31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x9				format version=1																total n triggers read											

```
class overall_header {
        private:
                DWORD pformat_version;
                DWORD ptotal_triggers_read;
                DWORD pdata_line;
        public:
                //functions to write data:
                void set_format_version (int version);
                void set _total_triggers_read (DWORD ttr);
                DWORD get_overall_header (void);

                //functions to get data:
                void set_overall_header (DWORD input_data_line);
                DWORD get_format_version (void);
                DWORD get_total_triggers_read (void);

                //functions to read back private member values:
                DWORD readback_format_version (void);
                DWORD readback_total_triggers_read (void);
                DWORD readback_data_line (void)
}

//functions:
void overall_header::set_format_version (int version) {
        pformat_version = version;
}

void overall_header::set_total_triggers_read (DWORD ttr){
        ptotal_triggers_read = ttr;
}

DWORD overall_header::write_overall_header (void){
        pdata_line = 0x90000000 + (format_version << 12) + total_triggers_read;
        return pdata_line;
}

void overall_header::read_overall_header (DWORD input_data_line){
        pdata_line = input;
}

DWORD overall_header::get_overall_header_version(void){
        pformat_version = (pdata_line – 0x90000000 – ptotal_triggers_read) >>12;
        return pformat_version;
}
```

```
DWORD overall_header::get_overall_header_ttr(void){
	ptotal_triggers_read = pdata_line – 0x90000000 – (pformat_version <<12);
	return ptotal_triggers_read;
}

DWORD overall_header::readback_format_version(void){
	return pformat_version;
}

DWORD overall_header::readback_total_triggers_read(void){
	return ptotal_triggers_read;
}

DWORD overall_header::readback_data_line(void){
	return pdata_line;
}
```

