Event Code Sequencer Specification

Stephen Norum, Stephanie Allison,

Christopher O'Grady, Bruce Hill,

Sheng Peng, Ernest Williams

rev M

Oct. 25, 2013

1 Introduction

The Event Generator (EVG), the head of the LCLS timing system, sends event codes to Event Receivers (EVRs) at 360 Hz, in sync with the machine's 360 Hz fiducial. EVRs are configured to create interrupts to attached computers and/or output hardware triggers upon receipt of specified event codes.

This document describes the Event Code Sequencer (ECS), which is a software feature of the EVG which allows sequences of event codes to be programmed for use by experimenters to trigger hardware and software in desired patterns.

1.1 Hutch ID Numbers

As of the Aug 2013 proposal, we are adding new features to protect an LCLS experimental hutch which is conducting experiments from inadvertent side effects due to someone testing sequences in a different hutch to prepare for their experiment. These features will use a hutch number to identify the hutch. The current LCLS Hutch numbers are: 1 AMO, 2 SXR, 3 XPP, 4 XCS, 5 CXI, and 6 MEC.

1.1.1 Invalid Hutch Number

If a hutch ID is set to a number other than those above, the software will clamp that hutch ID to 0, signifying no hutch.

2 Event Generator and Event Codes

Each event code is assigned a single, unique, and constant delay value in the EVG Input Output Controller (IOC). These values define when event codes will be transmitted from EVG in 8.4 ns ticks relative to the machine's 360 Hz fiducial.

Event codes and their delay values are written to an EVG sequence RAM when they are to be transmitted. After receiving a fiducial trigger, the EVG walks through the sequence RAM and transmits event codes once their delays expire. Since event code delay values are unique and constant, active event codes are transmitted from the EVG and are received by EVRs with constant delays from the 360 Hz fiducial.

2.1 LCLS Experiment Event Codes

The Aug 2013 ECS update will add event codes 201 to 216, to the current event codes 67 to 98 and 167 to 198, for a total of eighty event codes reserved for experiments. The event codes' delay values increase by one tick, 8.4 ns, per event code, creating three contiguous blocks of event codes in the EVG sequence RAM.

2.1.1 Event code 140

Event code 140, an event code transmitted on LCLS timeslots when a beam pulse has been generated, is used as a start marker for experiment software and appears before experiment event codes in the EVG sequence RAM - that is, all experiment event codes have delay values greater than event code 140.

2.1.2 Event code 150
Event code 150 is sent by the EVG whenever a burst pulse is generated.

2.1.3 Event code 164
Event code 164 will be used to acknowledge each requested burst pulse, but only when the sequence owner is not the beam owner. If multiple burst pulses have been requested or if in continuous burst mode, event code 164 will be sent for each time slot when a pulse could have been generated. For example, if 5 pulses are requested, event code 164 should be sent 5 times with appropriate synchronization.

This event code can be used by developers looking to test their event sequences and scripts at times when they are not receiving beam. As this event code may be sent in response to a request from a different hutch, beam experiments should use event code 150 to confirm burst pulses. Testing can be done with event code 164 as long as your test can handle unexpected occurrences of event code 164.

2.1.4 Event code zero
In an event sequence, event code zero can be used for a step which adds delay, but doesn't fire any event code.

2.2 Event Code Allocation
Each ECS event code will have a new field that can be assigned a hutch number. When non-zero, that event code will be reserved for use by that hutch and event sequences being used by other hutches will not be allowed to use it. These hutch numbers will be initialized to zero, or unused. The hutch numbers will be preserved by autosave so they won't be lost if the EVG IOC restarts.

2.2.1 Event code Management
Allocation of the ECS event codes will be determined by the LCLS Photon hutch controls group, by the respective POC (controls and daq point-of-contact) representatives for each hutch. Day to day management of a pool of unassigned event codes will be done via an EDM screen that each POC can access as needed.

2.2.2 Beam Owner
A new PV will be added to the ECS IOC that will hold the hutch number which is currently receiving beam. Other hutches will be able to run their event sequences, but the burst control will only generate actual burst pulses if the sequence is assigned to the beam owner.

