ALICE O2 time frame and metadata model

O2Message object:

- An O2Message object is a collection of annotated data payloads (O2DataBlock objects).
- An O2Message object travels atomically between O2 Devices: an O2Message object appears atomically in the O2Device. It is the job of the framework and transport to enable this.
- Each aggregate type in the O2 system (e.g. a time frame, a sub-time frame) is represented by O2Message and differs only by contents.

O2DataBlock:

- Self sufficient (self describing): the object contains full descriptor (HeaderStack) and the associated data payload.
- The associated payload is a contiguous byte array.
- The payload is immutable.
- The contents of the byte array in the payload are reinterpreted based on metadata contained in the header stack (HeaderStack).

HeaderStack:

- The header stack is a collection of headers describing the associated data payload.
- The header stack shall contain at least the DataHeader struct which provides the information necessary to decode the payload.
- The header stack may contain other O2 headers to carry additional metadata.
- The header stack occupies a contiguous sequence of bytes.
- There shall be no byte padding between the headers.

O2 compliant headers:

- Each O2 header shall start with a metaheader (BaseHeader) followed by a header body; the BaseHeader contains information needed to navigate the stack and decode individual O2 headers.
- The BaseHeader member shall be initialized with values appropriate for the header it is used in.
- The header body shall be POD or a serialized object.
- The binary layout of the BaseHeader is fixed to the following definition in C:

```
struct BaseDataHeader
{
```

```
  uint32_t fMagicNumber;          // 4 bytes
  uint32_t fHeaderSize;           // 4 bytes
  uint32_t fFlags;                // 4 bytes
  uint32_t fHeaderVersion;        // 4 bytes
  uint64_t fHeaderDescription;    // 8 bytes
  uint64_t fHeaderSerialization;  // 8 bytes
};
```

- The DataHeader required by the HeaderStack has the following layout:

```
struct DataHeader
{
  BaseDataHeader baseHeader;     // 32 bytes
  uint64_t fDataDescription[2];  // 16 bytes
  uint32_t fDataOrigin;          // 4 bytes
  uint32_t fReserved;            // 4 bytes
  uint64_t fDataSerialization;   // 8 bytes
  uint64_t fSpecification;       // 8 bytes
  uint64_t fPayloadSize;         // 8 bytes
};
```

  - fDataDescription member identifies the data type of the payload.
  - fDataOrigin member identifies the global origin (producer) of the data: e.g. a detector or software subsystem.
  - fDataSerialization member identifies the data serialization scheme used to encode the data (if any), e.g. ROOT.
  - Each data type (in the associated payload) is uniquely identified by the fDataDescription, fDataOrigin and fDataSerialization members.
  - fSpecification holds a data type dependent fine-grained specification, e.g. a link number for raw data.

Data layout per data block

Flat data types

For transient and raw data the default data representation is flat to keep the possibility open of direct access without serialization and deserialization overhead.

- A flat data type shall be a self contained POD; self containment means independence of the address space. In particular it means no pointer data members (no absolute addreses), this does not exclude internal referencing using other methods.
- A flat data type shall be annotated in DataHeader::fDataSerialization as “NONE”.

- References to data in other data blocks (e.g. relation between tracks and clusters) are based on index mapping.

