

Proposal for Representing in the Trigger Database
 Trigger lists allowed by Splitting and Oring
 Jim Linnemann, Marco Verzocchi, Igor Mandrechenko
 January 3, 2006; Revised Feb 17, 2006

As discussed in the requirements document, any trigger configuration can be written in terms of triplets of L1, L2, L3. The triplets are ordered by an index number entered by the triggermeister. Split triggers are naturally represented by triplets. The essence of this proposal is to use special values of L2 or L3 in the triplets in the database to extend the triplet representation to Or'd triggers. This trades some extra work in parts of the software against the software changes required for altering the database structure to support Oring more directly. As we descend to the implementation level, we will see that the syntax rules of the requirements document are implemented as constraints on ordering of triplets, and rules concerning the triplets allowed sharing a single L1 trigger. We discuss here mainly definitions of a single L1 trigger. An appendix discusses details of the resulting xml generation, and a new requirement which was discovered during testing.

1. Pure Splitting. In particular, all Split triggers can be expressed directly and without redundancy in this notation.

Index	L1 + [L2 + S[L3]] Triplets			S/O	L3 Group Name	Display/xml			S/O	L3 Group Name	L1bit	L2bit	L3bit
	L1	L2	L3			L1	L2	L3					
1	L1	A	a1	S	S1	L1	A	a1	S	S1	1	1	1
2	L1	A	a2	S	S1			a2	S	S1			2
3	L1	B	b1	S	S2		B	b1	S	S2		2	3
4	L1	B	b2	S	S2			b2	S	S2			4
5	L1	B	b3	S	S2			b3	S	S2			5
6	L1	C	c1	S	S3		C	c1	S	S3		3	6
7	L1	C	c2	S	S3			c2	S	S3			7
8	L1	D	d	S	S4		D	d	S	S4		4	8
9	L1	E	a1	S	S1		E	a1	S	S1		5	9
10	L1	E	a2	S	S1			a2	S	S1			10

The panel on the left indicates an entry-oriented view; on the right is closer to what would be displayed, or generated in xml; it shows more clearly the tree-oriented structure. The redundancy in this view is minimal: only the repeated L1 and L2 elements need to be re-entered. The S/O column is included only for clarity: it can actually be computed from the L1, L2, L3 triplet. In the case of pure splitting, only splitting elements are present. The L3 Group Name is also an optional feature. It would be advantageous if it is feasible to allow it to be appended to the fundamental trigger triplet (+ the index number entered by the triggermeister to impose correct ordering): groups of terms could be designated clearly for triggermeisters and users without having to carefully scan to see whether they were fully identical, and it might be possible to help support copy and paste in the user interface. At the simplest level, a low-level name could be computed from the triplet. The condition is that the group name must change when the L2 bit name changes. Generation of xml from the entries is straightforward, provided that the index numbers impose correct ordering. If the list were to be modified to insert a new L2 trigger, say L21D after index 7, the subsequent index numbers would have to be incremented accordingly. The bit numbers are counters to track usage of resources. This ordering is believed to be the order in which COOR will assign bit numbers, but in any case, COOR makes the final assignment, taking into account resources used for other runs. The L2 bit number tracked is the number of unique bits assigned in the 4096 mask, but does not necessarily correspond to the bit COOR would assign, as

some of these may be used for internal purposes. The L2 also produces a bit mask with entries numbered identically to the L1 bit (the confirmed-L1-bit mask actually used by the L3 to implement Oring).

2. **Pure Oring.** To allow a similarly direct representation of Or groups requires a mild extension of the notion of triplets. This extension requires a special Or script marker, which I here denote as *Or, to be recognized by the trigger database and the user interface. Although Or scripts could be represented as splitting triggers, they would be longer, and contain redundancies. This representation allows the redundancies to be removed. The implementation of the *Or markers is given in Appendix A.

L1 + [L2] + O[L3]				Pure Oring		Display/xml							
Index	L1	L2	L3	S/O	NAME	L1	L2	L3	S/O	NAME	L1bit	L2bit	L3bit
1	L1	A	*Or	O		L1	A		N		1	1	
2	L1	B	*Or	O			B		N			2	
3	L1	C	*Or	O			C		N			3	
4	L1	D	*Or	O			D		N			4	
5	L1	E	*Or	O			E		N			5	
6	L1	*Or	y1	O	O1		L2OR	y1	O	O1			1
7	L1	*Or	y2	O	O1		L2OR	y2	O	O1			2

Again, the S/O column can be computed from the triplet information: any triplet containing a *Or has been entered in order to express an Or. The entry order here begins with the L2 bits which comprise the Or, and finishes with the L3 terms comprising the Or group. A significant new constraint is needed to handle the change in notation: if a triplet with *Or in the L3 position appears, then definition of a L3 Or group is *required*. If a L3 group name is desired, the constraint is that the name must be defined when the first member of the group is defined. Again, a computer-generated name is possible if it is to be computed from the first L3 term in the group. Generation of xml from the left-hand entry form is again straightforward, provided the index numbers are correct. Modification of an existing trigger list by inserting another L2 term after line 5 would again require reassigning the index numbers.

3. Splitting with optional Oring: Mixed case 1.

Now let us see whether new issues arise when we consider a mixture of splitting and oring. First we consider adding an optional Or group to the splitting case. Here all L2 bits have L3 split groups, and addition a L3 Or group is assigned.

