file must be present)*
    - Record Indices File *(if present, a session Record Data file must be present)*
    - Segmented Records Directory *(if present)* containing:
        - Record Data Files *(if present, corresponding Record Indices file must be present)*
        - Record Indices Files *(if present, corresponding Record Data files must be present)*
    - Time Series Channel Directories containing:
        - Record Data File *(if present, a channel Record Indices file must be present)*
        - Record Indices File *(if present, a channel Record Data file must be present)*
        - Segment directories containing:
            - Metadata File
            - Data File
            - Indices File
            - Record Data File *(if present, a segment Record Indices file must be present)*
            - Record Indices File *(if present, a segment Record Data file must be present)*
    - Video Channel directories containing:
        - Record Data File *(if present, a channel Record Indices file must be present)*
        - Record Indices File *(if present, a channel Record Data file must be present)*
        - Segment directories containing:
            - Metadata File
            - Indices File
            - Data Files (native video format file - e.g. MPEG; there can be multiple video files per segment - see naming convention below)
            - Record Data File *(if present, a segment Record Indices file must be present)*
            - Record Indices File *(if present, a segment Record Data file must be present)*

MED Naming Conventions (See Figure 1)

- Session Directories are named according to user preference and carry the ".medd" extension.
- Segmented Session Record Directories are named with the session name appended by ".recd". As with all record entities, this directory is optional.
- Record Data Files are named as the level (session, channel, segment) name appended by ".rdat".
- Record Indices Files are named as the level name appended by ".ridx".
- Record files within a Segmented Session Record directory are named with the session name, appended with an underscore and the letter "s", and a sequential fixed-width (4 digit) numbers starting from 1 (e.g. 0001, 0002, …) corresponding to the segment number with which they are associated (e.g. "Sess_X_s0001.rdat" & "Sess_X_s0001.ridx").
- Time Series Channel Directories are named as the channel name appended by ".ticd".
- Video Channel Directories are named according to user preference appended by ".vidd".
- Segments are named with the channel name, appended with an underscore and the letter "s", and a sequential fixed-width (4 digit) numbers starting from 1 (e.g. 0001, 0002, …). (e.g. "Chan_01_s0001").
- Time Series Segment Directories are named with the segment name, appended with the extension ".segd". (e.g. "Chan_01_s0001.tisd").
- Video Segment Directories are named with the segment name, appended with the extension ".segd". (e.g. "Chan_01_s0001.visd").
- Time Series Metadata Files are named as the segment name appended by ".tmet".
- Time Series Indices Files are named as the segment name appended by ".tidx".
- Time Series Data Files are named as the segment name appended by ".tdat". There is only one time series data file per segment (as opposed to Video Data Files).
- Video Metadata Files are named as the segment name appended by ".vmet".
- Video Indices Files are named with the video directory name appended by ".vidx".
- The Video Data Files are named with the segment name, appended with an underscore and the letter "n", and a sequential fixed-width (4 digit) video file numbers starting from 1 (e.g. "Vid_1_s0001_n0001"). They are appended by their native data format extension (e.g. "Vid_1_s0001_n0001.mpeg"). There can be multiple video data files per segment (as opposed to Time Series Data Files).

MED Data Type Definitions:

Type Name	Description
tern	1 byte signed integer where 1 = true, 0 = unknown / not set, -1 = false
ui1	1 byte unsigned integer
si1	1 byte signed integer
ui4	4 byte unsigned integer
si4	4 byte signed integer
sf4	4 byte signed floating point number
ui8	8 byte unsigned integer
si8	8 byte signed integer
sf8	8 byte signed floating point number
sf16	16 byte signed floating point number
utf8[n]	zero-terminated UTF-8 encoded string of maximum length “n” characters (not including terminal zero)
asc[n]	zero-terminated ascii encoded string of maximum length “n” characters (not including terminal zero)

MED Segments

- Segments are defined in the session scope, and represent the same expanse of time across all channels of all types.
- A channel is not required to contain all session segments, but those that it does contain must align with all other segments of the same number.
- The only ***logical*** distinctions between discontinuities and segment breaks in MED are:
    1. Discontinuities may be channel specific (practically they are often global, and incur much less overhead).
    2. Segments do not necessarily represent discontinuities (no data lost in transition).
- The ***practical*** distinction between discontinuities and segment breaks is that segment breaks close all files in segment directories & create new files & directories. Contrariwise, discontinuities simply are flagged in the data & indices files. Separating file system directories may be desirable to physically separate data regions in which something unique occurred, such as a discrete experiment.

MED Time Series Data Format

- Data are stored in compressed (CMP) blocks, compressed with any of the following algorithms:
  - Range Encoded Derivatives (RED): lossless or lossy, best for real-time and hardware implementations
  - Predictive Range Encoded Derivatives (PRED): lossless or lossy, best compression ratio for standard CPU-based implementations (default)
  - Minimal Bit Encoding (MBE): lossless, best for degenerate data
  - Vectorized Data Stream (VDS): lossy, highest compression ratio
- MED can encode signed integer data with 32-bit resolution, giving a full range of $-(2^{31})$ to $+(2^{31} - 1)$. [decimal -2,147,483,648 to +2,147,483,647] [hex 0x80000000 to 0x7FFFFFFF]
- -2^{31} is reserved to represent NaN (not a number). [decimal -2,147,483,648] [hex 0x80000000]
- $+(2^{31} - 1)$ is reserved to represent positive infinity. [decimal 2,147,483,647] [hex 0x7FFFFFFF]
- $-(2^{31} - 1)$ is reserved to represent negative infinity. [decimal -2,147,483,647] [hex 0x80000001]
- The unreserved range is therefore $-(2^{31} - 2)$ to $+(2^{31} - 2)$. [decimal -2,147,483,646 to +2,147,483,646] [hex 0x80000002 to 0x7FFFFFFE]
- Data blocks are indexed in the Time Series Indices file for random access.
- I am not a number, I am a nan.

MED Data Alignment

- All fields in all files in the format are aligned such that their values align to a multiple of their size from the beginning of the file. This allows for data read to be cast directly into data structures and for memory mapping of files.
- This alignment also facilitates recovery in the event of file damage.
- Pad bytes are added, if necessary, to maintain alignment, at the end of CMP Blocks, and Record Bodies. The value of the the pad byte is specified to be 0x7E, the ascii tilde (“~”). Specification of this value is done to facilitate reproducible CRCs and may be useful in the case of data recovery if file damage were to occur.

MED Strings

- All strings related to naming and descriptive data use UTF-8 encoding to allow for international character sets.
- UTF-8 encoding:
  - variable length characters
  - 1 to 4 bytes per character
  - not endian-sensitive
  - strings are null-terminated
  - ASCII is a valid subset of UTF-8, and thus any utf8[n] field is equivalent to an asc[n*4] field
- Unused bytes in MED string fields are set to zero to promote reproducibility of CRC values.
- Library string functions facilitate all of the above.

Micro-UTC Time (*μ*UTC)

- All times in MED are represented as offset *μ*UTC times.
- A *μ*UTC time is an si8 containing the elapsed microseconds since January 1, 1970 at 00:00:00 in the UTC (Coordinated Universal Time; aka GMT) time zone.
- *μ*UTC is simply converted to UTC (Coordinated Universal Time: seconds since 1/1/1970 at 00:00:00 GMT. Referred to as “The Epoch”, defined by the International Telecommunications Union) by dividing by 1,000,000.
- In MED all *μ*UTC times are stored and utilized as offset *μ*UTC times (oUTC) by subtracting a recording time offset. If the recording time offset is zero, the times are effectively not offset. If the recording time offset is known, when reading a file, it will be used when displaying times and dates.

Recording Time Offsets

- All times in the MED format are obfuscated with a value called the “Recording Time Offset” which is stored in Section 3 of the Metadata files. Data are stored with the recording time offset applied and represent times based in the UTC timezone such that the recording start day is January 1, 1970; the recording start time of day is the same as the true local recording start time excepting daylight saving. This mechanism allows preservation of time of day information, without providing any true date or timezone information. True time & date values, excluding daylight offset, are retrieved by adding the “Recording Time Offset” value (stored in section 3 of the Metadata files).

- No Daylight saving time correction is used in the offset mechanism; it is based on standard local time. To include DST corrections accurately would require knowledge of the true recording location and start time, which this mechanism is designed to obscure. Thus time of day is inaccurate by the local DST offset during DST in obfuscated times. DST is accurately accounted for in un-obfuscated times if DST change data is present (stored in section 3 of the Metadata files).
- The Recording Time Offset is included in Section 3 of the Metadata files, and if times are not offset, this field is set to zero.
- As recording time offsets and DST change information are stored in section 3 of the Metadata files, to report true local time, Metadata files should be read first when reading a segment.

Encryption

- Four tiered levels of encryption are defined, referred to as Level 0*, Level 1, Level 2, & Level 3 (Level 3 is for Level 1 & 2 password recovery only - it is not a valid encryption level itself within the MED file schema)
- *Level 0 encryption indicates no encryption.
- Level 1 and Level 2 encryption can be selected in various places in the MED file hierarchy:
    - Sections 2 and 3 of Metadata Files
    - Individual records of Record Data Files
    - Individual CMP blocks of the Time Series Data Files
- Level 2 decryption ability guarantees Level 1 decryption ability, but not the converse.
- Level 1 encryption is typically used for technical data, and Level 2 encryption for potentially subject identifying data. This way technical data can be shared with collaborators with out violating subject privacy. *However, encryption levels can be designated in any way desired by the file creator.*
- Level 2 encryption requires specification of a Level 1 password, even if Level 1 encryption is not employed anywhere in the file.
- Password hints can be specified for Level 1 & Level 2 passwords. These are stored in section 1 of the metadata files (which is not itself an encryptable region).

