Raw Data IO Library

April 6th 2017

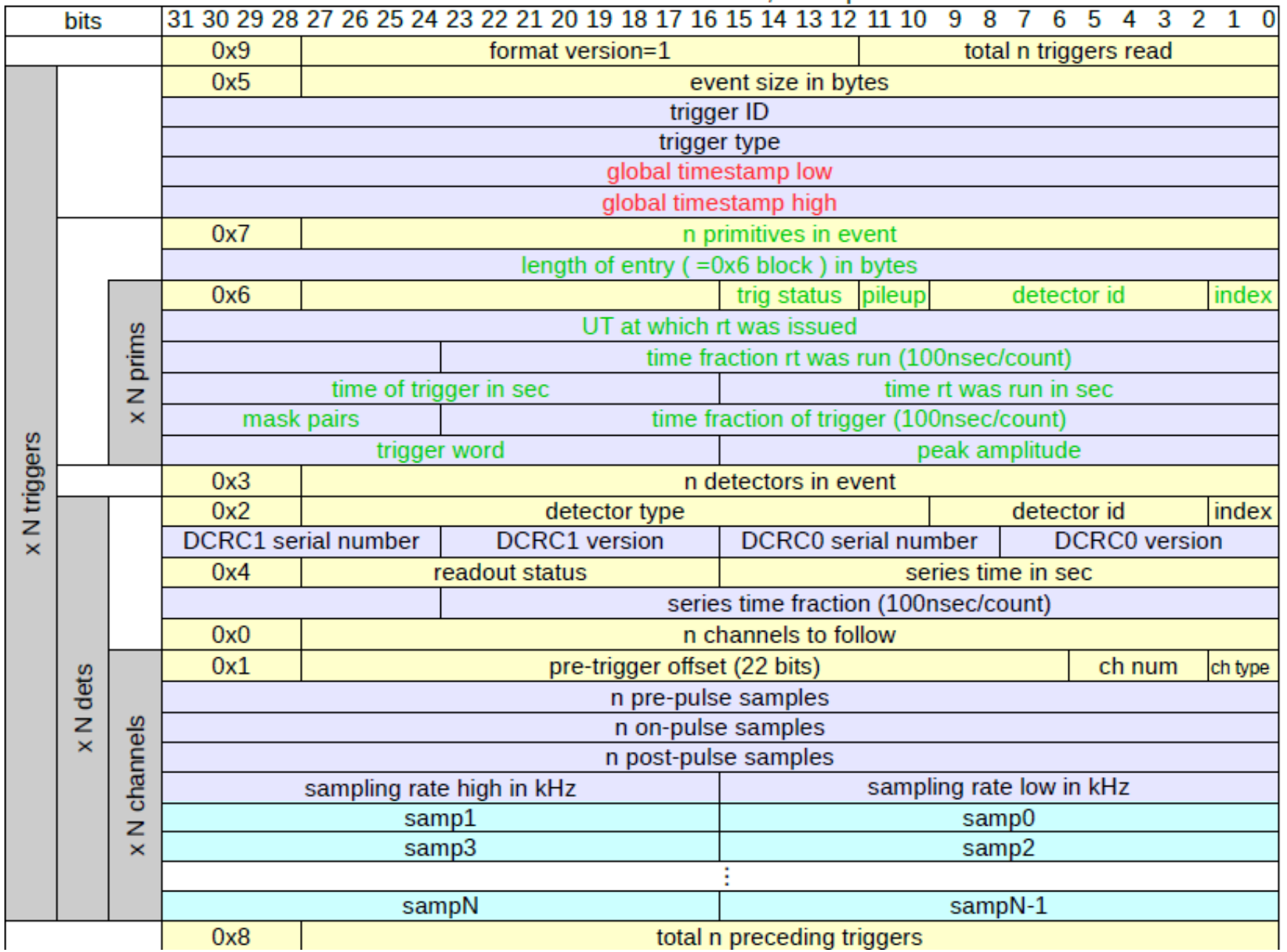
Matt Wilson

Purpose

- Standardize the output of SNOLab formatted data.
- Provide an simple tool for packing and unpacking raw data.
- Provide a library that can be used across multiple platforms (MidasDAQ, DMC).