Serialized data types

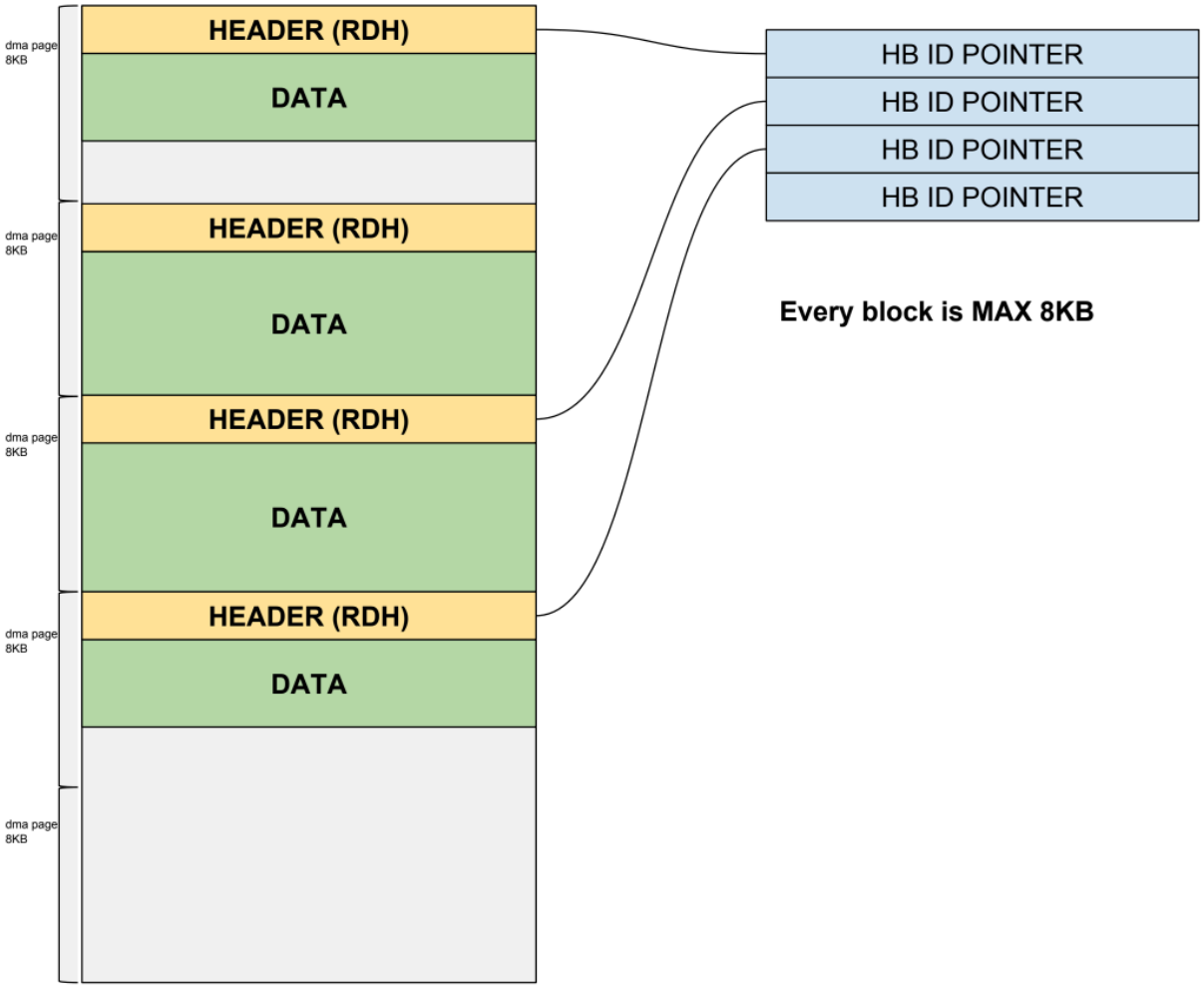
Serialization and deserialization incur a performance penalty. In data flows where the cost is acceptable, the following serialization schemes are in principle allowed.