- An *optional* Level 3 password can be specified during file creation which will allow retrieval of the Level 1 and Level 2 passwords. Level 3 is not an encryption level in itself, however. The intention of the Level 3 password is to allow for a *broad failsafe* against password loss. Obviously, if used, Level 3 passwords should be carefully guarded. A typical usage might be: All EEG studies collected by an institution are encoded with the same Level 3 password (perhaps changed on a fixed schedule), known only to system administrators.
- The encryption / decryption algorithm is the 128-bit Advanced Encryption Standard (AES). [http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf], which satisfies the Health Insurance Portability and Accountability Act (HIPAA) 112-bit requirement for symmetric encryption of human data.

Video Data File Encryption

If video data file (mp4, etc.) encryption is desired, the first 4096 bytes of the file are encrypted rendering the file uninterpretable by standard video viewers. Encryption status is marked in the universal header.

Passwords

- AES-128 requires a 16 byte key. Therefore passwords are limited to 16-bytes.
- ASCII passwords are simply the ascii characters.
- Multibyte UTF-8 password characters are used internally in MED by taking the last (most unique) byte in each character of the UTF-8 encoding.
- The password length limit is 16 (UTF-8) characters.
- Programming Note: Because MED passwords are required to be null terminated strings, the string buffer length must accommodate a terminal zero (typically 17 bytes, but up to 65 bytes (= (16 * 4) + 1) for UTF-8 passwords).
- Password validation fields are created using hashes (SHA-256 algorithm) of the passwords. The passwords themselves are not stored in MED files.

Time Series Compression

- At the time of this writing, compression is done by one of four algorithms:
    1. RED (Range Encoded Derivatives) differentiates the data, and then range encodes the derivatives. RED is lossless by default, but can be used as lossy for higher compression.
    2. PRED (Predictive Range Encoded Derivatives) uses 3 separate models to predictively encode the data using the RED algorithm. This algorithm is more computationally expensive on encoding, but produces higher compression ratios. PRED is lossless by default, but can be used as lossy for higher compression.
    3. MBE (Minimal Bit Encoding) simply encodes each raw sample with the minimum bits required for the range of the block. This is typically used when a RED or PRED encoded block would exceed the compression ratio of MBE. This is useful for blocks that contain highly uncorrelated (noisy) data.
    4. VDS (Vectorized Data Stream) is a lossy algorithm that stores the data as a series of anchor points with non-uniform spacing. VDS can produce very high compression ratios. VDS is computationally expensive on encoding.
- Data can optionally be detrended prior to applying compression. This operation is lossless, but is generally more useful in lossy compression routines.

- Lossy compression is permitted in time series with the RED or PRED algorithms data by scaling data prior to compression. Scaling is adaptive and may vary from block to block. The scaled values must be rounded to the nearest integer, introducing the loss. Lossy compression is not required, but can produce substantial storage savings with negligible data derivatives in data streams whose sample-value specificities exceed their information content. Compression can also be useful in speeding transmission and viewing of data. Lossy compression with RED or PRED is less computationally intensive than VDS, but is generally inferior in both quality and compression ratios.
- Four RED/PRED compression modes are currently supported:
    1. Lossless (default)
    2. Fixed Scale Factor: a user-specified scale factor is applied to the block (1.0 results in lossless compression)
    3. Fixed Compression Ratio: the scale factor is adjusted for each block until the compression ratio (block_bytes / input_array_size [as si4s]) is this number plus or minus a tolerance. e.g. 20% of the original si4 size with a 1% tolerance is 0.19 to 0.21. If lossless compression can achieve or exceed the desired ratio (plus the tolerance), lossless compression will be applied. This option may add noticeable processing time to compression, but once done, adds negligible time to decompression. This mode prioritizes size over fidelity.
    4. Mean Residual Ratio: the scale factor is adjusted for each block until the mean(abs((scaled_data$_i$ - original_data$_i$) / original_data$_i$)) for the values in the block, is this number plus or minus a tolerance. e.g. 0.5% difference with a 0.1% tolerance is 0.004-0.006. This option may add noticeable processing time to compression, but once done, adds negligible time to decompression. This mode prioritizes fidelity over size.
    5. Fixed Compression Ratio and Mean Residual Ratio find the scale factors algorithmically for each block individually, and so adapt to the local data.
    6. Require Normality: if this is set, lossy compression will only be performed on data blocks that pass a modified Kolmogorov-Smirnov test for normality at a user specified level.
    7. Use Relative Ratio: if this is set, the goal ratio for each block is divided by it's coefficient of variation, providing higher fidelity in blocks with higher variance.

- Two lossy compression dimensions are available:
    1. Amplitude
    2. Frequency
    3. These can be use in combination, further reducing data size:
        a. If used together, amplitude compression precedes frequency compression

b. If used together, the mode & tolerance are assumed to be the same for both (this could be changed with trivial custom programming)

Error Detection & Correction

- Checksums (32-bit CRCs) are judiciously distributed throughout the MED file structures to detect data corruption:
  1. Universal Header of every MED file:
     a. Universal Header CRC (for universal header itself)
     b. File Body CRC (for rest of file excluding the universal header)
  2. Record CRC: for each record in a record data file
  3. Time Series Data Block CRC: for each data block in a time series data file
  4. Video data files: as these file use native external video formats, there is no CRC for these files unless their format includes one.
- Parity Channels:
  1. Store bitwise parity information for a fixed set of MED data.
  2. Parity can be used to repair any file in a MED session, if all data in the set are available and unmodified since the parity data was created.
  3. Repair can be limited to just the region of CRC checksum mismatch in the damaged channel.
  4. Parity data files are the same size as the longest file used in their creation.
  5. Channel sets can include all channels or any subset of channels. Any channel type can be included in a channel set (times series, video channels, or any future channel type). However, it may be desirable to limit parity channels to just time series channels to reduce their size.
  6. Parity channel data must be rebuilt if channels are added, deleted, or modified. ***This is the only inter-channel dependency present in the MED format.***

Protected and Discretionary File Regions

- A protected region is reserved for possible future additions to the MED format and should not be modified by end users. Any “reserved” regions are also protected.
- A discretionary region is reserved for end user use so that custom data can be conveniently added to the files without interfering with the specified format fields.
- Protected and discretionary regions can be found in the universal header, each section of the metadata files, and optionally in CMP block headers.

MED Parity Data

MED offers the option of generating parity data for every file type. Storing parity data allows for targeted micro-repair of damaged MED files utilizing the CRCs inherent in MED file structures. Parity data adds to the size of a MED session by approximately the size of one extra channel. Parity is the single instance of channel interdependence in MED; i.e. if a MED file is modified, its corresponding parity file must be rebuilt.

The header, body, record & block CRCs in MED allow for localization of damage, and allow repair to be restricted to only the damaged region.

Session: test.medd

- 3 time series channels:
  - 5k_0001
  - 16k_0001
  - 40k_0001
- 2 segments
- Session-level records
- Segmented session-level records
- Channel-level records

Parity data is stored in parity files constructed from all the MED files in the same level of the MED hierarchy. Parity data has it's own independent blocked CRCs so that it's integrity can be verified prior to using it rebuild damaged data. This data is stored at the end of the file in a PCRC structure. PCRCs can be added to any file type. They are required for video data files which do not have their own internal CRCs, but can be added to other MED file types if desired. One situation in which this may be desirable is to increase the resolution of damage detection in very long index files (time series or video) which have only a universal header & body CRC. This is not the default behavior however, because damaged index files can be entirely rebuilt from their corresponding data files.

Functions for detecting damaged files, generating parity data, localizing damage within damaged files, and repairing that damage from parity, are all included into MED C library.

Video Data File CRCs:
If CRCs are desired for video data files (mp4, etc.), the CRCs are added to the end of the existing file (before the Universal Header, discussed below). The files remain readable by standard video viewers.

Example MED Session with Parity

```
test.medd
├── 5k_0001.ticd
│   ├── 5k_0001.rdat
│   ├── 5k_0001.ridx
│   ├── 5k_0001_s0001.tisd
│   │   ├── 5k_0001_s0001.tdat
│   │   ├── 5k_0001_s0001.tidx
│   │   └── 5k_0001_s0001.tmet
│   └── 5k_0001_s0002.tisd
│       ├── 5k_0001_s0002.tdat
│       ├── 5k_0001_s0002.tidx
│       └── 5k_0001_s0002.tmet
├── 16k_0001.ticd
│   ├── 16k_0001.rdat
│   ├── 16k_0001.rdat.bak
│   ├── 16k_0001.ridx
│   ├── 16k_0001_s0001.tisd
│   │   ├── 16k_0001_s0001.tdat
│   │   ├── 16k_0001_s0001.tidx
│   │   └── 16k_0001_s0001.tmet
│   └── 16k_0001_s0002.tisd
│       ├── 16k_0001_s0002.tdat
│       ├── 16k_0001_s0002.tidx
│       └── 16k_0001_s0002.tmet
├── 40k_0001.ticd
│   ├── 40k_0001.rdat
│   ├── 40k_0001.ridx
│   ├── 40k_0001_s0001.tisd
│   │   ├── 40k_0001_s0001.tdat
│   │   ├── 40k_0001_s0001.tidx
│   │   └── 40k_0001_s0001.tmet
│   └── 40k_0001_s0002.tisd
│       ├── 40k_0001_s0002.tdat
│       ├── 40k_0001_s0002.tidx
│       └── 40k_0001_s0002.tmet
├── parity.ticd (parity channel directory)
│   ├── parity.rdat (channel record data parity: across channels)
│   ├── parity.ridx (channel record indices parity: across channels)
│   ├── parity_s0001.tisd (parity channel segment 1 directory: across channels)
│   │   ├── parity_s0001.tdat (segment 1 time series data parity: across channels)
│   │   ├── parity_s0001.tidx (segment 1 time series indices parity: across channels)
│   │   └── parity_s0001.tmet (segment 1 time series metadata parity: across channels)
│   └── parity_s0002.tisd (parity channel segment 2 directory: across channels)
│       ├── parity_s0002.tdat (segment 2 time series data parity: across channels)
│       ├── parity_s0002.tidx (segment 2 time series indices parity: across channels)
│       └── parity_s0002.tmet (segment 2 time series metadata parity: across channels)
├── test.rdat
├── parity.rdat (session record data parity: single file)
├── test.ridx
├── parity.ridx (session record indices parity: single file)
└── test.recd
    ├── parity_s0000.rdat (segmented session record data parity: across segments)
    ├── parity_s0000.ridx (segmented session record indices parity: across segments)
    ├── test_s0001.rdat
    ├── test_s0001.ridx
    ├── test_s0002.rdat
    └── test_s0002.ridx
```

