# DØ Run II Trigger Definition Language

*G. Blazey, A. Boehnlein, M. Fortner, J. Linnemann, R. Moore, C. Silva, S. Snyder et al.*

**Abstract**

*This document sets out the basics of the language which will be used to program the trigger in run II. It is meant as a working document which means it will evolve over time as more details are resolved and parts are implemented.*

## 1. Introduction

The trigger definition language for Run II will be used to configure the Level 1, 2 and 3 triggers for data taking as well as diagnostic and test runs. The program will be stored in a single text file which will then go through several levels of processing before being downloaded to the hardware.

## 2. Overall System

The scheme of the configuration system is shown in Figure 1. The text file which describes the configuration will be generated by a Graphical User Interface (GUI). This interface will obtain information from a database about the current setup of the hardware to ensure the user generates a compatible file. It will also be possible to generate this text file by hand from an editor.

The next stage is to parse and check the language which is done by the *TrigParse* program. This program performs a rigourous check of the syntax and ensures that the file is consistent with the current hardware and software configuration which is stored in the database.

The final phase of combined processing is to pass the script to COOR. Here the resources which the script requests are checked against those in use and the state of the resources in are then updated in COOR.

After processing by COOR the script is split into three parts, one for each of the trigger levels. These subscripts are then passed to the individual machines which are responsible for configuring the hardware.

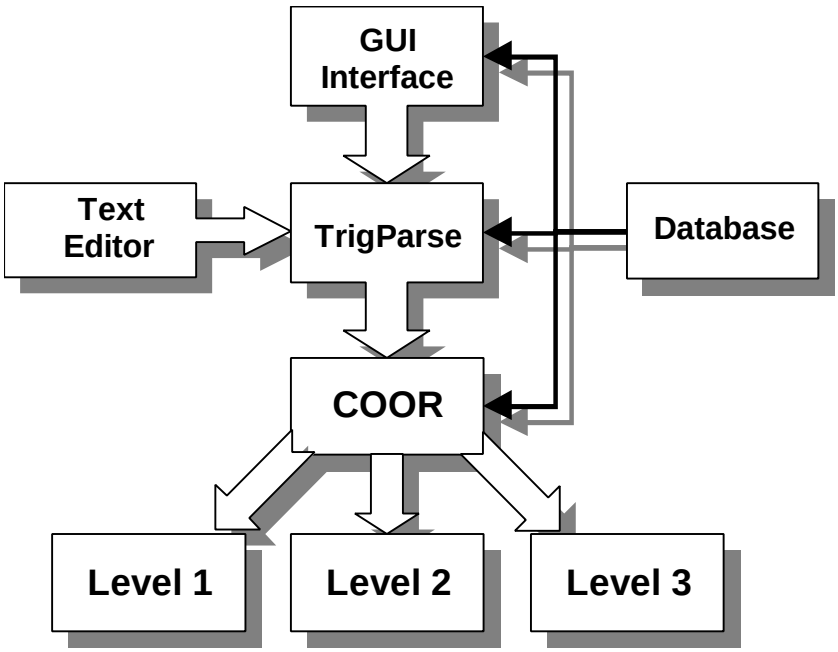


**Figure 1:** *Diagram showing how the configuration file will be parsed. The function of the various components is described in the text.*

## Data Types

There are four different data types supported in the configuration file. These are:

i) **Floating Point** : These are recognized as a string containing numerals and a decimal point. For example '123.' and '123.45' would be accepted as floating point numbers but '123' or '123.a' would not. All floating point numbers will be treated as 32 bits.

ii) **Integer** : These are defined as a string only containing numerals. For example '123' and '12345' are valid integers whereas '123.' and '1aB' are not. Numbers starting with '0x' will be treated as hexadecimal, allowing acceptance of 'a-f' as integers (case insensitive) e.g. "0x1aa" or "0x1F0". All integers will be treated as 32 bits.

iii) **Boolean** : These fields will only accept 0 or 'false' and 1 or 'true' and represent a simple one bit value. When using text to initialize the variable case is irrelevant and no quotes must be used.