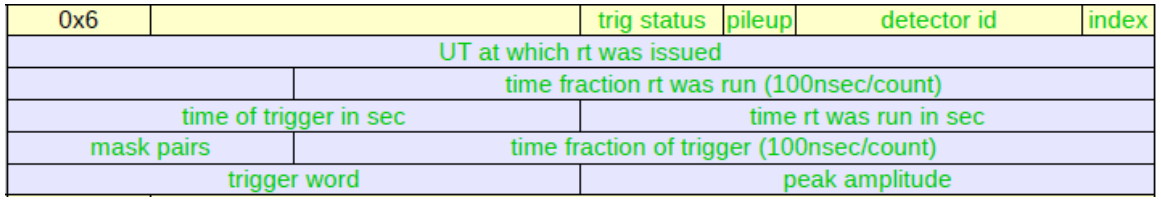
Author and Contact: Belina von Krosigk, UBC

DATA FORMAT VERSION 1: Created: 05. Feb. '16, Last updated: 30. Jun. '16



Terminology

- “Parameter” → an individual piece of information.
- “Buffer” → an array (or vector) that holds some data.
- “Block” → set of information containing one or more header blocks.
- “Header Block” → set of information containing a header and any data corresponding to that header.



= 0x6 header block
(naming temporary)

Basic Functionality of Library

- Get functions → retrieve certain parameters from library.
- Set functions → input certain parameters into library.
- Pack functions → packages information in the correct format into a block.
  - Packaged information is filled into empty array provided by user.
- Unpack functions → unpackages a block and resolves/sorts individual parameters.
- Clamping function → ensures parameters are the correct bit length.
  - Currently there are no measures taken if it is not the correct bit length.

“Low Level” Classes and Functions

- Able to pack, unpack *individual* header blocks
- Get and set functions for *individual* parameters within block
  - Individual parameters are private members within class

0x4	readout status	series time in sec
	series time fraction (100nsec/count)	

Parameters are private members in class

Header is 0x4

Two constructors, depending on whether you are reading/writing data

Pack data into empty buffer

```
class x4_header{
private:
    DWORD series_time_s;
    DWORD readout_status;
    DWORD series_timefrac_100ns;
public:
    static const unsigned char my_head_type = 0x4;

    x4_header(){ //constructor with no data array input
        readout_status = 0;
        series_time_s = 0;
        series_timefrac_100ns = 0;
    }

    x4_header(DWORD *data_array){ //constructor with data array input
        readout_status = clamp((data_array[0]>>16),bit12);
        series_time_s = clamp(data_array[0],bit16);
        series_timefrac_100ns = clamp(data_array[1],bit24);

        if (head_type(data_array[0]) != my_head_type){ //head type read from data must match head
type of class
            Error();
        }
    }

    void pack(DWORD *empty_buffer){ //pack information into DWORD output
        empty_buffer[0] = write_head(my_head_type)|(readout_status<<16)|series_time_s;
        empty_buffer[1] = series_timefrac_100ns;
    }
//get and set readout status
    DWORD get_readout_status(){
        return readout_status;
    }
    void set_readout_status(DWORD rs){
        readout_status = clamp(rs,bit12);
    }
//get and set series time in sec
    DWORD get_series_time_s(){
        return series_time_s;
    }
    void set_series_time_s(DWORD sts){
        series_time_s = clamp(sts,bit16);
    }
//get and set series time fraction (100ns/Count)
    DWORD get_series_timefrac_100ns(){
        return series_timefrac_100ns;
    }
    void set_series_timefrac_100ns(DWORD stf){
        series_timefrac_100ns = clamp(stf,bit24);
    }
};
```