L1 + [L2 + S[L3]] + O[L3]					Splitting with Optional Oring									Display/xml		
Index	L1	L2	L3	S/O	NAME	L1	L2	L3	S/O	NAME	L1bit	L2bit	L3bit			
1	L1	A	a1	S	S1	L1	A	a1	S	S1	1	1	1			
2	L1	A	a2	S	S1			a2	S	S1			2			
3	L1	B	b1	S	S2		B	b1	S	S2		2	3			
4	L1	B	b2	S	S2			b2	S	S2			4			
5	L1	B	b3	S	S2			b3	S	S2			5			
6	L1	C	c1	S	S3		C	c1	S	S3		3	6			
7	L1	C	c2	S	S3			c2	S	S3			7			
8	L1	D	D	S	S4		D	d	S	S4		4	8			
9	L1	E	a1	S	S1		E	a1	S	S1		5	9			
10	L1	E	a2	S	S1			a2	S	S1			10			
11	L1	*Or	y1	O	O1		L2OR	y1	O	O1			11			
12	L1	*Or	y2	O	O1		L2OR	y2	O	O1			12			

This introduces no notational issues. The rules given above for recognizing Oring, and for group naming, and list modification are unchanged. One new sequencing issue arises: the user interface should allow introduction of an optional Or group after splitting entries have been defined. The meaning of this configuration is that the final Or group is applied in L3 when any of the individual L2 triggers pass.

4. Oring with optional Splitting: Mixed case 2.

Finally, we consider the most general case, that of Oring with optional Splitting. Here, the syntax allows a L2 condition with no split: it exists solely to form part of an Or group, though other L2 bits also have split groups.

L1 + [L2 + S{L3}] + O[L3]						Oring with Optional Splitting								
Index	L1	L2	L3	S/O	Name	L1	L2	L3	S/O	NAME	L1bit	L2bit	L3bit	
1	L1	A	a1	S	S1	L1	A	a1	S	S1	1	1	1	
2	L1	A	a2	S	S1			a2	S	S1			2	
3	L1	B	b1	S	S2		B	b1	S	S2		2	3	
4	L1	B	b2	S	S2			b2	S	S2			4	
5	L1	B	b3	S	S2			b3	S	S2			5	
6	L1	C	c1	S	S3		C	c1	S	S3		3	6	
7	L1	C	c2	S	S3			c2	S	S3			7	
8	L1	D	d	S	S4		D	d	S	S4		4	8	
9	L1	E	a1	S	S1		E	a1	S	S1		5	9	
10	L1	E	a2	S	S1			a2	S	S1			10	
11	L1	F	*Or	O			F		N			6		
12	L1	*Or	y1	O	O1		L2OR	y1	O	O1			11	
13	L1	*Or	y2	O	O1		L2OR	y2	O	O1			12	

The new ordering issue here is that after any pure-splitting entries, actually two possibilities arise: one may add L2 terms without a split L3 member, then the following L3 Or group may be entered. So the recommended entry order would be: Splits, L2 to be Or'd and the L3 Or group. It is not clear that this order is required, but it certainly renders xml generation and user displays straightforward. No other new constraints or notational issues arise from considering this case.

A natural but much more disruptive generalization is to allow the L3 slot in the triplets to be a pointer to a group of terms, so that the triplets would consist of L1, L2, L3group. That would give even more support to a secure copy by reference.

5. Two Consecutive L1 groups with same L1 condition: First has an Or group

L1 + [L2] + O[L3]						Pure Oring		Display/xml						
Index	L1	L2	L3	S/O	Name	L1	L2	L3	S/O	NAME	L1bit	L2bit	L3bit	
1	L1	A	*Or	O		L1	A		N		1	1		
2	L1	B	*Or	O			B		N			2		
3	L1	*Or	y1	O	O1		L2OR	y1	O	O1			1	
4	L1	*Or	y2	O	O1		L2OR	y2	O	O1			2	
5	L1	C	a1	S	S1	L1	E	a1	S	S1	2	3	3	
6	L1	C	a2	S	S1			a2	S	S1			4	

One new issue arises when considering the grouping of entries by L1 trigger. Currently with no branching in L2, the rule is simple: if the L1 or L2 term changes, then a new L1 bit is consumed and a new L1 group is formed. With branching this must be revisited. The examples so far have assumed implicitly that the next L1 group uses a different L1 trigger. What if this is not the case? This might happen because it is desired that the second group of triggers using the same L1 condition is treated differently for some reason. For example, it might be desired to allow different prescales. In this example we see that the rule can be easily extended by causing a break between L1 trigger groups at index 5, since the L3 Or group has ended.

6. Two Consecutive L1 groups with same L1 condition: First has no Or group

L1 + [L2 + S[L3]] + O{L3}

Splitting with Optional Oring

Index	L1	L2	L3	S/O	NAME	Display/xml			S/O	NAME	L1bit	L2bit	L3bit
						L1	L2	L3					
1	L1	A	a1	S	S1	L1	A	A1	S	S1	1	1	1
2	L1	A	a2	S	S1			A2	S	S1			2
3	L1	B	b1	S	S2		B	B1	S	S2		2	3
4	L1	C	c1	S	S3		C	C1	S	S3		3	4
5	L1	*Or	*Or			L1	E	A1	S	S1	2	4	5
6	L1	D	a1	S	S1			A2	S	S1		5	6
7	L1	E	a2	S	S1		L2OR	y1	O	O1			7
8	L1	*Or	y1	O	O1		L2OR	y2	O	O1			8
9	L1	*Or	y2	O	O1								

However, if the first L1 group has no terminating L3 Or group, a new issue arises. Here we desire to end the first L1 group at index 4, with index 6-9 forming a second group. So Index 5 contains a trigger containing no real information. The existence of the *Or markers in BOTH the L2 and L3 slots constitutes a second special marker which forces termination of the first L1 group. The new L1 group is signaled in the display by incrementing the L1 bit.

Summary of Rules

A. Required features

A.1 Index ordering of entries: Split entries, then L2 Or entries, then L3 Or entries.

A.2 L3 Split groups are of length 1 or more. Triplets with all three entries are members of split groups.