Encryption Level Schema

- The following table contains codes for encryption that are useful in processing as well as in file encoding.

Encryption Level Schema:

Value	Meaning
0	No encryption
1	Level 1 encrypted
-1	Level 1 encryption specified, currently decrypted
2	Level 2 encrypted
-2	Level 2 encryption specified, currently decrypted
-128	No entry

Universal Header

- Each file in the MED structure begins with a universal header
- The only current exception is video data files whose content is determined entirely by their specific video format (e.g. MPEG).
- The universal header is not encrypted.
- Design concepts:

    a. Contains the minimum information required to read a file in the absence of any other files (e.g. indices or metadata). Appropriate interpretation of the data may still require metadata and passwords.

    b. Contains the minimum information to uniquely identify a file, its place in a MED hierarchy, and its provenance.

    c. Contains the minimum information required to detect file corruption.

    d. Facilitates decryption of potentially encrypted information.

    e. Fields whose values may change with each file write operation are clustered at the beginning of the universal header in the "Robust Mode Region", so termed because they may be updated with every write (see "Robust Mode" section below).

Video Data File Universal Headers:
The universal headers for video data file (mp4, etc.) are added to the end of the file (becoming footers, technically). The files remain readable by standard video viewers.

Robust Mode

- Advantage: MED file creation is robust to catastrophic failure during recording (e.g. power or network failure).
- Disadvantages:
    a. Disk thrashing causes increased wear & tear on hard drives.
    b. Increased write frequency on any medium (yet invented) reduces storage lifespan (particularly SSDs).
    c. Speed requirements of Robust Mode may exceed the maximum write speed of the storage medium.
- In Robust Mode the following are updated with every write (in addition to added data):
    a. File Universal Header (Robust Region fields)
    b. MED Indices file terminal indices
    c. MED Metadata files, section 2 (technical metadata)
- Catastrophic failure during recording without Robust Mode (standard recording) will **not** result in data loss. Update of the following is required to satisfy all MED format specifications, however:
        a. All MED file universal headers (Robust Mode fields)
        b. Indices file terminal indices
        c. Metadata file technical fields

    This information can all be retrieved from the recorded MED files before failure. It just requires an extra software step for correction. Thus choose robust mode by weighing the following factors:
        a. Obstacles to post acquisition data repair (e.g. data difficult to modify where recorded, personnel will not run repair software)
        b. Acquisition failure likelihood.
        c. Speed & lifespan of storage medium.
- For most settings, Robust Mode should not be the default.
- *Note: at the time of this writing, code for repairing interrupted MED recordings is not included in the open source library, but will be in the future.*

Universal Header:

Field	Offset	Bytes	Type	Contents
Robust Mode Region Start				
Header CRC	0	4	ui4	• CRC of the universal header after this field • 0 indicates no entry
Body CRC	4	4	ui4	• CRC of the entire file after the universal header • 0 indicates no entry
File End Time	8	8	si8	• File end time in offset μUTC format • If segment file, this is segment end time • 0x8000000000000000 indicates no entry • *In the ephemeral SESSION, CHANNEL, & SEGMENT library structures, this is the latest end time of all its contents*
Number of Entries	16	8	si8	• Number of entries in the file • See **Universal Header Number of Entries** table (below) for the specific meaning for each file type • -1 indicates no entry
Maximum Entry Size	24	4	ui4	• Maximum size of an entry in the file • See **Universal Header Number of Entries** table (below) for the specific meaning for each file type • 0 indicates no entry
Robust Mode Region End				
Segment Number	28	4	si4	• Number of the segment (if applicable) • Numbering starts at 1 • -1 indicates no entry • -2 indicates channel level • -3 indicates session level

Field	Offset	Bytes	Type	Contents
Type String or Type Code	32	5	asc[4] or ui4	• 4 ascii characters of file name extension, null terminated or used as ui4 value • 0 (all zeros = zero-length string) indicates no entry • In the ephemeral SESSION & CHANNEL library structures, this is the directory type
MED Version Major	37	1	ui1	• numeric value: 1, currently • 0xFF indicates no entry
MED Version Minor	38	1	ui1	• numeric value: 1, currently • 0xFF indicates no entry
Byte Order Code	39	1	ui1	• 0 ==> big-endian • 1 ==> little-endian • 0xFF indicates no entry • *Only little-endian byte order is supported by the library at this time*
Session Start Time	40	8	si8	• Session start time in offset μUTC format • 0x8000000000000000 indicates no entry
File Start Time	48	8	si8	• File start time in offset μUTC format • If segment file, this is segment start time • 0x8000000000000000 indicates no entry • *In the ephemeral SESSION, CHANNEL, & SEGMENT library structures, this is the earliest start time of all its contents*
Session Name	56	256	utf8[63]	• Session name without path or extension • Zero-length string indicates no entry

Field	Offset	Bytes	Type	Contents
Channel Name	312	256	utf8[63]	• Channel name without path or extension • Zero-length string indicates no entry
Supplemental Protected Region	568	256		• Filled with zeros • *Previously "Anonymized Subject ID" field, moved to Metadata, Section 1*
Session UID	824	8	ui8	• Unique Identifying Number • 8 random bytes shared by all files in the session • **Session UID** of originating data set • Zeros indicate no entry
Channel UID	832	8	ui8	• Unique Identifying Number • 8 random bytes shared by all files in the channel • **Channel UID** of originating data set • Zeros indicate no entry
Segment UID	840	8	ui8	• Unique Identifying Number • 8 random bytes shared by all files in the segment • **Segment UID** of originating data set • Zeros indicate no entry
File UID	848	8	ui8	• Unique Identifying Number • 8 random bytes unique to the current file • **Unique to the current file**, regardless of whether the data set is derivative • Zeros indicate no entry

Field	Offset	Bytes	Type	Contents
Provenance UID	856	8	ui8	• Unique Identifying Number • **File UID** of originating file • Identity with the **File UID** indicates that this is the originating file in a data set • Zeros indicate no entry
Level 1 Password Validation Field	864	16	ui1[16]	• First 16 binary bytes of a SHA-256 hash of the Level 1 password • Zeros indicate no entry
Level 2 Password Validation Field	880	16	ui1[16]	• Exclusive-or of first 16 bytes of a SHA-256 hash of the Level 2 password with the unhashed Level 1 password • Zeros indicate no entry
Level 3 Password Validation Field	896	16	ui1[16]	• Intended as ***optional*** password recovery mechanism • Allows extraction of Level 1 & Level 2 passwords, if specified • Level 3 is **not** a valid encryption level itself • Exclusive-or of first 16 bytes of a SHA-256 hash of the Level 3 password with the unhashed Level 1 or 2 password (if specified) • Zeros indicate no entry
Video Data File Number	912	4	ui4	• Segment video data file number • Added in MED 1.1
Data Encryption	916	1	si1	• Used with time series & video data • See Encryption Level Schema table • Added in MED 1.1
Metadata Section 2 Encryption	917	1	si1	• Used with metadata files • See Encryption Level Schema table • Added in MED 1.1

Field	Offset	Bytes	Type	Contents
Metadata Section 3 Encryption	918	1	si1	• Used with metadata files • See Encryption Level Schema table • Added in MED 1.1
Ordered	919	1	tern	• Used with data & indices files • Indicates the file entries are in order of start time • Added in MED 1.1
Protected Region	920	56		• Filled with zeros • Reserved for potential future use
Discretionary Region	976	48		• Filled with zeros if unused • Discretionary end-user use