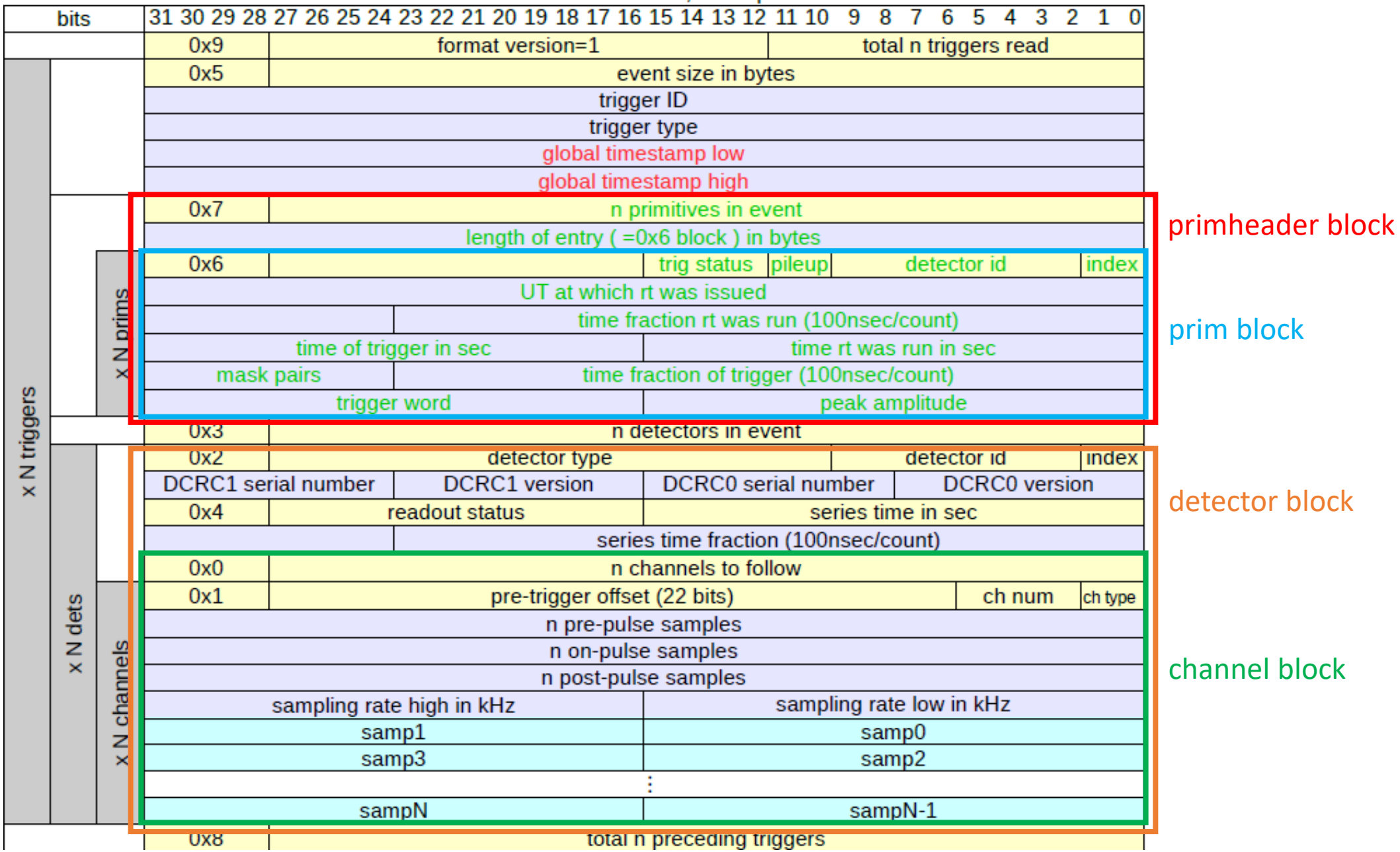
Additional ‘Super’ Functions

- A more convenient way of packing, unpacking larger blocks together.
- Uses objects and data structures that exist within CDMS DAQ.
- Uses low level classes/functions to format data and pack/unpack information correctly.

- Packing takes data in structures/objects, formats, packs into empty buffer.
- Unpacking takes formatted data, distinguishes individual parameters, puts back into known MidasDAQ structure object.