iv) **String** : String values must always be enclosed by double quotes. Any alphanumeric character is acceptable except for a space. However to avoid confusion with other data types it is recommended that you avoid starting strings with numerals.

v) **Labels and References** : These refer to another object by name, similar to pointers in C. They are represented as text strings which are not enclosed by quotes. This variable type cannot be given a range of allowed values.

vi) **Arrays** : Arrays can be of any data type and come in two flavours: fixed length and variable length. Fixed length arrays are specified by adding the length enclosed in square brackets after the name e.g. 'ARRAY[10]'. Variable length arrays use the same syntax but now the square brackets enclose the name of another variable in the structure being defined e.g. 'ARRAY[SIZE]'.

The length variable must be given a lower limit of zero and have a specified upper limit. Failing to do this will generate a syntax error.

| Type | Example Declaration | Example Initialization |
|---|---|---|
| floating point | float TEST | TEST=3.14159 |
| integer | int TEST | TEST=3 |
| boolean | bool TEST | TEST=true |
| string | string TEST | TEST="test" |
| label/reference | ref TEST | TEST=object |
| array | int TEST[10] | TEST=[1,2,3,4…] |
| | | TEST[0:4]=7 |
| 2d array | int TEST[-4:4][5] | TEST[-3][4]=7 |
| | | TEST[-2:4][]=5 |

**Table 1:** *Summary of data types, how to declare them and how to initialize them.*

## General Parsing Rules

To allow the configuration file to be more legible there are several general rules followed when parsing the file:

i) Any number of blank lines are allowed at any point in the file. These will be automatically removed by the parser
ii) Comments can be denoted using the hash character ('#'). Everything after the '#' up until the end of the line will be ignored by the parser.
iii) All keywords and names are case insensitive. The use of cases shown in this document is a suggestion to improve legibility but is not enforced. This should not be used to create names such as "L2eLeCtRoN"!
iv) Any number of spaces or tabs may be used to separate fields on a single line. This is to allow alignment of fields, again to improve legibility.
v) Long lines can be split using the standard UNIX line split approach. This involves placing a backslash ('\') at the end of the first split line. For example:

```
This is an example \
of a split line.
```

will be treated as a single line of text.

## Terms Used

i) **Structure** : a structure consists of a list of fields each, possibly, with a default value and an value allowed range. These are then used to create

specific instances of objects in the second part of the file. To use C/C++ terminology a structure is a 'class' or 'struct'.

ii) **Object** : an object is a collection of parameters. Each object type has a structure which lists all its parameters and, possibly, their default, minimum and maximum values. To use C/C++ terminology: an object can be considered as an instance of a class or struct.

iii) **Tool** : these are the objects responsible for creating the physics objects which are then used to pass or fail the event. The functionality of the tools differs slightly from level 2 to 3 but in essence these tools use input data, in conjunction with the tool's parameters, to create physics objects.

iv) **Script** : this is a collection of tools which are run when a particular trigger bit is set and decide whether or not to pass the event.

# 3. Anatomy of the Configuration File

The trigger configuration file is split into several sections as shown in Figure 2. The initial sections define and then configure the objects used by the actual trigger scripts, which is contained in the final section. Each section is described in detail below.