Universal Header: Number of Entries

File Type	Extension(s)	*Number of Entries* Contents	*Maximum Entry Size* Contents
Record Data File	rdat	• Number of records in the file • -1 indicates no entry	• Number of **bytes** (including record header and pad bytes) in the largest record in the file • -1 indicates no entry
Record Indices File	ridx	• Number of records indices in the file (= number of records) • -1 indicates no entry	• Number of **bytes** in a record index (a constant) • -1 indicates no entry
Metadata Files	tmet vmet	1	• Number of **bytes** in a metadata file (a constant) • -1 indicates no entry
Time Series Data File	tdat	• Number of CMP blocks in the file • -1 indicates no entry	• Number of **bytes** in the largest CMP block in the file • -1 indicates no entry
Time Series Indices File	tidx	• Number of time series indices in the file, including (extra) terminal index • -1 indicates no entry	• Number of **bytes** in a time series index (a constant) • -1 indicates no entry
Video Indices File	vidx	• Number of video indices in the file(s), including (extra) terminal index • -1 indicates no entry	• Number of **bytes** in a video index (a constant) • -1 indicates no entry
Ephemeral SESSION Metadata FPS		• Maximum number of Records/Record Indices in the Channel directories and Session level records • -1 indicates no entry • Note that the SESSION Universal Header structure is ephemeral (never written to disk)	• Maximum number of **bytes** in a Record in the Channel directories and Session level records • -1 indicates no entry • Note that the SESSION Universal Header structure is ephemeral (never written to disk)

File Type	Extension(s)	*Number of Entries* Contents	*Maximum Entry Size* Contents
Ephemeral CHANNEL Metadata FPS		• Maximum number of Records/Record Indices in the Segment directories and Channel level records • -1 indicates no entry • Note that the CHANNEL Universal Header structure is ephemeral (never written to disk)	• Maximum number of **bytes** in a Record in the Segment directories and Channel level records • -1 indicates no entry • Note that the CHANNEL Universal Header structure is ephemeral (never written to disk)

Metadata Files

- One for each channel segment in the MED hierarchy
- The metadata files share an identical format, but most section 2 fields are specific to the channel data type.
- Currently there are 2 types of metadata files specified: time-series and video. The first four fields of section 2 are common to all section 2 types: *Session Description, Channel Description, Segment Description, and Equipment Description.*
- Each type of metadata file has its own file type, which also serves as its file name extension.
- Ephemeral metadata files are not part of the stored MED file hierarchy, but are optionally created while reading data. They contain summary metadata for the levels below them: an ephemeral channel metadata file is created to summarize the data in a selected set of segments it contains. Likewise an ephemeral session metadata file may be created to summarize the data in a selected set of channels it contains. In the case of ephemeral *session* metadata files, one is created for each channel type in the session (e.g. time series, video)
- In Robust Mode, Section 2 of the metadata should be updated with every write, in addition to the universal headers.

Metadata Files:

Field	Offset	Bytes	Type	Contents	Encryption
Universal Header	0	1024		See “Universal Header” description	None
Section 1					
Level 1 Password Hint	1024	256	utf8[63]	• Zero-length string indicates no entry	None
Level 2 Password Hint	1280	256	utf8[63]	• Zero-length string indicates no entry	None
Anonymized Subject ID	1536	256	utf8[63]	• Anonymized subject ID • Anonymized name or number is typical • Zero-length string indicates no entry	None
Protected Region	1792	128		• Filled with zeros • Reserved for potential future use	None
Discretionary Region	1920	128		• Filled with zeros if unused • Discretionary end-user use	None
Section 2 (technical data)					
Metadata Section 2 Channel Type Specific Fields	2048	10240		See channel type specific tables below	As specified in Section 1
Section 3 (subject specific data)					
Recording Time Offset	12288	8	si8	• Value to add to all μUTC times to adjust them to true UTC time • Zero indicates no entry	As specified in Section 1

Field	Offset	Bytes	Type	Contents	Encryption
Daylight Time Start Code	12296	8	*Daylight Time Change Code* (si1[8] / si8)	• See Daylight Time Change Code Table below • Zero in regions that do not observe DST • (si8) -1 indicates no entry • *Note that this code reflects the regional rules at the time of the recording only*	As specified in Section 1
Daylight Time End Code	12304	8	*Daylight Time Change Code* (si1[8] / si8)	• See Daylight Time Change Code Table below • Zero in regions that do not observe DST • (si8) -1 indicates no entry • *Note that this code reflects the regional rules at the time of the recording only*	As specified in Section 1
Standard Timezone Acronym	12312	8	asc[7]	• Daylight Saving or Summer Time is not included in this acronym • e.g “MST” for United States Mountain Standard Time • Zero-length string indicates no entry	As specified in Section 1
Standard Timezone String	12320	64	asc[63]	• Daylight Saving or Summer Time is not included in this string • e.g “Mountain Standard Time” for United States Mountain Standard Time • Zero-length string indicates no entry	As specified in Section 1

Field	Offset	Bytes	Type	Contents	Encryption
Daylight Timezone Acronym	12384	8	asc[7]	• Daylight Saving or Summer Time version of the Standard Timezone Acronym • e.g “MDT” for United States Mountain Daylight Time • Zero-length string indicates no entry for regions that do not observe DST	As specified in Section 1
Daylight Timezone String	12392	64	asc[63]	• Daylight Saving or Summer Time version of the Standard Timezone String • e.g “Mountain Daylight Time” for United States Mountain Daylight Time • Zero-length string indicates no entry for regions that do not observe DST	As specified in Section 1
Subject Name 1	12456	128	utf8[31]	• Typically subject first name • Zero-length string indicates no entry	As specified in Section 1
Subject Name 2	12584	128	utf8[31]	• Typically subject middle name • Zero-length string indicates no entry	As specified in Section 1
Subject Name 3	12712	128	utf8[31]	• Typically subject last name • Zero-length string indicates no entry	As specified in Section 1
Subject ID	12840	128	utf8[31]	• Subject ID • Zero-length string indicates no entry	As specified in Section 1

Field	Offset	Bytes	Type	Contents	Encryption
Recording Country	12968	256	utf8[63]	• Country in which the recording occurred • Zero-length string indicates no entry	As specified in Section 1
Recording Territory	13224	256	utf8[63]	• Territory, Province, State, etc. in which the recording occurred • Zero-length string indicates no entry	As specified in Section 1
Recording Locality	13480	256	utf8[63]	• City, Township, Village, etc. in which the recording occurred • Zero-length string indicates no entry	As specified in Section 1
Recording Institution	13736	256	utf8[63]	• Organization, Institution, etc. in which the recording occurred, or other description of where recording occurred • Zero-length string indicates no entry	As specified in Section 1
GeoTag Format	13992	32	asc[31]	• GeoTag data format, e.g. “Exif”, “XMP”, “GeoSMS” • Zero-length string indicates no entry	As specified in Section 1
GeoTag Data	14024	1024	asc[1023]	• GeoTag data • Zero-length string indicates no entry	As specified in Section 1

Field	Offset	Bytes	Type	Contents	Encryption
Standard UTC Offset	15048	4	si4	• File recording time zone *expressed in seconds* ahead or behind UTC (GMT), in **Standard Time** • Daylight Saving or Summer Time is not included in this field • Added to μUTCs to get local time of day. • e.g: 0 indicates GMT (Greenwich Mean Time); -18000 (-5 * 60 * 60) indicates EST (Eastern Standard Time) • 0x7FFFFFFF indicates no entry	As specified in Section 1
Protected Region	15052	668		• Filled with zeros • Reserved for potential future use	As specified in Section 1
Discretionary Region	15720	664		• Filled with zeros if unused • Discretionary end-user use	As specified in Section 1

Daylight Time Change Code Table:

si1 Union Values		
Byte	Field	Values
0	Code Type	(DST end / DST Not Observed / DST start) == (-1 / 0 / +1)
1	Day of Week	(No Entry / [Sunday : Saturday]) == (-1 / [0 : 6]) *Unix time functions encode the days of the week in the range [0 : 6]*
2	Relative Weekday of Month	(No Entry / [First : Fifth] / Last) == (0 / [1 : 5] / 6)
3	Day of Month	(No Entry / [1 : 31]) == (0 / [1 : 31]) *Unix time functions encode the days of months in the range [1 : 31]*
4	Month	(No Entry / [January : December]) == (-1 / [0 : 11]) *Unix time functions encode months in the range [0 : 11]*
5	Hours of Day	[-128 : +127] hours relative to 0:00 (midnight)
6	Reference Time	(Local / UTC) == (0 / +1) *Any entry can be encoded in either, but local is usually more intuitive*
7	Shift Minutes	[-120 : +120] minutes *Typically +60 for DST start & -60 for DST end*
si8 Union Values		
0 indicates DST is not observed		
-1 indicates no entry		

Time Series Metadata Section 2:

Field	Offset	Bytes	Type	Contents	Encryption
Section 2 (technical data): Channel Type Independent Fields					
Session Description	2048	2048	utf8[511]	• Description of recording session • Zero-length string indicates no entry • Present in all section 2 metadata types	As specified in Section 1
Channel Description	4096	1024	utf8[255]	• Description of recording channel • Zero-length string indicates no entry • Present in all section 2 metadata types	As specified in Section 1
Segment Description	5120	1024	utf8[255]	• Description of recording segment • Zero-length string indicates no entry • Present in all section 2 metadata types	As specified in Section 1
Equipment Description	6144	2044	utf8[510]	• Description of recording equipment • Zero-length string indicates no entry • Present in all section 2 metadata types	As specified in Section 1
Acquisition Channel Number	8188	4	si4	• Number of the time series channel in the original recording • -1 indicates no entry • *Library default numbering is from 1, but zero-based or other numbering schemes may be used*	As specified in Section 1
Section 2 (technical data): Channel Type Specific Fields					