Updating the beam owner PV is for now up to the hutch POC or operator when they want to use the event sequencer during their beam time. This could be automated or handed over to MCC operators if needed in the future.

3 Event Code Sequences

Up to eight variable length sequences can be created. Sequences can be 1-2048 steps in length with each step consisting of an event code to transmit after an adjustable delay relative to the previous step.

Each sequence will have an assignable PV containing the number of the LCLS experimental hutch which is currently using that sequence.   In practice, we expect sequences 1-6 to be used by hutches 1-6 respectively, with sequences 7 and 8 available for hutches that need more than one sequence. However, to avoid problems, developers should be careful to avoid assumptions about the sequence number matching the hutch number.

Sequences can be started and stopped at any time. They can also be configured to repeat any number of times, including indefinitely.

3.1 5.1 Sequence Steps

Each step in the sequence supports four parameters:

- Event code
- Beam Delay
- Fiducial Delay
- Burst Count

3.1.1 Event Code Step Parameter

This parameter specifies what event code will be generated when this step in the sequence is performed. The event code specified must be assigned the same hutch id as the sequence, or the sequence will be invalidated. Event code 0 specifies no event code for this step.

3.1.2 Beam Delay Step Parameter

Delays are specified in the positive number of beam shots and 360 Hz fiducials from the previous shots. Event codes with zeroed delay times will be transmitted along with the previous step in order of increasing event code, a restriction imposed by the EVG (see Chapter 2). Beam delays are counted down at the machine's current beam rate. If the machine's rate drops to 0 Hz, sequences waiting on beam delays will pause. Negative beam delays are not supported.

3.1.2.1 Removal of prior restriction on non-zero delay for first step

Prior to the Aug 2013 proposal, the first step of a sequence was required to have a non-zero delay which was used to determine the delay when repeating the sequence.   That restriction is no longer required as each sequence has its own start phase to determine the next available start time, and any additional delay can be handled with a non-zero delay-only step.

3.1.3 Fiducial Delay Step Parameter

Fiducial delays are counted down at the machine's fiducial rate of 360 Hz, giving a 2.78 ms delay for each fiducial tick. The fiducial rate is not affected by beam rate. Negative fiducial delays are not supported.

If a step is assigned both a beam and fiducial delay, the beam delay is counted down before beginning the fiducial delay. That is, if a step is assigned a beam delay of three and a fiducial delay of two, the step's event code will be transmitted 5.56 ms after the third beam pulse since the previous step.

The EVG will continue playing sequences in the event beam is lost in the machine unknown to the EVG. Timestamps are provided to inform users of the last time each sequence has started and stopped.

3.1.4 Burst Count Step Parameter

Each step now supports a burst count. If burst count is zero, no burst pulses will be requested when the step is performed. If burst count is -1, a burst start will be generated initiating a continuous series of burst pulses. If burst count is -2, a burst stop will be generated, stopping burst pulses. For any positive burst count, the specified number of burst pulses will be requested. Event code 164 is generated each time a burst pulse has been requested by a non-beam-owner hutch, and event code 150 is generated each time a real burst pulse actually occurs.

If the sequence's hutch id is not the same as the beam owner, no burst pulses will be requested. However event code 164 will still be generated for each time slot that would normally produce a burst pulse.

3.2 Sequence Synchronization

Each sequence will have a programmable "start phase" settable via a SYNCMARKER EPICS PV, one for each sequence. This will allow the sequence to start on any of the common LCLS phases: 360 Hz, 120 Hz, 60 Hz, 30 Hz, 10 Hz, 5 Hz, 1 Hz, 0.5 Hz. The 360 Hz option will have lowest latency in the startup of the sequence (2.8 ms), while the 0.5 Hz option will have up to a 2 second startup latency.

3.3 Running a sequence without beam

As it is sometimes necessary to run a sequence with non-zero beam delays during times when there is no actual beam pulse, each sequence will have its own PV, BEAMPULSEREQ, to determine whether or not to require actual beam pulses or just use the 120Hz Event code 40 for the beam delay countdown.