A.3 *Or in the L2 slot indicates that a L3 definition is there as a part of a L3 Or group.

A.4 *Or in a L3 slot indicates that a L2 definition is there *only* as a part of a L2 Or group. A L1 trigger group with such a definition *must* also contain a L3 Or group definition.

A.5 A terminating L3 Or group is always allowed for a L1 bit, but only required as in A.4

A.6 There is only one L3 Or group (of one or more entries) per L1 trigger.

A.7 L1, L2, and L3 bit numbers increment as each new element is introduced. These are not actual bit assignments, but running counts to track resource use. They appear in the report interface, but are not part of the database, nor of the generated xml. Coor assigns the actual bit numbers.

A.8 The index ordering must respect the grouping of Split and Or groups, and generation of xml directly follows from the index ordering.

A.9 Modification of a trigger list by insertion forces renumbering of index numbers and L2/L3 bit counting. Any such insertion must obey rules A.1-8. It is desirable (*but not strictly required?*) that the index renumbering be automatic.

A.10 A new L1 trigger condition starts a new L1 trigger group.

A.11 Two adjacent L1 trigger groups with the same L1 trigger are separated either by a L3 Or group at the end of the first group, or a *End marker in the L1 slot at the end of the first L1 group.

A.12 The features relevant to the Trigger List Report should also be reflected in the Skeleton Report.

A.13 The modified xmlgen should be backwards compatible:

A.13.a The UniqueL1L2 mode must be preserved, though it need not operate on new trigger lists including Oring.

A.13.b xmlgen should have a NoL2Branching mode which supports neither Splitting nor Oring, and produces identical xml as the old xmlgen for old trigger lists with neither feature included. Appendix B contains more details on backwards compatibility.

A14 The resulting xml is discussed in Appendix C.

B. Optional features (L3 Split/Or group names)

Group names (for groups of L3 terms) can help both triggermeisters and physics users to recognize blocks repeated across triggers. It would be useful for group names to be generated by the user interface (even as simple as numbers); it would be more desirable to allow users to create the group names, and to manipulate (e.g. copy and paste) groups.

B.1 If identical content does not guarantee an identical name, the main motivation for naming is lost.

B.2 Repetition of a **name** (within or across L1 trigger groups) must **guarantee identity of content**. This must remain true even if a trigger list is revised. If one of several “instances” of a group is altered, it must then be guaranteed to have a new name. *Name may be used slightly loosely to include a version number.* Useful names are inherently non-local if they represent re-used groups of items. Maintaining this property could either be the responsibility of the triggermeister (as assignment of the Index number is), or more ambitiously, the TDB or its user interface.

B.3 A Split Group name is defined at first usage, and may change when the L2 bit changes (i.e. a new split group begins). Each entire Split group has the same name.

B.4 An Or Group name is defined when the first element of the group is entered. The whole Or group has the same name.

Appendix A: Implementation of Special Markers

The L2 *Or marker is implemented as a special script named L2_OR_MARKER

The L3 *Or marker is implemented as a special script named L3_OR_MARKER

Appendix B: Backwards Compatibility

Backwards Compatibility is part of the implementation of Splitting/Oring. It is not necessary by the end of February (though that would be welcome and might be a best use of time) but certainly needed before the end of the shutdown.

1) Backwards compatibility mode for old scripts; perhaps controlled by a command line argument, say L2NoBranch: This is new functionality for xmlgen (though it simply preserves old functionality)
 default: allow more than 1 L2 attached to a L1 trigger normal new operation
 if -L2Nobranh: no Splitting, no Oring allowed special mode: old operation

2) UniqueL1L2 mode is required to be supported. This was already existing functionality; it should be verified that it still works:

for each L3 trigger, generate a separate L1 and L2 bit.

This is more restrictive still than L2NoBranch.

For each of these two special modes:

L2 Oring and L2 mixed split/or scripts are forbidden and should generate an error. The implementer may choose to allow (but ignore) the L1 *End separator, but *Or is not allowed either mode.

For each of these, L2 Splitting scripts will generate valid xml, but be interpreted differently under normal operation, and under these two modes. To see how these special modes will work, consider a set of scripts

Index L1
 1 L1Q L2a L3a

2	L1Q	L2a	L3b
3	L1Q	L2b	L3c
4	L1Q	L2b	L3d

UniqueL1L2 will generate xml for 4 L1 triggers (one per index line) L2Nobbranch on the other hand would generate xml for 2 L1 triggers, breaking between index 3 and 4.

Appendix C: xml generation and *Or null scripts

Here we discuss three items related to the generated xml file, and give a commentary on the xml generated by a test trigger. First we discuss the portion of the xml file aimed at the L3 parser, second we discuss how *Or null scripts interact with the regular xml, and the L3 parser part of the file, and finally we discuss a new attribute required in one item of the regular xml to maintain proper linkage between the two parts of the xml file. The fourth part of this appendix gives the input trigger list, and the fifth part gives the xml output with some interleaved commentary.

C.1 <triglist> part of xml file

The <triglist> part of the file is aimed at the L3 parser, and is not in true xml format. Part A consists of directives, and detailed information of items used by L3 scripts. It does not interact with splitting and oring, except that no content should be generated for terms of L3_OR_MARKER L3 scripts used solely to implement the *Or marker.

Part B consists of a series of three-part structures:

```
L2trig l2name
filter TriggerName
script term(s)
```

One of these structures is emitted for each real L3 script in the trigger. In principle, one such structure is emitted for each index line of the trigger list, since each line has a L3 script. However, no such structure is emitted for a line which contains a null L3 script implementing a *Or marker. The L2trig name, l2name, is based on the content of the index line containing the L3 script. Its purpose is to help the L3 parser recognize when it has received all the relevant COOR messages giving the associations of L2 bits, for which the L3 script is to be run when that L2 bit fires. For a split, this is the individual L2 bit.