Field	Offset	Bytes	Type	Contents	Encryption
Reference Description	8192	1024	utf8[255]	• Description of recording reference channel • Zero-length string indicates no entry	As specified in Section 1
Sampling Frequency	9216	8	sf8	• Sampling frequency • This is the acquisition sampling frequency: individual blocks may be subsampled from this without affecting this value • -1.0 indicates no entry • -2.0 indicates variable sampling frequency during acquisition	As specified in Section 1
Low Frequency Filter Setting	9224	8	sf8	• High-pass filter setting, in Hertz • -1.0 indicates no entry	As specified in Section 1
High Frequency Filter Setting	9232	8	sf8	• Low-pass filter setting, in Hertz • -1.0 indicates no entry	As specified in Section 1
Notch Filter Frequency Setting	9240	8	sf8	• Notch filter setting, in Hertz • -1.0 indicates no entry	As specified in Section 1
AC Line Frequency	9248	8	sf8	• AC line frequency, in Hertz • -1.0 indicates no entry	As specified in Section 1
Amplitude Units Conversion Factor	9256	8	sf8	• Value to multiply sample values by to get native units (“Units Description” field) • 0.0 indicates no entry • Negative values indicate values are inverted	As specified in Section 1

Field	Offset	Bytes	Type	Contents	Encryption
Amplitude Units Description	9264	128	utf8[31]	• String describing units (e.g. “microvolts”) • Zero-length string indicates no entry	As specified in Section 1
Time Base Units Conversion Factor	9392	8	sf8	• Value to multiply time values by to get μUTC time • 0.0 indicates no entry • Allows format to accommodate time bases coarser or finer than microseconds	As specified in Section 1
Time Base Units Description	9400	128	utf8[31]	• String describing time base units (e.g. “μUTC”) • Zero-length string indicates no entry	As specified in Section 1
Absolute Start Sample Number	9528	8	si8	• Number of the first sample in the CMP block data relative to all samples in the channel (*not the segment*) • The number of the first sample number in *first* segment is zero • 0x8000000000000000 indicates no entry	As specified in Section 1
Number of Samples	9536	8	si8	• Total recorded samples in the segment • -1 indicates no entry	As specified in Section 1
Number of Blocks	9544	8	si8	• Total recorded CMP blocks in the file • -1 indicates no entry	As specified in Section 1

Field	Offset	Bytes	Type	Contents	Encryption
Maximum Block Bytes	9552	8	si8	• Maximum bytes, including header & pad bytes, in any CMP block in the file • -1 indicates no entry	As specified in Section 1
Maximum Block Samples	9560	4	ui4	• Maximum number of samples in a CMP block • 0xFFFFFFFF indicates no entry • Duplicated (as an si8) in Universal Header of Time Series Data Files	As specified in Section 1
Maximum Block Keysample Bytes	9564	4	ui4	• Maximum bytes required for the keysample data in RED/ PRED compressed blocks • 0xFFFFFFFF indicates no entry	As specified in Section 1
Maximum Block Duration	9568	8	sf8	• Duration of CMP blocks (*intended*) • Units described in Time Base Units Description (default units are microseconds) • -1.0 indicates no entry	As specified in Section 1
Number of Discontinuities	9576	8	si8	• Number of discontinuities in the segment • The first sample in a session, but not necessarily a segment, is considered a discontinuity • -1 indicates no entry	As specified in Section 1
Maximum Contiguous Blocks	9584	8	si8	• Maximum number of contiguous CMP blocks between discontinuities in the segment • -1 indicates no entry	As specified in Section 1

Field	Offset	Bytes	Type	Contents	Encryption
Maximum Contiguous Block Bytes	9592	8	si8	• Maximum number of contiguous compressed bytes between discontinuities in the segment (including block headers and pad bytes) • -1 indicates no entry	As specified in Section 1
Maximum Contiguous Samples	9600	8	si8	• Maximum number of contiguous samples between discontinuities • -1 indicates no entry	As specified in Section 1
Protected Region	9608	1344		• Filled with zeros • Reserved for potential future use	As specified in Section 1
Discretionary Region	10952	1336		• Filled with zeros if unused • Discretionary end-user use	As specified in Section 1

Video Metadata Section 2

Field	Offset	Bytes	Type	Contents	Encryption
Section 2 (technical data): Channel Type Independent Fields					
Session Description	2048	2048	utf8[511]	• Description of recording session • Zero-length string indicates no entry • Present in all section 2 types	As specified in Section 1
Channel Description	4096	1024	utf8[255]	• Description of the video stream • Zero-length string indicates no entry • Present in all section 2 types	As specified in Section 1
Segment Description	5120	1024	utf8[255]	• Description of the segment of the video stream • Zero-length string indicates no entry • Present in all section 2 types	As specified in Section 1
Equipment Description	6144	2044	utf8[510]	• Description of recording equipment • Zero-length string indicates no entry • Present in all section 2 metadata types	As specified in Section 1
Acquisition Channel Number	8188	4	si4	• Number of the video channel in the original recording • -1 indicates no entry • *Library default numbering is from 1, but zero-based or other numbering schemes may be used*	As specified in Section 1
Section 2 (technical data): Channel Type Specific Fields					

Field	Offset	Bytes	Type	Contents	Encryption
Time Base Units Conversion Factor	8192	8	sf8	• Value to multiply time values by to get μUTC time • 0.0 indicates no entry • Allows format to accommodate time bases coarser or finer than microseconds	As specified in Section 1
Time Base Units Description	8200	128	utf8[31]	• String describing time base units (e.g. "μUTC") • Zero-length string indicates no entry	As specified in Section 1
Absolute Start Frame Number	8328	8	si8	• Number of the first sample in the CMP block data relative to all samples in the channel (*not the segment*) • The number of the first sample number in *first* segment is zero • 0x8000000000000000 indicates no entry	As specified in Section 1
Number of Frames	8336	8	si8	• Total recorded frames in the segment • -1 indicates no entry	As specified in Section 1
Frame Rate	8344	8	sf8	• Frames per second • -1.0 indicates no entry • -2.0 indicates variable frame rate	As specified in Section 1
Number of Clips	8352	8	si8	• Total recorded clips in the segment • A clip is the video data between video indices • -1 indicates no entry	As specified in Section 1
Maximum Clip Bytes	8360	8	si8	• Maximum bytes, in video data in a single clip • -1 indicates no entry	As specified in Section 1

Field	Offset	Bytes	Type	Contents	Encryption
Maximum Clip Frames	8368	4	ui4	• Maximum number of frames in a clip • 0xFFFFFFFF indicates no entry	As specified in Section 1
Number of Video Files	8372	4	si4	• Number of video files in the segment • -1 indicates no entry	As specified in Section 1
Maximum Clip Duration	8376	8	sf8	• Duration of clips (*intended*) • Units described in Time Base Units Description (default units are microseconds) • -1.0 indicates no entry	As specified in Section 1
Number of Discontinuities	8384	8	si8	• Number of discontinuities in the segment • The first frame in a session, but not necessarily a segment, is considered a discontinuity • -1 indicates no entry	As specified in Section 1
Maximum Contiguous Clips	8392	8	si8	• Maximum number of contiguous clips between discontinuities in the segment • -1 indicates no entry	As specified in Section 1
Maximum Contiguous Clip Bytes	8400	8	si8	• Maximum number of contiguous compressed bytes between discontinuities in the segment • -1 indicates no entry	As specified in Section 1
Maximum Contiguous Frames	8408	8	si8	• Maximum number of contiguous frames between discontinuities • -1 indicates no entry	As specified in Section 1
Horizontal Pixels	8416	4	ui4	• Horizontal pixels • 0 indicates no entry	As specified in Section 1

Field	Offset	Bytes	Type	Contents	Encryption
Vertical Pixels	8420	4	ui4	• Vertical pixels • 0 indicates no entry	As specified in Section 1
Video Format	8424	256	utf8[63]	• e.g. “MPEG-4” • Zero-length string indicates no entry	As specified in Section 1
Protected Region	8680	1808		• Filled with zeros • Reserved for potential future use	As specified in Section 1
Discretionary Region	10488	1800		• Filled with zeros if unused • Discretionary end-user use	As specified in Section 1

Note: *semantic congruencies between time series & video channels: “sample” ~= “frame” ; “block” ~= “clip”*

Records Data File

- Binary format described below
- Can be present at any level of the MED hierarchy, but is never required.
- Session Records can be segmented, and if they exist, are stored in the “.recd” directory at the Session level of the file hierarchy. Segmented Session Records do no preclude the existence of unsegmented session records: either, both, or neither can exist.
- If a Records Data File is present, a Records Index File must also be present, and vice versa.
- Each record begins with a record header
- Example record types include:
    - Electrode & probe descriptions
    - Electrode coordinates
    - Electrode diagrams
    - Spike records
    - Seizure marks
    - Event related study data
    - Sleep stage / behavioral state

    - Miscellaneous notes
    - Acquisition system log entries
    - Acquisition system configuration
    - End-user defined record types
- Records can also be compressed, but the specific compression algorithm (e.g. jpeg, png, bzip) should be defined in the record description documentation.
- The length of the body of each record must be padded to a multiple of 16 for encryption. The pad-byte value is 0xFE (ascii tilde, “~”).