31 30 29 28	27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12	11 10 9 8 7 6 5 4 3 2 1 0
0x5	event size in bytes	
trigger ID		
trigger type		
global timestamp low		
global timestamp high		

```
class trigger_header{
	private:
		DWORD event_size_bytes;
		DWORD trigger_id;
		DWORD trigger_type;
		DWORD global_timestamp_low;
		DWORD glocal_timestamp_high;
		DWORD data_line1;
		DWORD data_line2;
		DWORD data_line3;
		DWORD data_line4;
		DWORD data_line5;
	public:
		//functions to write data:
		void set_event_size_bytes(DWORD esb);
		void set_trigger_id(DWORD tid);
		void set_trigger_type(DWORD tt);
		void set_global_timestamp_low(DWORD gtl);
		void set_global_timestamp_high(DWORD gth);
		DWORD write_trigger_header1(void);
		DWORD write_trigger_header2(void);
		DWORD write_trigger_header3(void);
		DWORD write_trigger_header4(void);
		DWORD write_trigger_header5(void);

		//functions to read data:
		void read_trigger_header1(DWORD input_data_line1);
		void read_trigger_header2(DWORD input_data_line2);
		void read_trigger_header3(DWORD input_data_line3);
		void read_trigger_header4(DWORD input_data_line4);
		void read_trigger_header5(DWORD input_data_line5);
		DWORD read_event_size_bytes(void);
		DWORD read_trigger_id(void);
		DWORD read_trigger_type(void);
		DWORD read_global_timestamp_low(void);
		DWORD read_global_timestamp_high(void);

		//functions to readback private member values:
		DWORD readback_event_size_bytes (void);
		DWORD readback_trigger_id (void);
		DWORD readback_trigger_type (void);
		DWORD readback_global_timestamp_low (void);
```

```
                    DWORD readback_global_timestamp_high (void);
                    DWORD readback_data_line1 (void);
                    DWORD readback_data_line2 (void);
                    DWORD readback_data_line3 (void);
                    DWORD readback_data_line4 (void);
                    DWORD readback_data_line5 (void);
}

//functions:

void trigger_header::set_event_size_bytes(DWORD esb){
          event_size_bytes = esb;
}

void trigger_header::set_trigger_id(DWORD tid){
          trigger_id = tid;
}

void trigger_header::set_trigger_type(DWORD tt){
          trigger_type = tt;
}

void trigger_header::set_global_timestamp_low(DWORD gtl){
          global_timestamp_low = gtl;
}

void trigger_header::set_global_timestamp_high(DWORD gth){
          global_timestamp_high = gth;
}

DWORD trigger_header::write_trigger_header1(void){
          data_line1 = 0x50000000 + event_size_bytes;
          return data_line1;
}

DWORD trigger_header::write_trigger_header2(void){
          data_line2 = trigger_id;
          return data_line2;
}

DWORD trigger_header::write_trigger_header3(void){
          data_line3 = trigger_type;
          return data_line3;
}


DWORD trigger_header::write_trigger_header4(void){
          data_line4 = global_timestamp_low;
```

```
	return data_line4;
}

DWORD trigger_header::write_trigger_header5(void){
	data_line5 = global_timestamp_high;
	return data_line5;
}

void trigger_header::read_trigger_header1(DWORD input_data_line1){
	data_line1 = input_data_line1;
}

void trigger_header::read_trigger_header2(DWORD input_data_line2){
	data_line2 = input_data_line2;
}

void trigger_header::read_trigger_header3(DWORD input_data_line3){
	data_line3 = input_data_line3;
}

void trigger_header::read_trigger_header4(DWORD input_data_line4){
	data_line4 = input_data_line4;
}

void trigger_header::read_trigger_header5(DWORD input_data_line5){
	data_line5 = input_data_line5;
}

DWORD trigger_header::read_event_size_bytes(void){
	event_size_bytes = data_line1 – 0x50000000;
	return event_size_bytes;
}

DWORD trigger_header::read_trigger_id(void){
	trigger_id = data_line2;
	return trigger_id;
}

DWORD trigger_header::read_trigger_type(void){
	trigger_type = data_line3;
	return trigger_type;
}


DWORD trigger_header::read_global_timestamp_low(void){
	global_timestamp_low = data_line4;
	return global_timestamp_low;
}
```

```
DWORD trigger_header::read_global_timestamp_high(void){
		global_timestamp_high = data_line5;
		return global_timestamp_high;
}

DWORD trigger_header::readback_event_size_bytes(void){
		return event_size_bytes;
}

DWORD trigger_header::readback_trigger_id(void){
		return trigger_id;
}

DWORD trigger_header::readback_trigger_type(void){
		return trigger_type;
}

DWORD trigger_header::readback_global_timestamp_low(void){
		return global_timestamp_low;
}

DWORD trigger_header::readback_global_timestamp_high(void){
		return global_timestamp_high;
}

DWORD trigger_header::readback_data_line1(void){
		return data_line1;
}

DWORD trigger_header::readback_data_line2(void){
		return data_line2;
}

DWORD trigger_header::readback_data_line3(void){
		return data_line3;
}

DWORD trigger_header::readback_data_line4(void){
		return data_line4;
}

DWORD trigger_header::readback_data_line5(void){
		return data_line5;
}
```

- In the case of the overall_header, the data_line member will end up with the same information as the format_version and total_triggers_read members.
    - Similarly, without inspection of the class's implementation, it's not clear what the difference between the "read" and "readback" functions are. So do one or the other. Either store all of the "basic" information in separate members, or store the whole header DWORD and extract individual info on the fly. I would suggest the former makes more sense.
- Think about what is the natural form for data elements and provide access in that way.
    - For example, in the overall header, for format_version is really 16 bits. Is it signed? Probably not, but letting it be signed might allow negative values to indicate an error. In your setter you take an int. That may be OK, but you should make sure that it is properly cast in that case. Why not have both the member and getter/setter return uint16_t (which I think is equivalent to WORD in MIDAS)?
- Similarly, event_size in the trigger header shouldn't have a setter. It should be either read from a previously-written buffer or calculated.
    - Global_timestamp? The high and low words are just for packing into 32-bit wide arrays, but the end user doesn't care about that, we want to access it as a 64 bit integer.
- Once you get to waveforms where the event format can have variable size, the method you're taking now where you have a private member for each data line will break I think. I definitely don't want to know or care that the trigger header has 5 words, and I don't want to call write for each one. I want to be able to say trigger_header::pack_to_buffer(DWORD* bufferptr);

- A fairly common idiom for this sort of thing is to provide a constructor that takes a DWORD* pointing to the start of a memory buffer. This would be the most natural way to construct one of these objects from previously-written data. trigger_header trig(MIDAS::GetPointerToEventBuffer()); That gets around the question about whether you want to take an array as input to construct.