For an Or, the reference is instead actually to the L1 bit (and this reference is repeated for each of the L3 scripts of the L3 or group), since the L3 scripts are each run whenever that L1 bit is confirmed by ANY of the L2 scripts (in the example, by either L2CALEM or L2CALDIEM). In our example, this name is generated by taking the trigger list index (line) number (5) and pre-pending it to the Trigger Name of Index line 5, where the Or group begins.

C.2 *Or “scripts” in the xml

Again a general rule is respected: L2 or L3 scripts used to implement *Or should not produce output about their details; they should only be used to generate the correct xml tree structure. The xml file annotations give an example if what would be (incorrectly) implied were a L3 *Or marker script be allowed to generate output: it would indicate a no-contents L3 script, which would produce a real L3 bit tied to a (probably) always-pass split script, while the intention was to produce no L3 split bit, but only to produce a L2 bit to be used as an input to an Or of L2 bits triggering an L3 Or group: L3 scripts each reported on their own bit, but which are attempted only when one of the L2 scripts of the L2 Or group pass.

In the regular xml, there is a line <l2trigger name='3^ARNOLD_OR1'>. This name is generated by taking the trigger list index (line) number (3) and pre-pending it to the Trigger Name of Index line 3. This is the current convention for the L2 trigger name. A most peculiar one, as it has no required relation to the L2

script name, but this convention was chosen solely to generate a unique name. (This convention may be revisited some time in the future). Such lines will be generated only for real (non-marker) L2 scripts, so this example has only L2 names for lines 3 and 4 in the xml section, while similarly, there are only <l3trigger> lines for lines 5 and 6, since those are the ones with real, non-marker, L3 scripts.

C.3 l2orname attribute of <l1trigger name> xml object

Notice that there is an apparent mismatch between the two parts of the xml file: the regular xml file carries l2names for the real L2 scripts, while the <triglist> part of the file contains references to the l2 name derived from the TriggerName of line 5, which does not have a real L2 script, only a *Or marker, and in fact this particular l2name therefore appears nowhere in the regular xml. As mentioned above, this “l2”name is a standing for a L2 bit associated with the or of all L2 scripts for the L1 bit. COOR needs to be told about this name in the regular xml in order to generate the right COOR directives to L3 to point the L3 or group to the correct L2 bit in the “L2-confirmed L1 bit” output mask. Therefore, a new item, l2orname, is added to the <l1trigger name> item in the xml. The name referenced there must be identical to the name used in the l2trig element of the <triglist> for each filter of the L3 or group. The good news is the computation is local in that it involves only a single trigger list line. The bad news is that it is used non-locally: it is inserted in the L1 xml structure, which begins at the first use of a new L1 bit.

C.4 A pure-splitting trigger list example

The report below is obtained from http://d0db-prd.fnal.gov/trigdb/cgi/tdb_report_element.py?function=Trigger_List&intlname=Test_L2oring&intlversion=1.00

The logical structure of this trigger is:

Index	TriggerName	L1	L2	L3
1(1)	Min_bias_nim_NCU	Afastz_ncu	None	Pf1
2(2)	Zero_bias_NCU	ALiveBX_ncu	None	Pf1
3(3)	Arnold_or1	TTK(1,10.)	L2CALEM	L3_OR_MARKER
4(3)	Arnold_or2	TTK(1,10.)	L2CALDIEM	L3_OR_MARKER
5(3)	Arnold_or3	TTK(1,10.)	L2_OR_MARKER	Mp_Ele
6(3)	Arnold_or4	TTK(1,10.)	L2_OR_MARKER	L3FEle

Notice that the (3) in the Index column corrects the included report by counting the L1 trigger bit correctly.

Trigger List Report

Trigger List Name input: [intlname , intlversion] = [TEST_L2ORING , 1.00]

TRIGGER LIST Name/Version= [Test_L2oring / 1.00](#), Use_Status= **unused**, Current_Status= **local**
 Implementation in: **primary** DAQ system, Configuration Type = **physics**, autopause= **yes**,
 comics_runtype= **data**, l3_type= **regular**, num_nodes= **0**, Trigger_count= **7**, Link to [RunsDB](#) using this
 TriggerList.

Created (Modified) by Pompos on 23-Jan-2006_16:37 (30-Jan-2006_10:43)

Description: **A simple trigger list to test pure L2 orring.**

Group 1 [allcrates / 1](#) **regular** **0**

L1 Cal Trigger Tower Programming ([L1Dialog](#)): [em3](#) [em9](#) [null](#)

L1 detector [Neotypes](#) : [CFT/CPS](#) [Calorimeter](#) [Special \(Named\) And/Or](#)
(Link to [Neoterms](#)) [ctt/2.00](#) [emcount/1.00](#) [specterm/1.00](#)

L2 filters: [none](#) [EM](#) [HT](#) [EM](#) [RANDOMPASS](#) [none](#)

L2 tools: [EM\(0,6.,0.,0.,0.,0.,1.,6.,0.,none,5,0,3,3,50\) / 1](#)

[EM\(0,120.,999.,999.,999.,999.,1.,1.,0.,none,5,0,3,3,50\) / 1](#) [COMMISSION / 1](#)

L3 filters: [PassFraction](#) [OR MARKER](#) [Ele](#) [Ele](#) [mp](#) [Ele](#) [Ele](#) [Ele](#) [mp](#) [PassFraction](#)