ROOT objects

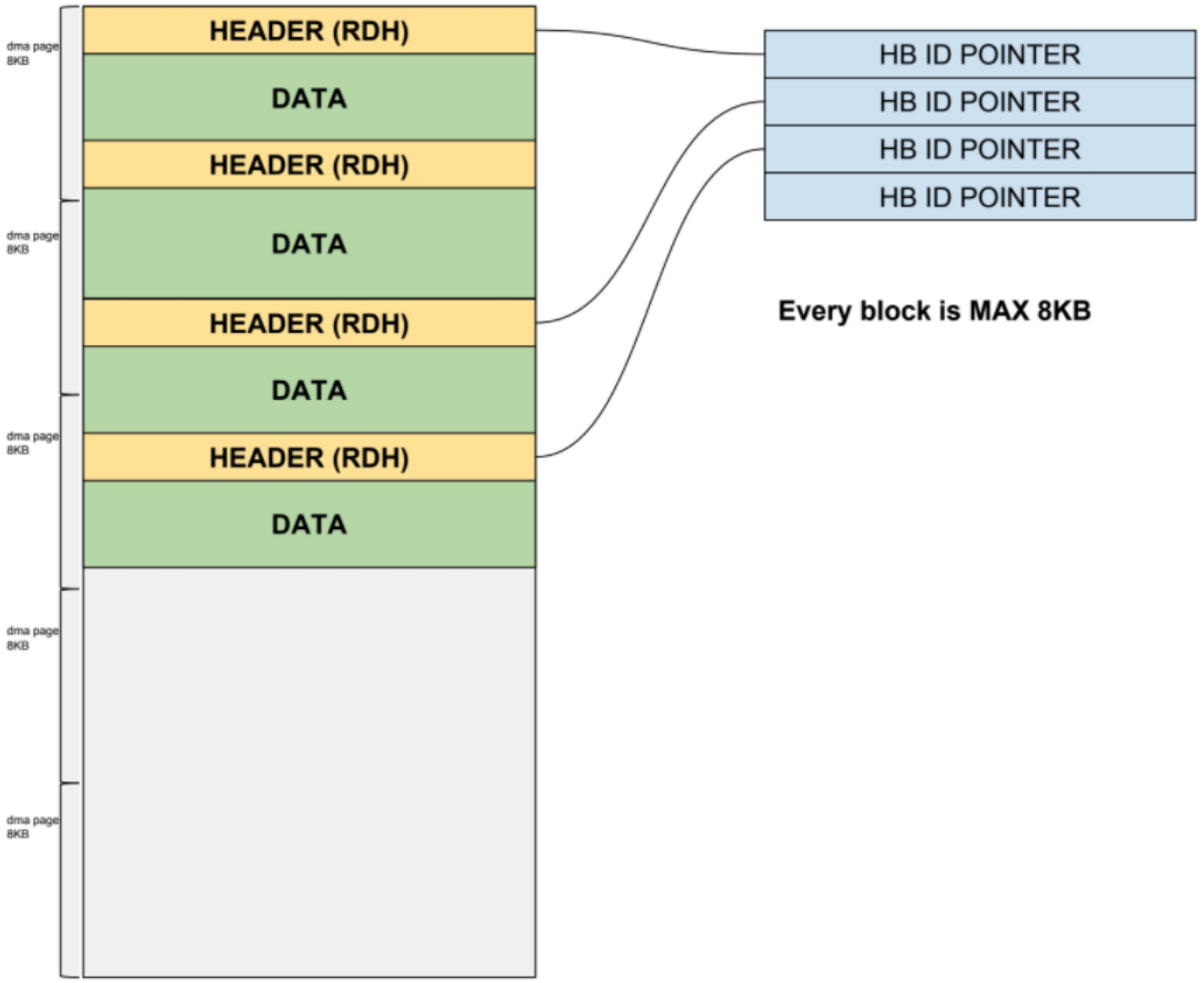
- ROOT objects are allowed; serialization and deserialization into/out of the payload handled by framework.
- A serialized ROOT object shall be annotated in DataHeader::fDataSerialization as “ROOT”.
- ROOT containers shall own their content after deserialization.
- The use of ROOT objects of non-fixed size in the reconstruction data chain is discouraged.

Raw data

- A raw data block is annotated with DataDescription “RAW” with DataOrigin set to the corresponding detector and Specification set to the corresponding link number.
- A raw data block payload consists of a sequence of annotated raw data payloads.
- Raw data payloads are annotated with the raw data header (RDH) placed directly in front of the data.
- Raw data payloads have maximum size of 8KB
- A table with a pointer to each HB ID can be generated by the software after the superpage has been filled with data
- A summary lookup table is provided.