Author and Contact: Belina von Krosigk, UBC

DATA FORMAT VERSION 1: Created: 05. Feb. '16, Last updated: 30. Jun. '16

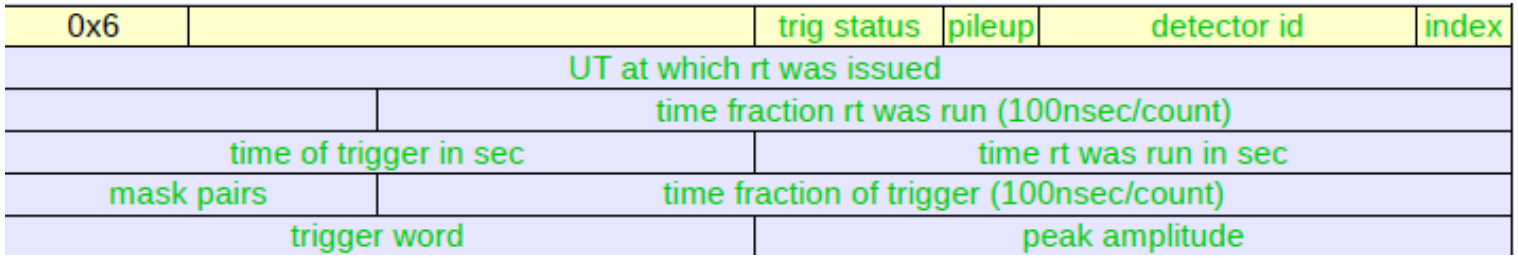


primheader block

prim block

detector block

channel block



Pointer to the CDMS DAQ class TRIGPRIM_BANK_DATA

Constructs 0x6 header

Sets parameters

Packages data into empty buffer

```
//pack prim block
void pack_prim_block(DWORD *empty_buffer, const TRIGPRIM_BANK_DATA *primdataptr){ //pass
pointer to empty buffer, pass pointer to TRIGPRIM_BANK_DATA class
    x6_header x6;
    x6.set_trigger_status((DWORD)(primdataptr->trigger_status));
    x6.set_piled_up((DWORD)(primdataptr->piled_up));
    x6.set_detector_id((DWORD)(primdataptr->fifo_list_header.detector_number));
    x6.set_index((DWORD)(primdataptr->fifo_list_entry.DCRC));
    x6.set_unixtime((DWORD)(primdataptr->fifo_list_header.unixtime));
    x6.set_rt_timefrac_100ns((DWORD)(primdataptr->fifo_list_header.phonptr_time_100ns));
    x6.set_trigger_time_s((DWORD)(primdataptr->fifo_list_entry.phonptr_time_s));
    x6.set_rt_time_s((DWORD)(primdataptr->fifo_list_header.phonptr_time_s));
    x6.set_mask_pairs((DWORD)(primdataptr->fifo_list_entry.maskpairs));
    x6.set_trigger_timefrac_100ns((DWORD)(primdataptr->fifo_list_entry.phonptr_time_100ns));
    x6.set_trigger_word((DWORD)(primdataptr->fifo_list_entry.triggerword));
    x6.set_peak_amp((DWORD)(primdataptr->fifo_list_entry.amplitude));

    x6.pack(empty_buffer);
}
```

Waveform Data

- Packing function(s) will have pointer to waveform data.
  - Waveform data will always be stored in user's code.
- Packing functions(s) will copy waveform data into the empty buffer.

```
std::copy(wfptr->data.begin(), wfptr->data.end(), emptybuffptr);
```

Pointer to waveform data

Pointer to empty buffer where waveform data is filled

- Waveform data is expected to be in the correct format.

Next Steps

Short term:

- Have IO Library operational for MidasDAQ.
  - Resolve any discrepancy issues between current MidasDAQ write out and IO Library.
  - Debugging/testing
  - Implement checks and balances (if necessary)
- Merge CDMS DAQ classes/data structures with IO Library.

Longer term:

- Have IO Library operation on DMC.
  - More involved, DMC currently does not write out to binary files.
- unpack_event functionality
  - Useful for offline platforms (like cdmsBats).