L3 tool s:	L3ERR_online / 2	GEO / 1	RUN_CFG / 1	CAL_UNP_NLC_NA_DA / 1	SmtUnp / 5	CFTUnp / 4
	GlobalTracker / 5	PrVTX3 / 2	CAL_CLUS4_PV3_NLC_ON / 2	NONE / 1	ELE_NLV_SHT / 2	PhTrk1 / 2
	IsoEle_SHT / 2	ELE_NLV / 2	ELE_NLV_SH / 2			

index	Trigger Name	Level 1	Level 2	Level 3
0	SRTOOLS_ONLINE / 5			This trigger definition includes a set of tools required by Level 3 ScriptRunner (a run configuration, an error handling tool and a geometry tool). Because it includes 'null' scripts at Level 1 and 2, it is not part of any specific trigger (a bit is not assigned), rather, it defines tools used by general programming instructions to Level 3 for this configuration to be listed before any trigger specific tools or filters in the element. This version has the error handling tool at the 'error' threshold. SRtools_online / 5

The following triggers belong to the same Exposure Group.

They share Device Group = [allcrates / 1](#) and Exposure related L1 And/Or Terms: [[ALiveBX](#) & NOT([ASkip0](#)) & NOT([Acaltc00](#))]

1(1)	min_bias_nim_NCU / 1	requires beam crossing and N/S luminosity monitors above threshold in coincidence and NOT unsuppressed Calorimeter read out. This is the same as min_bias_NCU with the nim to denote run I electronics. Afastz_ncu / 1	none / 1	pf1 / 1
2(2)	zero_bias_NCU / 2	requires beam crossing (an accelerator condition) and NOT unsuppressed Calorimeter read out ALiveBX_ncu / 1	none / 1	pf1 / 1
3(3)	ARNOLD_OR1 / 1	L1 : E13_ISHT22 / 2 L2 : E13_ISHT22 / 2 L3 : placeholder TTK(1,10.)_CEM(2,3)_CEM(1,9)_ncu / 1	L2CALEM(15,x) / 2	L3_OR_MARKER / 1
4(4)	ARNOLD_OR2 / 1	L1 : E13_ISHT22 / 2 L2 : E14_2L15_SH15L20 / 2 L3 : placeholder L2CALDIEM(18) / 1		L3_OR_MARKER / 1
5(5)	ARNOLD	L1 : E13_ISHT22 / 2 L2 : placeholder L3 : E13_ISHT22 / 2		

	OR3 / 1		L2 OR MARKER / 1	mp17000 Ele(ELE_NLV_SHT,1,22.,0.,3.6) Ele(IsoEle_SHT,1,22.,0.,3.6) / 2
6	ARNOLD OR4 / 1	L1 : E13_ISHT22 / 2 L2 : placeholder L3 : E14_2L15_SH15L20 / 2		L3FEle(ELE_NLV,2,15.,0.,3.6) L3FEle(ELE_NLV_SH,1,15.,0.,3.6) L3FEle(ELE_NLV,1,20.,0.,3.6) / 2
<p>The following triggers belong to the same Exposure Group.</p> <p>They share Device Group = allcrates / 1 and Exposure related L1 And/Or Terms: [ALiveBX & Acaltc00 & NOT(ASkip0)]</p>				
7(6)	Cal_unsuppressed / 3	A trigger to read out all Calorimeter channels in upsuppressed mode.		
	Acaltc00 / 1	none / 1	mp1 / 2	

C.5 xml with commentary

This is from the **Test_L2oring-1.00** triggerlist. The file generated by the XML generator needs three modifications:

1) add an attribute

[l2orname='5^ARNOLD_OR3'](#)

in the <l1trigger name='.....' > element for the L1 triggers which contain L2-oring (this is not needed for triggers which do not use L2-oring) . Note the correspondence between the name used for this new [l2orname='....'](#) attribute and the name of the L2 trigger

[L2trig 5^ARNOLD_OR3|](#)

[filter ARNOLD_OR3 |](#)

in the corresponding portion of the L3 instructions in the XML file.

(Is this possible without a two pass procedure ? Is the XML code for a given L1 trigger written after extracting all the information from the database ?)

2) remove the lines

[<l3trigger name='ARNOLD_OR1'/>](#)

and

[<l3trigger name='ARNOLD_OR2'/>](#)

from the L1/L2 portion of the XML file.

3) remove the lines

[L2trig 3^ARNOLD_OR1|](#)

[filter ARNOLD_OR1 |](#)

[L2trig 4^ARNOLD_OR2|](#)

[filter ARNOLD_OR2 |](#)

from the L3 instructions in the XML file.