3.4 Repeating a sequence

Each sequence will have a repeat mode PV with options: Once, Repeat N Times, and Repeat Forever, as well as a PV specifying the count for the Repeat N Times option. When set to repeat, after the final step is complete, the sequence will restart from the beginning with its assigned synchronization start phase. Any additional delays before the repeat can be handled by adding an additional delay step at the end of the sequence with event code 0.

3.5 Example Sequence 1

Table 5.1 contains an example sequence. This example assumes:

- a constant beam rate of 60 Hz
- SYNCMARKER set to 120Hz
- the sequence is started less than 2.78 ms after a beam pulse, i.e. before the next fiducial
- the sequence is configured to repeat forever.

Step	Event Code	Beam Delay	Fiducial Delay
1	67	0	0
2	68	1	0
3	70	0	2
4	69	0	0
5	71	2	1

Table 5.1. Example Sequence

Assuming the start request has time to process before the fiducial, the synchronization check is done immediately, and the sequence will start on the next 120Hz fiducial.   The sequence begins with event code 67 being output as soon as it's time slot occurs. Currently, event code 67 is sent 100,455 time slots after the 360Hz fiducial interrupt.   Each time slot is one cycle of the 119Mhz clock, around 8.4ns for a total delay relative to the fiducial interrupt of 844.16us.

Step two has a beam delay of one, and will output on the next beam pulse. Since the synchronization was set to 120Hz, we might get a beam pulse on the next 120Hz fiducial, or we might have to wait till the next 60Hz fiducial.   Thus, event code 68 will be sent either 16.6 ms or 33.3 ms after the start of the sequence.   To avoid this type of ambiguity, set your synchronization consistent with your beam rate.

The third step is delayed two fiducials from the previous step and will be transmitted 5.56 ms after step two. The fourth step is not delayed and will be transmitted along with step three. However, step four's event code will arrive at the EVR 8.4ns before step three's since its event code is a lesser value.

Since the beam rate is 60 Hz, and we have only had 2 fiducial delays since step 2, there will be 4 more fiducial delays before the next beam pulse. After one more beam pulse, step five's 2 beam delays will be exhausted and the fiducial delays begin. Step five's event code are transmitted on the fiducial following the second beam pulse after step four.

Because the sequence is configured to repeat and has not reached its repeat limit, the sequence begins again from step one.

Note that due to the fiducial delay of 1 on step 5, the sequence will restart after two more fiducials, on the next 120Hz fiducial, three fiducials before the next beam pulse.

3.6 Example Sequence 2

Table 5.2 contains an simple sequence that illuminates repeat rates.   This example assumes:

- a constant beam rate of 60 Hz
- SYNCMARKER set to 30Hz
- the sequence is started less than 2.78 ms after a beam pulse, i.e. before the next fiducial
- the sequence is configured to repeat forever.

Step	Event Code	Beam Delay	Fiducial Delay
1	83	6	0

Table 5.1. Example Sequence

Assuming the start request has time to process before the fiducial, the synchronization check is done immediately, and the sequence will start on the next 30Hz fiducial. Since the beam rate is 60Hz, we will see the first pulse on the same 30Hz fiducial. Call this fiducial number 0. We then skip the next 6 beam pulses. Since the beam rate is 60Hz, pulses come every 6 fiducials, (0, 6, 12, …), so the delay ends on fiducial number 36.

As fiducial number 36 is also a 30Hz fiducial, event code 83 is sent the first time on fiducial 36, with the following repeat sending 83 on fiducial 72, for a 10Hz repeat rate.

Note that if the beam delay was 5 in step 1, we would not get a 12Hz repeat rate as fiducial 30 is not a 30Hz fiducial. The 30Hz fiducials are 0, 12, 24, and 36, so we would still get a 10Hz repeat rate.

4 EPICS PV Names

Each event sequencer PV starts with a common prefix, noted below as $(P). Currently, the MCC EVG uses prefix "IOC:IN20:EV01" .