Records Data File:

Field	Offset	Bytes	Contents
Universal Header	0	1024	*See “Universal Header” description*
Records	1024		*See “Record Header Format” description*
…			

Record Header Format:

Field	Offset	Bytes	Type	Contents	Encryption
Record CRC	0	4	ui4	• Cyclically Redundant Checksum for record and remainder of Record Header • 0 indicates no entry	None
Total Bytes	4	4	ui4	• Record size in bytes, including record header and pad bytes if any. • 0 indicates no entry	None
Start Time	8	8	si8	• Record start time in μUTC time format. If needed, the record body should contain an end time. • If recording time offset is used for the session it is applied here also. • 0x8000000000000000 indicates no entry	None
Type String or Type Code	16	5	asc[4] or ui4	• 4 byte integer, typically representing 4 ascii characters, designating record type, null terminated, or used as ui4 value • 0 (all zeros = zero-length string) indicates no entry	None
Record Version Major	21	1	ui1	• Record type's major version • 0xFF indicates no entry	None
Record Version Minor	22	1	ui1	• Record type's minor version • 0xFF indicates no entry	None
Encryption Level	23	1	si1	• Changes sign when record is encrypted / decrypted • See "Encryption Level Schema" table	None

Record Indices File Format

- Universal header
- Sequential record index data
- 8-byte boundary aligned

Record Indices File:

Field	Offset	Bytes	Contents
Universal Header	0	1024	*See “Universal Header” description*
Record Index	1024	24	*See “Record Index Format” description*
…			

Record Index Format:

Field	Offset	Bytes	Type	Contents
File Offset	0	8	si8	• Record start file offset in bytes. • -1 indicates no entry • There is one terminal record index with **File Offset** equal to the record data file length
Start Time	8	8	si8	• Record time in μUTC time format. • If recording time offset is used for the session it is applied here also. • 0x8000000000000000 indicates no entry • There is one terminal record index with **Time** equal to the *segment* end μUTC + 1
Type String or Type Code	16	5	asc[4] or ui4	• 4 byte integer, typically representing 4 ascii characters, designating record type, null terminated, or used as ui4 value • 0 (all zeros = zero-length string) indicates no entry • There is one terminal record index with **Type String / Type Code** REC_Term_TYPE_CODE_m1x • REC_Term_TYPE_CODE_m1x is “Term” / 0x6d726554 (little endian) defined in medrec_m1x.h
Record Version Major	21	1	ui1	• Record type’s major version • 0xFF indicates no entry • The **Version Major** is 0xFF in the terminal record index
Record Version Minor	22	1	ui1	• Record type’s minor version • 0xFF indicates no entry • The **Version Minor** is 0xFF in the terminal record index

Field	Offset	Bytes	Type	Contents
Encryption Level	23	1	si1	• Does ***not*** change sign when corresponding record is encrypted / decrypted • See “Encryption Level Schema” table • The **Encryption Level** is 0 in the terminal record index

Segment (Sgmt) Records:

These records are not required, but are convenient when stored at the session or channel levels. They document segment start & end times and descriptions. Specifically they contain the following fields:

```
———————— Record Header START ————————
CRC:
Type Code/String: Sgmt
Version: 1.1
Encryption Level:
Total Bytes:
Start Time:
————————— Record Header END —————————
————————— Record Body START —————————
End Time:
Absolute Start Sample or Frame Number:
Absolute End Sample or Frame Number:
Segment Number:
Acquisition Channel Number:
Segment Description: (not required)
————————— Record Body END ——————————
```

They are useful (but not required) for finding a segment or range of segments based on time, or description. For example a segment may often be created within a longer continuous recording for the purpose of delimiting an experiment or other condition. Sgmt records stored in (non-segmented) ***session*** records are useful for this purpose. Sgmt records stored at the ***channel*** level make processing or sharing subsets of channels, without session data simpler for locating segments of interest. Because they are small and generally infrequent, they are often stored at both session & channel levels.

Sgmt records are of little utility in Segmented Session Records. If Segmented Session Records are implemented and Sgmt records are desired, Sgmt records should be kept in ***parallel*** non-segmented Session Records. This is the most efficient arrangement for long recordings with many segments and records.

Time Series Indices File Format

- Universal header
- Sequential time series index data
- The first sample in a recording is considered a discontinuity. Initial samples in subsequent segments are not necessarily discontinuities.
- The ***terminal index*** in each segment points to a virtual, non-existent block where:
    - File Offset = Time Series data file length
    - Start Time = estimated μUTC of the next sample (or segment end μUTC + 1)
    - Start Sample Number = number of samples in segment

Time Series Indices File:

Field	Offset	Bytes	Contents
Universal Header	0	1024	*See "Universal Header" description*
Time Series Index	1024	24	*See "Time Series Index Format" description*
...			

Time Series Index Format:

Field	Offset	Bytes	Type	Contents
File Offset	0	8	si8	• Offset to the beginning of the indexed CMP Block in the time series data file (in bytes). • Negative **File Offsets** indicate that the sample comes after a discontinuity. The file offset is the positive of this value. • The first sample in a *Channel (*but not necessarily a *Segment*) is always considered a discontinuity. • There is one terminal time series index with **File Offset** equal to the time series data file length
Start Time	8	8	si8	• *μ*UTC time of block start • If recording time offset is used for the session it is applied here also. • 0x8000000000000000 indicates no entry • There is one terminal time series index with **Start Time** equal to the estimated *μ*UTC of the next sample, assuming no discontinuity (or segment end *μ*UTC + 1)
Start Sample Number	16	8	si8	• Number of the first sample in the CMP Block relative to samples in the segment (*not the channel*). • 0x8000000000000000 indicates no entry • There is one terminal time series index with **Start Sample Number** equal to total segment samples

Video Indices File Format

- Universal header
- Sequential video index data
- The first frame in a recording is considered a discontinuity

- Frame numbering starts at zero for each segment, *but continues across video data file boundaries within a segment.*
- The ***terminal index*** in each segment points to a virtual, non-existent clip where:
    - File Offset = Video data file length
    - Start Time = estimated μUTC of the next frame (or segment end μUTC + 1)
    - Start Frame Number = number of frames in segment
    - Video File Number = the number of video files in the segment + 1

Video Indices File:

Field	Offset	Bytes	Type	Contents
Universal Header	0	1024		*See "Universal Header" description*
Video Index	1024	24		*See "Video Index Format" description*
...				

Video Index Format:

Field	Offset	Bytes	Type	Contents
File Offset	0	8	si8	• File offset to the start frame, typically a keyframe, depending on format • Negative **File Offsets** indicate that the frame comes after a discontinuity. The file offset is the positive of this value. • The first frame in a *Channel (*but not necessarily a *Segment*) is always considered a discontinuity. • There is one terminal video index with **File Offset** equal to the video data file length
Start Time	8	8	si8	• *μ*UTC time of first frame in clip. • If recording time offset is used for the session it is applied here also. • 0x8000000000000000 indicates no entry • There is one terminal video index with **Start Time** equal to the estimated *μ*UTC of the next frame, assuming no discontinuity (or segment end *μ*UTC + 1)
Start Frame Number	16	4	ui4	• The first frame in a video file is always indexed • Frame numbering starts at zero *for the first video file in a segment* • *The relative frame number within a specific video file can be calculated from the the frame number of the first video index for the specified video file (programmers see get_segment_video_start_frames_m1x() in library)* • 0xFFFFFFFF indicates no entry • There is one terminal video index with **Start Frame Number** equal to total frames in the segment • At 30 frames per second (the most common rate), the maximum continuous segment duration is ~4.5 years

Field	Offset	Bytes	Type	Contents
Video File Number	20	4	ui4	• Number of the video file within the segment • Combined with Segment Number and Base File Name to create the full file name of each video file • Numbering starts at one, *not zero* • 0 indicates no entry • There is one terminal video index with **Video File Number** equal to the number of video files in the segment + 1

Time Series Data File Format

- Universal header
- Sequential compressed (CMP) blocks
- Each block is 8-byte boundary aligned

Time Series Data Encryption

- Optionally the time series data can be encrypted with either Level 1 or 2 encryption
- The encryption uses AES-128 to encrypt the first 16 (typically most significant) bytes of the statistical model in each CMP compressed block.
- Encryption / decryption adds negligible time to data processing.

Time Series Data File:

Field	Offset	Bytes	Type	Contents
Universal Header	0	1024		See “Universal Header” description
CMP Block	1024	varies		See “CMP Block Format” description
...				

CMP (Compressed) Blocks

- Data are stored in compressed independent blocks.
- Blocks are structured in the following hierarchy:
  - Block Header
    - Fixed Region: *always present*, cast into CMP_BLOCK_FIXED_HEADER_m1x structure
    - Variable Region: *sometimes present*, depending on user choices
      - Records Region: user defined records
      - Parameter Region: up to 32 4-byte parameters
      - Protected Region: unstructured space for future MED development
      - Discretionary Region: unstructured space for file creator user
    - Model Region: *always present*, contains details required by codecs
  - Compressed Data
- In RED/PRED, raw data are differentiated. Derivatives are encoded in a single signed byte. If there is overflow, i.e > +127 or < -127, then a keysample is introduced, flagged by the reserved value -128 (0x80). The 4 bytes following the keysample flag contain the full undifferenced value of the (second) data point generating the overflow difference, as an si4. (Note if multiple derivatives are used, the keysample value will be that of the previous level of differentiation.)
- In RED/PRED, the differentiated data are statistically modeled, the model is stored in the CMP block header. RED generates 1 model; PRED generates 3 models.
- In RED/PRED, Range Encoding is used to compress the derivatives, using the statistical model(s).
- RED/PRED 1 vs 2: The compression concepts and data fields are the same in both versions, but the version 2 decodes substantially faster than the version 1. To achieve this speed increase, minor changes in the encoding strategy were necessary, requiring the second version. Version 2 is the preferred, and default, version of these algorithms. Version 1 is maintained for compatibility with MED data encoded prior to the version 2 algorithms.
- In MBE, the minimal bits required to encode all the samples in the block are encoded from the raw data.
- In MBE the data may be differentiated, but generally are not. The reason for this is that in order for MBE to compress better than RED/PRED, the data must be close to random, resulting in no between-sample temporal correlations and thus no reduction of the bit range. Differentiation in this setting adds computational time & complexity typically with no increased compression.