The following is an XML generated from the trigger. It includes comments of the form [comment]...[end comment]. Linebreaks may not be accurate in this document.

```
<?xml version='1.0' encoding='US-ASCII'?>
```

```
<!DOCTYPE configuration SYSTEM "trigger_config.dtd">
```

```
<configuration autopause='yes' name='Test_L2oring' type='physics' comics_runtype='data' version='1.00'
```

```
physics='yes'
```

```
<!-- A simple trigger list to test pure L2 orring. -->
```

```
<!--This file was xmlGenerated using the following program options:
```

```
../xmlgen.2.py -tname Test_L2oring -tversion 1 -file -L2or yes-->&allcrates_readout;
```

```
<!--File allcrates_readout.xml must contain definitions of cratelists named allcrates and allcrates_novbd -->
```

```
<l1refsets>
```

```
<l1em_refset name='em3'>
```

```
Value 3.0</l1em_refset>
```

```
<l1em_refset name='em9'>
```

```
Value 9.0</l1em_refset>
```

```
<l1hadveto_refset name='null'>
```

```
Value 10000.0</l1hadveto_refset>
```

```
</l1refsets>
```

```
<level2>
```

```
<l2calem box_size='3' ieta_min='4' ieta_max='35'/>
```

```
<l2emtool mingsingletoweremfrac='1.' major_version='4' mingsingletoweret='6.'
```

```
name='EM(0,6.,0.,0.,0.,0.,1.,6.,0,none,5,0,3,3,50)' minneighboretafwdet='0.' minet='6.'
```

```
minneighborphicenet='0.' requirecps='0' requiretrack='0' minor_version='0' cpswindowiphi='3'
```

```
minneighboretafenet='0.' maxem='50' trackwindowiphi='5' minneighborphifwdet='0.' cpswindowieta='3'/>
```

```
<l2emfilter emfrac='0.' name='EM(0,0.,1.,15.,50,EM(0,6.,0.,0.,0.,0.,1.,6.,0,none,5,0,3,3,50))' minet='15.'
```

```
major_version='2' minor_version='0' isofrac='1.' maxem='50'
```

```
tool='EM(0,6.,0.,0.,0.,0.,1.,6.,0,none,5,0,3,3,50)'/>
```

```
<l2emtool mingsingletoweremfrac='1.' major_version='4' mingsingletoweret='1.'
```

```
name='EM(0,120.,999.,999.,999.,999.,1.,1.,0,none,5,0,3,3,50)' minneighboretafwdet='999.' minet='120.'
```

```
minneighborphicenet='999.' requirecps='0' requiretrack='0' minor_version='0' cpswindowiphi='3'
```

```
minneighboretafenet='999.' maxem='50' trackwindowiphi='5' minneighborphifwdet='999.'
```

```
cpswindowieta='3'/>
```

```
<l2emfilter emfrac='0.' name='EM(0,0.,1.,1.,2,EM(0,120.,999.,999.,999.,999.,1.,1.,0,none,5,0,3,3,50))'
```

```
minet='1.' major_version='2' minor_version='0' isofrac='1.' maxem='2'
```

```
tool='EM(0,120.,999.,999.,999.,999.,1.,1.,0,none,5,0,3,3,50)'/>
```

```
<l2commissiontool major_version='1' name='COMMISSION' minor_version='0'/>
```

```
<l2randompassfilter passpercent='100.' tool='COMMISSION' major_version='1' name='PASS100'
```

```
minor_version='0'/>
```

```
<l2htfilter
```

```
name='HT(0,18.,1,EM(0,0.,1.,1.,2,EM(0,120.,999.,999.,999.,999.,1.,1.,0,none,5,0,3,3,50)),PASS100,PASS100)'
```

```
major_version='1' nfilters='1' filter1='PASS100'
```

```
filter0='EM(0,0.,1.,1.,2,EM(0,120.,999.,999.,999.,999.,1.,1.,0,none,5,0,3,3,50))' filter2='PASS100'
```

```
minor_version='0' htmin='18.'/>
```

```
</level2>
```

```
<trigdef l3type='regular' num_nodes='0'>
```

```
<expogroup other_gs='allcrates_novbd' name='eg1_Test_1.00' readout='allcrates'>
```

```
<l1termlist>
```

```
<l1specterm name='live_accel_bx'/>
```

```
<l1specterm require='veto' name='skip_next_n_0'/>
```

```
<l1specterm require='veto' name='caltc00'/>
```

```
</l1termlist>
```

```
<l1trigger name='Afastz_ncu' prescale='0'>
```

```
<l1termlist>
```

```
<l1specterm name='fastz'/>
```

```
<l1specterm name='live_accel_bx'/>
```

```

<l1specterm require='veto' name='skip_next_n_0'/>
<l1specterm require='veto' name='caltc00'/>
</l1termlist>
<l2trigger name='1^min_bias_nim_NCU'>
  <l3trigger name='min_bias_nim_NCU'/>
</l2trigger>
</l1trigger>
<l1trigger name='ALiveBX_ncu' prescale='3400001'>
  <l1termlist>
    <l1specterm require='veto' name='caltc00'/>
    <l1specterm name='live_accel_bx'/>
    <l1specterm require='veto' name='skip_next_n_0'/>
  </l1termlist>
  <l2trigger name='2^zero_bias_NCU'>
    <l3trigger name='zero_bias_NCU'/>
  </l2trigger>
</l1trigger>

```

[comment] The l2orname attribute highlighted here is a new item, discussed above. Note this name must be identical to that used later as a l2trig reference for members of the L3 or group in the <triglist> part of the file.[end comment]

```

<l1trigger name='TTK(1,10.)_CEM(2,3)CEM(1,9)_ncu' prescale='0' l2orname='5^ARNOLD_OR3'>
  <l1termlist>
    <l1ctt name='TTK(1,10.)'/>
    <l1emcount count='1' hadveto_refset='null' em_refset='em9'/>
    <l1emcount count='2' hadveto_refset='null' em_refset='em3'/>
    <l1specterm require='veto' name='skip_next_n_0'/>
    <l1specterm name='live_accel_bx'/>
    <l1specterm require='veto' name='caltc00'/>
  </l1termlist>
  <l2trigger name='3^ARNOLD_OR1'>
    <l2script>
      <l2filter count='1' name='EM(0,0.,1.,15.,50,EM(0,6.,0.,0.,0.,1.,6.,0,none,5,0,3,3,50))'/>
    </l2script>

```

[Comment]

<l3trigger name='ARNOLD_OR1'/>

This line would be emitted if the L3 script at Index 3 had been a real script instead of a *Or marker, as it would have indicated a L3 script to be run if the L2 script L2CALEM at index 3 passed. Such a real L3 script would indicate a mixed split and or structure, and is allowed, but not included in this trigger list. The lack of the real script (*Or marker) indicates that the L2 script exists here not to support a split, but only as a member of a group to be Or'd; if either L2CALEM or L2CALDIEM pass, then the two real L3 scripts Mp_Ele and L3FEle will each be run.