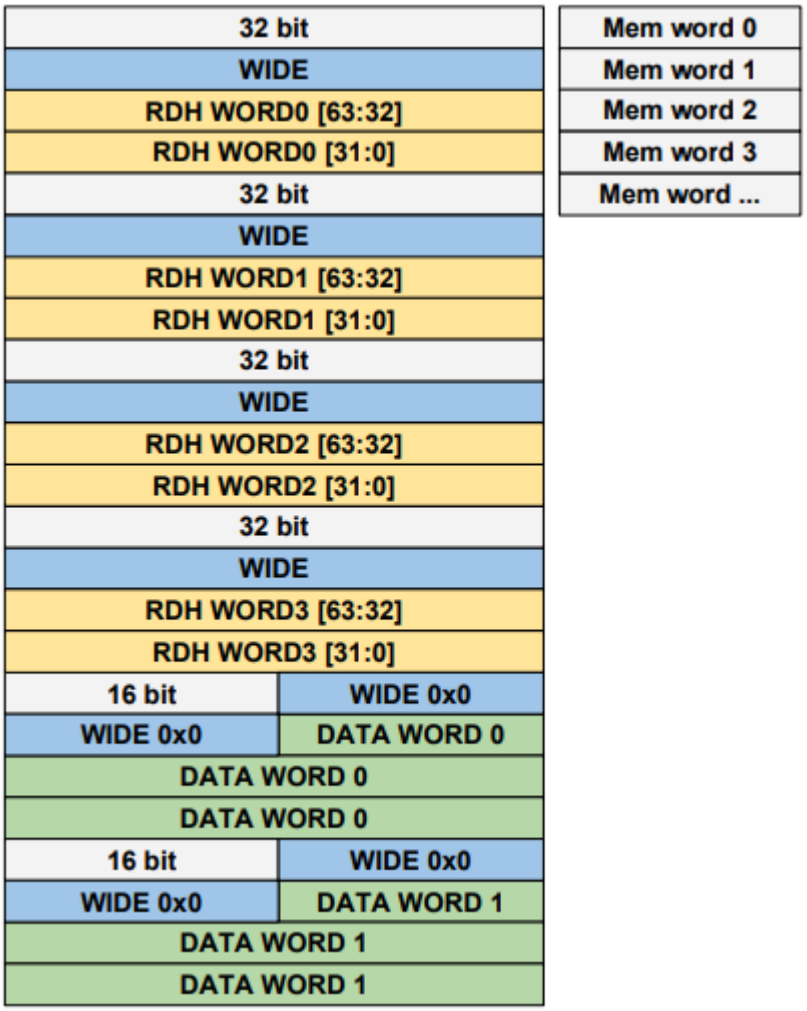


MEMORY layout 1.0: every Raw data payload starts at 8KB boundary. There could be holes between 2 consecutive Raw data payload



MEMORY layout 1.1: every Raw data payload is consecutive and there are no holes between 2 of them (this model is under studies)

GBT detector data in the memory of the FLP will look like (reading 32 bit words)

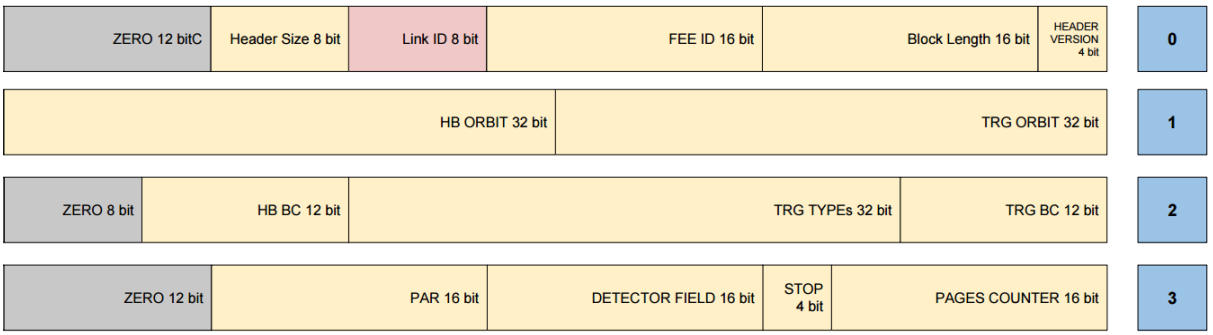


DDL detector data in the memory of the FLP will look like

0x0
0x0
RDH WORD0[63:32]
RDH WORD0[31:0]
0x0
0x0
RDH WORD1[63:32]
RDH WORD1[31:0]
0x0
0x0
RDH WORD2[63:32]
RDH WORD2[31:0]
0x0
0x0
RDH WORD3[63:32]
RDH WORD3[31:0]
DATA WORD0
DATA WORD1
DATA WORD2
DATA WORD3

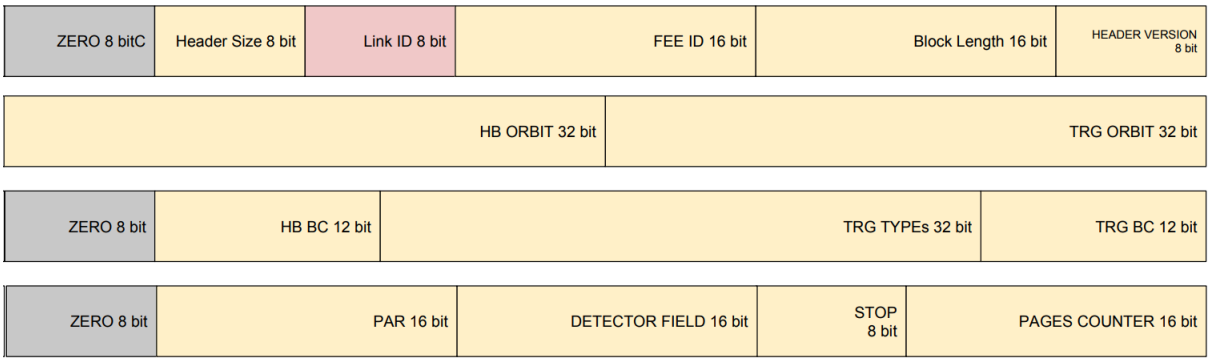
Raw data header V1

Every word in the RDH is 64 bit long, to be compatible between GBT and DDL based detectors.