4.1 Global PV's, one per EVG

4.1.1 $(P):0:BEAM_OWNER_ID

This PV holds the hutch ID number for the hutch that is currently receiving beam. It would be nice if we can find a way to determine this number automatically, but likely we'll need to have each hutch responsible for setting this PV when they are taking beam and wish to use the event sequencer.

4.1.2 $(P):0:BEAM_OWNER_NAME

This PV holds the hutch name, an EPICS string. HUTCH_NAME is read-only as it is derived from HUTCH_ID.

4.2 Per sequence PV's

Each of the Event Code Sequences will have one each of the following PV's.

4.2.1 $(P):$(ID):SEQ

This PV holds the event sequence, with the ID macro being replaced by the sequence number. Each sequence has 4 fields, named A, B, C, and D, which are each signed 32 bit integer arrays holding up to 2048 values, one for each step. Field A holds the event codes for each step. Field B holds the beam delay, field C the fiducial delay, and field D the burst request count. Updating a sequence requires writing a new array to each field. For example, to update the event codes for each step, write the new array of event codes to $(P):$(ID):SEQ.A.

4.2.2 $(P):$(ID):LEN

This PV holds the number of steps in the sequence. You must update this PV each time you update the sequence steps to the correct number of steps being used. Subsequent values in the step arrays are ignored.

4.2.3 $(P):$(ID):HUTCH_ID
This PV holds the hutch owner id, a scalar unsigned 32 bit integer.

4.2.4 $(P):$(ID):HUTCH_NAME
This PV holds the hutch name, an EPICS string. HUTCH_NAME is read-only as it is derived from HUTCH_ID.

4.2.5 $(P):$(ID):SYNCMARKER
This PV holds the sequence synchronization value for each sequence.

- 0=0.5hz, 1=1hz, 2=5hz, 3=10hz, 4=30hz, 5=60hz, 6=120hz, 7=360hz

4.2.6 $(P):$(ID):BEAMPULSEREQ
This PV specifies whether or not to require actual beam pulses for the beam delay countdown. Write 0 to this PV for no pulses required, and write 1 to require actual beam pulses.

- 0=use appropriate timeslots (independent of beam). Steps will be executed with simulated 120hz beam rate, on the same timeslot as event code 40.
- 1=use pulses with beam

4.2.7 $(P):$(ID):PLYCTL
This PV is used to start and stop the sequence. Write 0 to stop and 1 to play.

4.2.8 $(P):$(ID):PLYMOD
This PV is used to set the play mode for the sequence. Write 0 for play once. Write 1 for Play N times, and write 2 for Play forever.

4.2.9 $(P):$(ID):REPCNT
This PV holds the repeat count for use when the play mode is "Repeat N Times".

4.2.10 $(P):$(ID):PLYCNT
This read-only PV updates as the sequence is played to show the play count.

4.2.11 $(P):$(ID):CURSTP
This read-only PV updates each step to show the current step number.

4.2.12 $(P):$(ID):ERRTYP
This read-only PV is an enumeration that shows the current validation state of the sequence. Value 2 is the string "Valid Sequence", and indicates a valid sequence. Other values are set to indicate errors with descriptive strings for each.

4.2.13 $(P):$(ID):ERRIDX
This read-only PV is used to show the step that has an error when the sequence is invalid. For a valid sequence, this value will be zero.

4.3 Per Event Code PV's
Each of the event codes allocated for event sequencer use will have one each of the following PV's:

4.3.1 $(P):0:EC_$(EC)_OWNER_ID

One of these is created for each ECS event code, substituting the event code number for the EC macro. This unsigned 32 bit integer contains the hutch number which currently is assigned that event code.

Appendix

The sections that follow contain comments and working notes from earlier revisions of the Event Code Sequencer Manual.

A. Sequencing Experience Jan 2010 to Dec 2010

Between Jan. 2010 and Dec. 2010 we have accumulated some running experience with the event sequencing. It has been critical for running LCLS experiments. It has become clear, however, that 4 changes are necessary. These are described in the following 4 paragraphs.