[end comment]

```

  </l2trigger>
  <l2trigger name='4^ARNOLD_OR2'>
    <l2script>
      <l2filter count='1'
name='HT(0,18.,1,EM(0,0.,1.,1.,2,EM(0,120.,999.,999.,999.,999.,1.,1.,0,none,5,0,3,3,50)),PASS100,PASS1
00)'/>
    </l2script>

```

[Comment]

```
<l3trigger name='ARNOLD_OR2'/>
```

This line would be emitted if the L3 script at Index 4 had been a real script instead of a *Or marker, as it would have indicated a L3 script to be run if the L2 script L2CALEM at index 4 passed.

```
[end comment]
```

```
</l2trigger>
```

[comment] Here are the members of the L3 or group for this L1 trigger. Their name is simply the TriggerName on the index lines defining them. These

```
<l3trigger name='ARNOLD_OR3'/>
```

```
<l3trigger name='ARNOLD_OR4'/>
```

```
</l1trigger>
```

```
</expogroup>
```

```
<expogroup other_gs='allcrates_novbd' name='eg2_Test_1.00' readout='allcrates'>
```

```
<l1termlist>
```

```
<l1specterm name='live_accel_bx'/>
```

```
<l1specterm require='veto' name='skip_next_n_0'/>
```

```
<l1specterm name='caltc00'/>
```

```
</l1termlist>
```

```
<l1trigger name='Acaltc00' prescale='0'>
```

```
<l1termlist>
```

```
<l1specterm name='caltc00'/>
```

```
<l1specterm require='veto' name='skip_next_n_0'/>
```

```
<l1specterm name='live_accel_bx'/>
```

```
</l1termlist>
```

```
<l2trigger name='7^Cal_unsuppressed'>
```

```
<l3trigger name='Cal_unsuppressed'/>
```

```
</l2trigger>
```

```
</l1trigger>
```

```
</expogroup>
```

```
<triglist>
```

```
[comment] This begins part A of the <triglist> [end comment]
```

```
SRDirective (useL2=yes)|
```

```
SRDirective (monitorinfo=10)|
```

```
SRDirective (sendmoninfo=yes)|
```

```
SRDirective (allowInclusiveMonitorStream=yes)|
```

```
L3ERR_online L3ErrHandle(
```

```
  logfile="L3_SR_tsim.log",
```

```
  statsfile="L3_SR_stat.log",
```

```
  filethreshold="error",
```

```
  statsthreshold="info",
```

```
  tooltype="ErrorHandle",
```

```
  port=52245,
```

```
  host="d0o139.fnal.gov",
```

```
  exename="Snode")|
```

```
RUN_CFG L3RunConfigMgr(
```

```
  runcfg_file="cfg.dat",
```

```
  tooltype="utility")|
```

```
GEO L3GeometryManagement(
```

```
  tooltype="utility",
```

```
  RunNo=-1,
```

```
  MCTag="xxx.xx.xx")|
```

```
SmtUnp L3TSmtUnpack(  
  chanthreshold=.023,  
  maxstrips=33,  
  basegeom="SiBaseGeometry",  
  channelgeom="SiChannelGeometry",  
  adcthreshold=7,  
  clusterthreshold=.028,  
  tooltype="unpack")|  
CFTUnp L3TCFTUnpack(  
  tooltype="unpack",  
  adcthreshold=20,  
  maxfibers=8,  
  basegeom="CftBaseGeometry",  
  channelgeom="CftChannelGeometry",  
  realdata=TRUE)|  
GlobalTracker L3TGlobalTracker(  
  SMTUnpack=SmtUnp,  
  CFTUnpack=CFTUnp,  
  UseSMT=TRUE,  
  DoStereo=TRUE,  
  tooltype="data")|  
CAL_UNP_NLC_NADA L3TCalUnp(  
  RUN_CFG="RUN_CFG",  
  trgthrsh=0.,  
  unpthrsh=0.,  
  nbrad=1,  
  unpall=0,  
  caltype="MC_PLATE",  
  L3Nada=1,  
  L3NadaThreshold=3.,  
  Calib=1,  
  dataVersion="latest",  
  tooltype="unpack")|  
PrVTX3 L3TCFTVertex(  
  TRACKER=GlobalTracker,  
  ptcut=3.,  
  UsePt=FALSE,  
  tooltype="data")|  
CAL_CLUS4_PV3_NLC_ON L3TCalCluster(  
  calunp=CAL_UNP_NLC_NADA,  
  vertex=PrVTX3,  
  conesize=.4,  
  MinSeedEt=.5,  
  tooltype="data")|  
ELE_NLV_SHT L3TEle(  
  CAL=CAL_CLUS4_PV3_NLC_ON,  
  EMFR=.9,  
  Cal_Cps_dphi=.1,  
  Cal_Cps_dz=100.,  
  Cal_Cps_deta=.1,
```

```

    Cal_Cps_dcut=-.1,
    Cps_Track_dphi=.1,
    Cps_Track_dz=5.,
    Cps_Track_dcut=-.1,
    Cal_Track_dphi=.1,
    Cal_Track_deta=.1,
    Cal_Track_dcut=-.1,
    CALUNP=NONE,
    TRACK=NONE,
    CPS=NONE,
    CHI2=-100.,
    EMSEED=NONE,
    isolation=-1.,
    ConeSize=.25,
    Width_EM1=1.8,
    Width_EM2=1.4,
    Width_EM3=1.15,
    Width_ECEM1=1.,
    Width_ECEM2=1.,
    Width_ECEM3=1.2,
    tootype="physics")|
PhTrk1 L3TPhysTracker(
    TRACKER=GlobalTracker,
    ptcut=1.,
    highestNtracks=50,
    MinXYHits=10,
    MinZHits=0,
    tootype="physics")|
IsoEle_SHT L3TIsolation(
    tootype="physics",
    srctrackrefset=ELE_NLV_SHT,
    vertexrefset=PrVTX3,
    calunprefset=CAL_UNP_NLC_NADA,
    swarmtrackrefset=PhTrk1,
    forcelocalmuon=0,
    calcone_r=0.,
    calcone_core=.1,
    cal_e_hcone=999.,
    rapproach=.4,
    rapproach_min=.05,
    maxptsum=1.,
    maxtracks=20)|
ELE_NLV L3TEle(
    CAL=CAL_CLUS4_PV3_NLC_ON,
    EMFR=.9,
    Cal_Cps_dphi=.1,
    Cal_Cps_dz=100.,
    Cal_Cps_deta=.1,
    Cal_Cps_dcut=-.1,
    Cps_Track_dphi=.1,