- Blocks are required to be 8-byte boundary aligned, and are terminally padded to an 8-byte boundary with the value 0x7E (tilde, “~”) as necessary. Pad bytes are included in the block bytes value, and in the block CRC.
- In compression, if the CMP_PROCESSING_DIRECTIVE detrend_data is set, each sample will be detrended prior to scaling and compressing. The gradient and intercept will be stored in the block header. This is a lossless operation, but has more utility in lossy compression.
- In compression, if the Amplitude Scale parameter flag bit is set, the (possibly offset) values will be divided by this value and rounded, prior to differentiating. This is a lossy operation.
- In decompression, if the Amplitude Scale parameter flag bit is set, the values of the samples will be multiplied by this value and rounded after integrating.
- In compression, if the Frequency Scale parameter flag bit is set, the (possibly offset) values will be downsampled by this value, prior to differentiating. This is a lossy operation.
- In decompression, if the Frequency Scale parameter flag bit is set, the values of the samples will be upsampled after un-differencing.
- In decompression, if the Gradient and Intercept parameter flag bits are set, data will be retrended after integrating and possibly scaling.

CMP Block Layout:

<table>
<tr><td colspan="2">CMP Block Header: Fixed Region</td></tr>
<tr><td rowspan="4">CMP Block Header: Variable Region</td><td>Records Region</td></tr>
<tr><td>Parameter Region</td></tr>
<tr><td>Protected Region</td></tr>
<tr><td>Discretionary Region</td></tr>
<tr><td colspan="2">CMP Block Header: Model Region</td></tr>
<tr><td colspan="2">CMP Block Compressed Data</td></tr>
</table>

CMP Block Format:

Field	Offset	Bytes	Type	Contents
CMP Block Header: Fixed Region Start				
Block Start UID	0	8	ui8	• Fixed value • Hexadecimal: 0x0123456789ABCDEF • Decimal: 81,985,529,216,486,895
Block CRC	8	4	ui4	• CRC of the remainder of block • If block encryption is used, it is performed before the CRC is calculated • 0 indicates no entry
Block Flags	12	4	ui4	• See CMP Block Flags table below • 0 indicates no entry
Start Time	16	8	si8	• μUTC time • If recording time offset is used for the session it is applied here also • 0x8000000000000000 indicates no entry
Acquisition Channel Number	24	4	si4	• Number of the channel in the original recording • -1 indicates no entry • Duplicated in Time Series Metadata

Field	Offset	Bytes	Type	Contents
Total Block Bytes	28	4	ui4	• Number of bytes in the compressed block including header and pad (boundary alignment) bytes • 0 indicates no entry
CMP Block Encryption Start • Block encryption is optional: specified in Block Flags • In RED/PRED encoding 16 byte blocks are encoded sequentially from here until a minimum of 16 bytes of compressed data have been encrypted (see below) • In rare cases there may be insufficient compressed data bytes to fulfill this requirement. In these cases encryption stops at the closest 16 byte boundary to the end of the block (including pad bytes). • In MBE encoding the entire block is encrypted to the last 16 byte boundary to the end of the block (including pad bytes)				
Number of Samples	32	4	ui4	• Number of data samples encoded in the block • 0xFFFFFFFF indicates no entry
Number of Records	36	2	ui2	• Number of records stored in the records region • Equals bits set in Parameter Flags • 0xFFFF indicates no entry
Record Region Bytes	38	2	ui2	• Number of records region bytes in the block • Range 0-65532 • Must be a multiple of 8
Parameter Flags	40	4	ui4	• See CMP Parameter Flags table below • Each bit corresponds to a 4-byte entry in the parameters region
Parameter Region Bytes	44	2	ui2	• Number of parameter region bytes in the block • Range 0-65532 • Must be a multiple of 4
Protected Region Bytes	46	2	ui2	• Number of protected region bytes in the block • Range 0-65532 • Must be a multiple of 4

Field	Offset	Bytes	Type	Contents
Discretionary Region Bytes	48	2	ui2	• Number of discretionary region bytes in the block • Range 0-65532 • Must be a multiple of 4
Model Region Bytes	50	2	ui2	• Number of compression model bytes in the block • Range 0-65535 • The model region is guaranteed to start on a 4-byte memory boundary
Total Header Bytes	52	4	ui4	• Number of bytes in the block header including fixed header, record, parameters, protected, discretionary, and model region bytes • 0 indicates no entry
CMP Block Header: Variable Region Start				

Record Region	56	Multiple of 8 bytes		• Must be a multiple of 8 bytes long
				• CMP Record Region Header:
				  • Bytes 0 - 3: record type code
				  • Byte 4 (ui1): record version major
				  • Byte 5 (ui1): record version minor
				  • Bytes 6-7 (ui2): record size

Parameter Region	*varies*	Multiple of 4 bytes		• 4-bytes for each parameter bit set • Accessed via parameter map in CPS • Contains reserved discretionary parameters
Protected Region	*varies*	Multiple of 4 bytes		• Reserved for future use • Must be a multiple of 4 bytes long • No required format
Discretionary Region	*varies*	Multiple of 4 bytes		• Discretionary end-user use • Must be a multiple of 4 bytes long • No required format
CMP Block Header: Model Region Start				

Field	Offset	Bytes	Type	Contents
Model Region	*varies*	*varies*		• See CMP Block Models Table • Model Region is considered part of the header, but not part of the variable region
Compressed Data				
Compressed Data	*varies*	*varies*	ui1	• Compressed data • A minimum of 16 bytes of compressed data will be encrypted if block encryption is performed (unless not available: see note with **Block Encryption Start** above
Alignment Bytes				
Pad Bytes	*varies*	*varies*	ui1	• 0-7 bytes as needed for 8-byte alignment • Value: 0xFE (ascii tilde, “~”)

CMP Block Flags:

Field	Name	Contents
General CMP Flags		
Bit 0	Discontinuity Bit	• 0 indicates no discontinuity • 1 indicates that this block began after a discontinuity in recording. • The first block in a session is always considered a discontinuity.
Bits 1 to 3	*Protected Bits*	• Reserved for potential future use
Bit 4	Level 1 Encryption Bit	• 0 indicates the block is not currently level 1 encrypted. • 1 indicates the block is currently level 1 encrypted. • The desired encryption level is set by the “encryption” field in the CMP_PROCESSING_DIRECTIVES. • This bit is mutually exclusive with “Level 2 Encrypted Block Bit” (bit 5)

Field	Name	Contents
Bit 5	Level 2 Encryption Bit	• 0 indicates the block is not currently level 2 encrypted. • 1 indicates the block is currently level 2 encrypted. • The encryption level desired is set by the “encryption” field in the RED_PROCESSING_DIRECTIVES. • This bit is mutually exclusive with “Level 1 Encrypted Block Bit” (bit 1)
Bits 6 to 7	*Protected Bits*	• Reserved for potential future use
Bit 8	RED1 Encoding Bit	• 0 indicates the block is not RED1 encoded • 1 indicates the block is RED1 encoded • This bit is mutually exclusive with bits 8-13
Bit 9	PRED1 Encoding Bit	• 0 indicates the block is not PRED1 encoded • 1 indicates the block is PRED1 encoded • This bit is mutually exclusive with bits 8-13
Bit 10	MBE Encoding Bit	• 0 indicates the block is not MBE encoded • 1 indicates the block is MBE encoded • This bit is mutually exclusive with bits 8-13
Bit 11	VDS Encoding Bit	• 0 indicates the block is not VDS encoded • 1 indicates the block is VDS encoded • This bit is mutually exclusive with bits 8-13
Bit 12	RED2 Encoding Bit	• 0 indicates the block is not RED2 encoded • 1 indicates the block is RED2 encoded • This bit is mutually exclusive with bits 8-13
Bit 13	PRED2 Encoding Bit	• 0 indicates the block is not PRED2 encoded • 1 indicates the block is PRED2 encoded • This bit is mutually exclusive with bits 8-13
Bits 14 to 23	*Protected Bits*	• Reserved for potential future use
Bits 24 to 31	*Discretionary Bits*	• Reserved for end-user use

CMP Parameter Flags:

Field	Name	Contents
Bit 0	Intercept Bit	• 0 indicates the block data was not offset prior to compression • 1 indicates the block data was offset prior to compression and the intercept value is stored in the Block Parameters Region • If this bit is set, this value will be subtracted from the data before compression and added back during decompression • This is used in conjunction with the Gradient bit • Ordinate intercept of the block's first order trend line (fit using least absolute deviations) • Units Conversion Factor is not applied to this number • This operation is **not** inherently lossy
Bit 1	Gradient Bit	• 0 indicates the block data was not offset prior to compression • 1 indicates the block data was offset prior to compression and the gradient value is stored in the Block Parameters Region • If this bit is set, this gradient will be subtracted from the data before compression and added back during decompression • This is used in conjunction with the Intercept bit • Slope of the block's first order trend line (fit using least absolute deviations) • Units Conversion Factor is not applied to this number • This operation is **not** inherently lossy
Bit 2	Amplitude Scale Bit	• 0 indicates the block amplitude was not scaled prior to compression • 1 indicates the block data was amplitude scaled prior to compression and the scale value is stored in the Block Parameters Region • If this bit is set, data is divided by the scale factor during compression and multiplied by it during decompression • This operation is inherently lossy

Field	Name	Contents
Bit 3	Frequency Scale Bit	• 0 indicates the block amplitude was not scaled prior to compression • 1 indicates the block data was amplitude scaled prior to compression and the scale value is stored in the Block Parameters Region • If this bit is set, data is divided by the scale factor during compression and multiplied by it during decompression • This operation is inherently lossy

Bit 4	Noise Scores	• Scores range 0-255:
		  • ui1s (1 byte each)
		  • 0 - 254 lowest to highest noise
		  • 255 denotes no entry
		• Four scores:
		  • Byte 0: Line Noise score
		  • Byte 1: Entropy score
		  • Byte 2: Normality score (Kolmogorov Smirnov)
		  • Byte 3: Local Linear Prediction score
		• Noise scores are calculated on the data that will be decoded, so if lossy encoding is used, the scores are calculated on the lossy data.