Having 8 event codes per group is not sufficient. The RSXS experiment in SXR needs 9, and as more complicated experiments are performed in the future we would predict more would be necessary. We believe 4 groups of 16 would be sufficient for the foreseeable future.

The 4 sets are useful because 2 experiments run at the same time (day shift and night shift) while 2 other experiments are setting up for the next run (typically the following week).

It is also important for the sequences to have a programmable "start phase" settable via an EPICS variable. This will allow the sequence to start on any of the common LCLS phases: 360 Hz, 120 Hz, 60 Hz, 30 Hz, 10 Hz, 5 Hz, 1 Hz, 0.5 Hz. The 360 Hz option will have lowest latency in the startup of the sequence (2.8 ms), while the 0.5 Hz option will have up to a 2 second startup latency.

The per-group terminator eventcode idea should be eliminated. We will use eventcode 1 as the terminator (this arrives at the beginning of the next fiducial). This is being done to decrease the complexity, at the cost of 2.8ms of PCDS DAQ EVR multicast packet delivery latency and 360 Hz CPU interrupts in the PCDS DAQ master crate. Users were frequently making errors not setting the correct terminator, and often non-sequencing eventcodes users were interested in recording in the DAQ system were coming after the terminator.

We will start to run most of the time with only fiducial delays, and beam delays will be set to zero. This is because we need to run sequences at times when event code 140 is not available (e.g. MD/ROD days).

B. New version in 2011

Around April 2011 new event sequencer EVG code was deployed that provided 16 event codes per ECS group, and supported a pv to specify the start phase for the sequence.   It also eliminated the need for per-group terminator codes.

C. Sequencing Experience Apr 2011 to Mar 2013

The event sequencer has been heavily used, and while it has been a useful tool, the following issues have been raised and led to this update of the event sequencer design.

1. Number one complaint is having to share 4 sequences among 6 active hutches.
2. Dividing the available 64 event codes into same size groups is too inflexible. Request is for each hutch to have a static range of event codes that are reserved for that hutch, along with the ability to use an additional range of event codes from the pool of event codes left after the static ranges are assigned.
3. Photon controls would like more flexibility in organizing and assigning event codes to hutches.
4. In 2012 a feature was added to allow burst control via special event codes in the sequences. Each event code sequence reserved a burst start event code and a burst stop event code. As we're currently limited to a pool of 64 event codes for 6 hutches, photon controls would like to see two dedicated event codes for this purpose, outside the range of 64 reserved for the sequencer. To avoid conflicts between a hutch taking beam and one testing it's sequences, a global "beam owner" pv would be assigned an enumerated value: AMO, SXR, XPP, XCS, CXI, or MEC. When the EVG sees the burst start or stop event codes, it would check the beam owner PV and only trigger a burst if that event code sequence was the beam owner.
5. The experimenter's would like to get an event code per pulse in response to burst requests, even when their hutch isn't taking beam. Currently, event code 150 signals a burst mode pulse, but if our burst controls are reserved to the beam owner, some other mechanism would be need to generate event codes for the hutches not currently taking beam.
6. The sync marker pv used to synchronize the sequence start to various beam rates currently is a global pv, IOC:IN20:EV01:ECS_SYNCMARKER, that can be written by any hutch. Options for the sync marker are: .5Hz, 1Hz, 5Hz, 10Hz, 30Hz, 60Hz and 120Hz. Current EVG software only supports one master sequence thread which is where the sync marker is implemented. This is a problem as others may want to test their event sequence scripts without affecting the timing for a hutch which is taking beam. Photon controls would like one sync marker PV per event sequence and to have the EVG use the sync marker PV for the beam owner to synchronize the sequence start times.
7. The beam request pv used to specify whether or not to run without actual beam pulses currently is a global pv, IOC:IN20:EV01:ECS_BEAMPULSEREQ, that can be written by any hutch. The beam request PV options are: TS4|TS1, or BeamFull. i.e. With or without beam. Photon controls would like one beam request PV per event sequence and to have the EVG honor the beam request PV for the beam owner and run the sequence without beam for the other hutches.
8. Two spare event code sequences beyond the 6 (one per hutch) are requested so a hutch could run two different event code sequence's at the same time.
9. If we could allow photon controls to run their own EVG, chained to the MCC EVG for synchronization, then we could just propagate the event codes we need and reclaim many more event codes for use in the hutches.