```

```

Cps_Track_dz=5.,
Cps_Track_dcut=-.1,
Cal_Track_dphi=.1,
Cal_Track_deta=.1,
Cal_Track_dcut=-.1,
CALUNP=NONE,
TRACK=NONE,
CPS=NONE,
CHI2=-100.,
EMSEED=NONE,
isolation=-1.,
ConeSize=.25,
Width_EM1=-1.,
Width_EM2=-1.,
Width_EM3=-1.,
Width_ECEM1=-1.,
Width_ECEM2=-1.,
Width_ECEM3=-1.,
tooltype="physics")|
ELE_NLV_SH L3TEle(
CAL=CAL_CLUS4_PV3_NLC_ON,
EMFR=.9,
Cal_Cps_dphi=.1,
Cal_Cps_dz=100.,
Cal_Cps_deta=.1,
Cal_Cps_dcut=-.1,
Cps_Track_dphi=.1,
Cps_Track_dz=5.,
Cps_Track_dcut=-.1,
Cal_Track_dphi=.1,
Cal_Track_deta=.1,
Cal_Track_dcut=-.1,
CALUNP=NONE,
TRACK=NONE,
CPS=NONE,
CHI2=-100.,
EMSEED=NONE,
isolation=-1.,
ConeSize=.25,
Width_EM1=2.3,
Width_EM2=1.7,
Width_EM3=1.5,
Width_ECEM1=1.4,
Width_ECEM2=1.35,
Width_ECEM3=1.4,
tooltype="physics")|

```

[comment] This begins part B of the <triglist>

This part consists of three-part structures:

```

L2trig l2name
filter TriggerName

```

script term(s)

One of these structures is emitted for each real L3 script in the trigger

[end comment]

```
L2trig 1^min_bias_nim_NCU|
filter min_bias_nim_NCU |
L3FPassFraction key=t1s1_PassFraction passFraction=1. Stream="DAQ_TEST" |
L2trig 2^zero_bias_NCU|
filter zero_bias_NCU |
L3FPassFraction key=t2s1_PassFraction passFraction=1. Stream="DAQ_TEST" |
```

[comment]

At this point the following lines would be emitted if the two L3_OR_MARKER scripts were real.

```
L2trig 3^ARNOLD_OR1|
filter ARNOLD_OR1 |
L2trig 4^ARNOLD_OR2|
filter ARNOLD_OR2 |
```

[end comment]

[comment] Here begins the structure for the first member of the L3 Or group [end comment]

```
L2trig 5^ARNOLD_OR3|
filter ARNOLD_OR3 |
L3FMarkAndPass key=t5s1_mp17000 pass_1_of_n=17000. Stream="DAQ_TEST" |
L3FEle key=t5s2_Ele refset=ELE_NLV_SHT number=1 MinEt=22. MinEta=0. MaxEta=3.6
MinEoverp=-99. MaxEoverp=99. Stream="DAQ_TEST" |
L3FEle key=t5s3_Ele refset=IsoEle_SHT number=1 MinEt=22. MinEta=0. MaxEta=3.6
MinEoverp=-99. MaxEoverp=99. Stream="DAQ_TEST" |
```

[comment] Notice that the L2trig referenced in the next structure is the same as the one just referenced. This is because ARNOLD_OR3 and ARNOLD_OR4 refer to L3 scripts part of a single L3 OR group.

[end comment]

```
L2trig 5^ARNOLD_OR3|
filter ARNOLD_OR4 |
L3FEle key=t6s1_Ele refset=ELE_NLV number=2 MinEt=15. MinEta=0. MaxEta=3.6
MinEoverp=-99. MaxEoverp=99. Stream="DAQ_TEST" |
L3FEle key=t6s2_Ele refset=ELE_NLV_SH number=1 MinEt=15. MinEta=0. MaxEta=3.6
MinEoverp=-99. MaxEoverp=99. Stream="DAQ_TEST" |
L3FEle key=t6s3_Ele refset=ELE_NLV number=1 MinEt=20. MinEta=0. MaxEta=3.6
MinEoverp=-99. MaxEoverp=99. Stream="DAQ_TEST" |
L2trig 7^Cal_unsuppressed|
filter Cal_unsuppressed |
L3FMarkAndPass key=t7s1_mp1 pass_1_of_n=1. Stream="DAQ_TEST" |
L3FPassFraction key=t7s2_PassFraction passFraction=0. Stream="DAQ_TEST" |
```

||</triglist>

```
<l3update name='magnet'>
  <l3parm name='sol_pol' value='$l3.sol_pol'!/>
</l3update>
</trigdef>
```

<!-->&smt_monitoring;<!--An online monitoring process for SMT may be engaged in the CRATER GUI via file smt_monitoring.xml-->

```
<stream substreams='1' name='daq_test' family='daq_test'!/>
<stream substreams='1' name='monitor' family='monitor'!/>
</configuration>
```