	Reserved for future use
	Must be filled by det FEE (can be filled by CRU if FEE can't fill it properly)
	Must be filled by CRU

Raw data header V2



Main changes:

- **Header version** is 8 bit now to be byte aligned
- **STOP BIT** is 8 bit to be byte aligned

RDH fields description (still work in progress)

- **Block Length [16 bit]** : the length of the page (can be 0xFFFF if detector can't fill it properly).
- **FEE ID [16 bit]** : unique ID assigned to the FEE to know from which section of the detector data is coming (at least 18 bit).
- **The link ID [8 bit]** : used to identify from which GBT link in the CRU data is coming (0 - 23).
- **Header version [8 bit]** : version number to identify the header when new fields are added
- **Header Size [8 bit]** : number of 64 bit words composing the header
- **BC [12 bit] and Orbit [32 bit]** : the Trigger and HB identification.
- **TRG type [32 bit] :** trigger type set by CTP
- **Detector field [16 bit] :** detector specific field used by detector.
- **PAR [16 bit] :** field used by detector to trigger a configuration of the FEE

These 2 fields are needed only if the MEMORY DATA MODEL 1 is selected.

- **Pages counter [16 bit]** : counter to keep track of the different pages belonging to the same trigger.
- **STOP bit [8 bit]** : 1 bit to identify the last page (if there are more pages of 8KB belonging to the same trigger).

Data types in reconstruction

Base track model

- The result of reconstruction in different detectors which participate in tracking is non-virtual and pointers-less class containing as a data members
    - The track parametrization (final) class o2::Base::TrackParCov, providing track kinematics and its covariance matrix and methods for their propagation.
    - Extra general (like chi^2, nclusters used etc.) and detector specific (de/dx, TOF etc.) information to be defined by detector.

- For the exchange between different components of the same detectors experts may prefer transient data formats customized for their needs.

Transient data

- Transient data is temporary data used by intermediate reconstruction steps and it is never stored to disk (unless for debugging reasons).
- Transient data is produced and consumed on-the-fly by consecutive O2 devices (or within one device) by the same software version without schema evolution.
- Transient data types may change with new software versions.
- Transient data types shall be optimized for performance during reconstruction: they shall be flat data types, they shall favor short representations using data types supported by the processor natively (e.g. float versus double but no compressed custom precision floating point format).
- Transient data types may define an internal version number in it's metadata if multiple versions can be produced by the same software version.
- An example of a chain of transient data types used in the reconstruction is:
    - Input: TPC clusters (either from raw clusters or decompressed clusters)
    - 1st transient type: Spacial coordinates of TPC clusters with dummy z-coordinate to facilitate seeding:
        - float y, z;
    - 2nd transient type: Cluster links found in the cellular automaton phase:
        - int upwardLinkClusterID;
    - 3rd transient type: Sector tracks consisting of track parameters (X, Y, Z, SinPhi, DzDs, Pt), SignCosPhi, chi², number of clusters, array with one cluster index per TPC pad row or -1 otherwise, diagonal entries of cov. matrix plus selected off-diagonal entries (all 32 bit float or int).
    - 4th transient type: Fully reconstructed TPC track, same as 3rd type with following change:
        - Variable length lists of clusters, one with one cluster per row per leg of looper.
        - Additional list of (unassigned) clusters close to the trajectory used for track-model-compression.

        - Online dE/dx information per track.
    - Output: Final track as defined in the output data type below, dE/dx used for online gain monitoring histogram, list of clusters is used by the data compression device.

The intermediate data types are subject to change during development to meet optimizations needs. Input and output remains fixed.

Output data

- Output data types is output data of the reconstruction that may be stored to disk.
- They may support schema evolution, new software versions shall be backward compatible with previous output data type versions.
- Such data types are:
    - Hardware clusters consist of:
        - Header with version, number of clusters, and possibly time offset.
        - Array clusters as produced by the hardware cluster finder. C-struct with to be defined data types, can be either 32 bit float or custom length integers.
        - Cluster properties: row, pad, time, width in Y and Z, charge (total and max), flags for split and border clusters.
    - Compressed clusters consist of:
        - Compressed bitstream of encoded clusters produced by Huffman or arithmetic compression or likewise.
        - Version, data size in bits, and CRC contained in the metadata.

Data layout per detector

The table below links to the definitions of the data types (classes in the O2 repository).

detector	RAW	Intermediate (clusters,...)	Reconstructed
ACO			
CPV			
CTP			
EMC			
FIT			
HMP			
ITS			
MCH			
MFT			
MID			
PHS			
TOF			
TPC			
TRD			
ZDC			

AOD format

Preliminary work done in CWG4 is summarized in AOD TWiki.