Bits 5 to 15	*Protected Bits*	• Reserved for potential future parameters
Bits 16 to 31	*Discretionary Bits*	• Reserved for end-user parameters

CMP Parameter Flags Usage:

The existence of a block parameter is indicated by a bit being set in the block parameter flags. If a bit is set, 4 bytes of space will be allocated in the parameter region for that value. If any parameters exist, a parameter map will be created in the CMP_processing_struct.

Access a block parameter as in the following examples:

```
//  get block parameters pointer
CMP_BLOCK_FIXED_HEADER_m1x       *bh;
ui4                              *params;
```

```
ui1                             *param_map;

bh = cps->block_header;
params = cps->block_parameters;
param_map = cps->parameters.block_parameter_map;

// get value (values are considered ui4s, so cast may be required)
intercept_value = *((si4 *) params + param_map[CMP_PM_INTERCEPT_INDEX_m1x]);


Where:
// Note: “CMP_PM_” prefix denotes CMP “parameter map”
CMP_PM_INTERCEPT_IDX_m1x = 0;
param_map[CMP_PM_INTERCEPT_IDX_m1x] = 0

If, for example, the data is not detrended, but is amplitude scaled, the amplitude
scale bit will be the first bit set. The map will function as follows:

sf4    amplitude_scale;

amplitude_scale = *((sf4 *) params + param_map[CMP_PM_AMPLITUDE_SCALE_IDX_m1x]);

Where:

CMP_PM_AMPLITUDE_SCALE_IDX_m1x = 2;
param_map[CMP_PM_AMPLITUDE_SCALE_IDX_m1x] = 0
```

To write a custom parameter (bits 16-31):

```
si4    PM_CUSTOM_PARAM_IDX = 16;
sf4    custom_value = 1.61803;

// set the parameter bit (before calling CMP_encode_m1x())
cps->parameters.discretionary_parameter_flags |= (1 << PM_CUSTOM_PARAM_INDEX);

// generate parameter map (encode)
CMP_encode_m1x();   // CMP_encode_m1x(); automatically creates parameter map if any
                    // parameter bit is set by calling
                    // CMP_generate_parameter_map_m1x(). If you call this yourself,
                    // realize that no further bits can be set, and the built-in bits
                    // are set by CMP_encode, depending on directives.

// set the value (parameters are considered ui4s, so we cast)
params[param_map[PM_CUSTOM_PARAM_INDEX]] = *((ui4 *) &custom_value);
```

To read a custom parameter (bits 16-31):

```
// generate parameter map (decode)
CMP_decode_m1x();   // CMP_decode_m1x(); automatically creates parameter map if any
                    // parameter bit is set in the block by calling
                    // CMP_generate_parameter_map_m1x(). You can also call this
```

```
                    // yourself

// get the value (parameters are considered ui4s, we have to cast to sf4)
custom_value = *((sf4 *) params + param_map[PM_CUSTOM_PARAM_INDEX]);
```

CMP Records Region Usage:

The records region exists to store record structures in the CMP block header. Typically these structures will contain information about the block data, but can contain anything. Block records are very similar to MED records with the following differences:

- The record header is defined by CMP_RECORD_HEADER_m1x instead of RECORD_HEADER_m1x. It is 8 bytes vs. 24 bytes (because the CMP_FIXED_BLOCK_HEADER_m1x contains the other relevant information)
- The CMP_RECORD_HEADER_m1x and RECORD_HEADER_m1x structures are very similar, *but not identical*:
  - Bytes 0 - 3 (ui4): record type code *(not null-terminated string)*
  - Byte 4 (ui1): record version major
  - Byte 5 (ui1): record version minor
  - Byte 6-7 (ui2): record size: *Maximum of 65535 bytes.*
  - Bytes 8 to end: content of entry plus pad bytes, if needed
- The body alignment requirement is 8 byte instead of 16 byte (because if encryption is desired, block encryption can be selected)
- Any standard MED record body can be used as a block record, but not vice-versa (i.e. if the body is 8-byte aligned, but not 16-byte aligned)
- Example: a MED “Stat” record could be included to store statistics for every block
- CMP record structures are define in medrec.c & medrec.h with the standard MED records

CMP Block Models:

Range Encoded Derivatives (RED)				
Number of Keysample Bytes	0	4	ui4	• The number of keysample bytes in the encoded block
Derivative Level	4	1	ui1	• The number of derivatives employed in encoding the data • 1 is default

Pad Bytes	5	3	ui1[3]	• Present to maintain 4 byte alignment • Filled with 0
Number of Statistics Bins	8	2	ui2	• Number of statistics entries minus one • Range 0-256 • Zero-count bins are not encoded
RED Flags	10	2	ui2	• Bit 0: encoded using No Zero Counts directive • Bit 1: positive derivatives only (range 1 to 255 instead of -127 to +127 • Bit 2: two-byte overflows • Bit 3: three-byte overflows
End RED Model Fixed Region (12 bytes)				
Derivative Initial Values	*varies*	*varies*	si4[*derivs*]	• One initial value for each derivative level
Statistics Bin Counts	*varies*	*varies*	ui2[*bins*]	• Statistical model of difference values for the block • There are no entries for zero count bins
Statistics Bin Values	*varies*	*varies*	ui1[*bins*]	• The difference values corresponding to the counts bins • There are no entries for zero count bins
Predictive RED (PRED)				
Number of Keysample Bytes	0	4	ui4	• The number of keysample bytes in the encoded block
Derivative Level	4	1	ui1	• The number of derivatives employed in encoding the data • 1 is default
Pad Bytes	5	3	ui1[3]	• Present to maintain 4 byte alignment • Filled with 0
Number of NIL Statistics Bins	8	2	ui2	• Number of statistics entries minus one • Range 0-255 • Zero-count bins are not encoded

Number of POS Statistics Bins	10	2	ui2	• Number of statistics entries minus one • Range 0-255 • Zero-count bins are not encoded
Number of NEG Statistics Bins	12	2	ui2	• Number of statistics entries minus one • Range 0-255 • Zero-count bins are not encoded
PRED Flags	14	2	ui2	• Bit 0: encoded using No Zero Counts directive • Bit 1: unused (to parallel RED header) • Bit 2: two-byte overflows • Bit 3: three-byte overflows
End PRED Model Fixed Region (16 bytes)				
Derivative Initial Values	*varies*	*varies*	si4[*derivs*]	• One initial value for each derivative level
NIL Statistics Bin Counts	*varies*	*varies*	ui2 [NIL *bins*]	• NIL statistical model of difference values for the block • There are no entries for zero count bins
POS Statistics Bin Counts	*varies*	*varies*	ui2 [POS *bins*]	• POS statistical model of difference values for the block • There are no entries for zero count bins
NEG Statistics Bin Counts	*varies*	*varies*	ui2 [NEG *bins*]	• NEG statistical model of difference values for the block • There are no entries for zero count bins
NIL Statistics Bin Values	*varies*	*varies*	ui1 [NIL *bins*]	• The difference values corresponding to the NIL counts bins • There are no entries for zero count bins
POS Statistics Bin Values	*varies*	*varies*	ui1 [POS *bins*]	• The difference values corresponding to the POS counts bins • There are no entries for zero count bins
NEG Statistics Bin Values	*varies*	*varies*	ui1 [NEG *bins*]	• The difference values corresponding to the NEG counts bins • There are no entries for zero count bins

Minimal Bit Encoding (MBE)				
Minimum Value	0	4	si4	• Minimum value of the samples in the block
Bits per Sample	4	1	ui1	• The number of bits employed in encoding each sample of the data • 0 indicates no entry
Derivative Level	5	1	ui1	• The number of derivatives employed in encoding the data • 1 is default
MBE Flags	6	2	ui2	• 0 indicates no entry
End MBE Model Fixed Region (8 bytes)				
Derivative Initial Values	*varies*	*varies*	si4[*derivs*]	• One initial value for each derivative level
Vectorized Data Stream (VDS)				
Number of VDS Samples	0	4	ui4	• Number of data samples encoded in the block • 0xFFFFFFFF indicates no entry
Amplitude Block Total Bytes	4	4	ui4	• Number of bytes in the compressed amplitude block including model and pad bytes
Amplitude Block Model Bytes	8	2	ui2	• Number of bytes in the compressed amplitude block model
Time Block Model Bytes	10	2	ui2	• Number of bytes in the compressed time block model
VDS Flags	12	4	ui4	• 0 indicates no entry
End VDS Model Fixed Region (16 bytes)				
VDS Amplitudes (headless RED, PRED, or MBE block) ***block amplitude scaling & detrending may apply***				
VDS Times (headless RED, PRED, or MBE block) ***positive derivatives applied***				