D. Proposal

Item's 1 and 8 are ready now, in that the latest EVG code supports 8 event code sequences.

Item 2 is not, as each of the 8 new sequences is assigned only 8 event codes.

Item 3 looks for more flexibility in allocating event codes.   There was a suggestion in our meeting that photon controls could handle both the allocation and the error handling when conflicts arise.   I think the allocation and bounds checking could be handled by a photon controls IOC, but I think we still need error checking in the EVG when it runs the sequences.   We can modify our event sequence client ioc to check event codes to see if they are allocated to the right hutch, but we also generate these sequence's via python scripts which write directly to the MCC EVG sequence arrays.   Given the chaotic nature of how python scripts are handed around and modified, I think it's essential to keep some sanity error checks in the EVG code.

Item's 4, 6, and 7, (burst ctrl, sync marker, and beam request), require a "beam owner" pv and changes to the EVG software to determine when to suppress burst controls or modify the start synchronization.

Item 5 will be difficult to provide based on discussions with Kukhee, as the burst support in the sequencer code just passes the requests on to the normal burst control logic.   We'd need to duplicate those controls for each hutch to make this work.

Item 9 is beyond the scope of this proposal.

E. First release

As developer time is in short supply and many of the top priorities are either resolved or nearly resolved, the first release should minimize additional changes so it can be deployed within the next month.

New features

- At least 6 ECS groups, preferably 8
- 2 event code ranges per ECS group: static and dynamic, fully controllable by PCDS
- Two event codes reserved for ECS burst start and stop, replacing burst codes in each ECS group.   (This is needed to make EVG not care where each ECS event code range starts and stops, as opposed to the current implementation in which those codes are specified in the EVG software.)
- Sync marker and Beam Pulse Request have separate PV's for each hutch, but only active for beam owner.

New PV's

- Beam owner PV, one per EVG
- Dynamic range minimum event code number, one per ECS group
- Dynamic range maximum event code number, one per ECS group
- Sync Marker PV, one per ECS group
- Beam Pulse Request PV, one per ECS group
- Hutch ID, same enum as beam owner pv, one per ECS group.   Needed if we're ever going to support a hutch using 2 different sequence's at the same time.

EVG changes required

- Add new pv's
- Modify event code range checks so codes can be in either the static or the dynamic range
- Replace ECS specific burst start/stop control with dedicated burst start/stop codes, arbitrated by the beam owner pv.
- Add check for beam owner before honoring a start phase PV

DAQ changes required

- Modify scripts and software to follow changes to event code assignments
- Adapt start phase controls to new ECS group specific PV names

Photon Controls changes required

- Meet to determine static event code allocations
- Modify sequencer IOC to conform to new PV's and support dual ranges
- Add beam owner PV monitoring and control to sequencer IOC
- Write a new IOC to manage the dynamic event code pool (Not required for initial operation.)
- Modify scripts as needed for new PV's

F. Second release

My initial thought was to leave the beam owner changes to the 2nd release, but when I realized that we needed to avoid hard coding the burst start and stop event codes I moved what look to be minor "beam owner" checks to the first release.   If implementation of the first set of features looks like it will take too long we can reconsider splitting the features into two releases.

That leaves item 5:

The experimenter's would like to get an event code per pulse in response to burst requests, even when their hutch isn't taking beam.   Currently, event code 150 signals a burst mode pulse, but if our burst controls are reserved to the beam owner, some other mechanism would be need to generate event codes for the hutches not currently taking beam.

I think we need a better idea of why this is needed and how we can reasonably implement it before finalizing a plan.