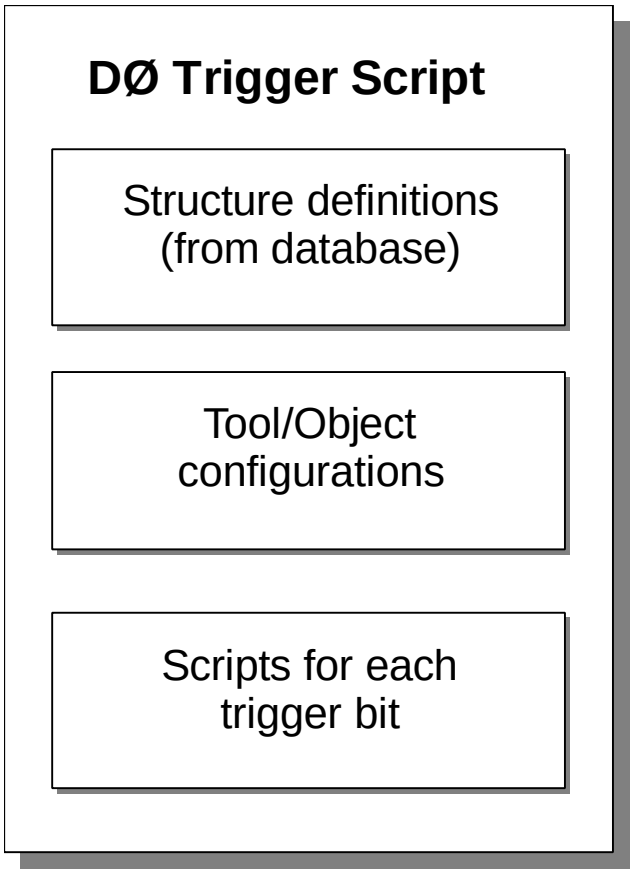


**Figure 2:** *Internal layout of the trigger configuration script. The first section is optional and can be dropped once the online database is fully functional.*

# 4. Structure Definitions

The first part of the configuration file deals with defining the structure of the objects used by the various trigger levels by listing their parameters and ascribing them unique identifiers. This information is added to the head of the file to make it easier to parse, although it is envisaged that the primary source for this data will be the online database when that is added to the system.

There are three sections of definitions; one for each of the three trigger levels. Each starts with the lines:

```
L#structures
```

respectively, where the hash (#) is replaced by the trigger level, e.g. `L2structures`. The list of structure definitions is ended automatically by the next section declaration.

## 4.1 Level 1 Definitions

The level 1 definitions section includes a list of event qualifiers. These are a collection of 16 bits which are set according to which trigger bits fired. The qualifier bits are used to change the behaviour of the level 2 pre-processors. Each qualifier is assigned a name using the following syntax:

```
define L1Qualifiers
  bit  0  Qualifier0_Name
  bit  1  Qualifier1_Name
  ...
endDefine
```

where "BIT x" denotes that you are referring to the $x^{th}$ bit of the qualifiers, starting to count at 0 and the following string is the name assigned to that bit.

### 4.1.1 Level 1 Calorimeter

The L1 calorimeter has several structures to define here. First are the reference sets, or refsets. These are an array of thresholds for the

## 4.2 Level 2 Definitions

The level 2 structure definitions consist of the tool types and pre-processors as well as the generic argument structure used by all L2 tools and the configuration of the level 2 global executable. Each definition starts and ends with the lines:

`define L2`*`ToolType`* and `endDefine`

or

`define L2PP`*`PreprocType`* and `endDefine`

Depending on whether the object structure being defined is for a pre-processor of for a tool. The text after the "define" is the name of the structure being defined. For all level 2 tools this must start with 'L2' and for all level 2 pre-processors it must begin 'L2PP'. **For this reason level 2 tool types must not start with 'PP'!**

All level 2 objects must have a version number and, optionally, qualifiers which will be set for all events for which the object is used. The version number and the name of the qualifier is specified in the definition using the following syntax

```
version    VERSION_NUMBER
L1qualifier     L2QUALIFIER [,L2QUALIFIER2,…]
```

where VERSION_NUMBER is the code version of the pre-processor and QUALIFIER should be replaced with the name of the relevant qualifier bit. All level 2 qualifiers should start with 'L2'. Failure to specify this will lead to a syntax error from the parser.

**Level 2 Tools Only**

Level 2 tools must contain is a list of all the level 2 pre-processors which might be required by this tool, irrespective of its parameters. This is specified using the syntax:

```
preprocessors  L2pp1,L2pp2
```

where *pp1* and *pp2* are the names of one of the level 2 pre-processors e.g. L2Cal. Only pre-processors which are directly used by the tool in question should be listed here i.e. tools which call other tools should not include a list of pre-processors used by the tools they call. The pre-processors specified here will have their needed qualifiers set for all events in which this tool is used.

Contained inside each structure definition is the list of fields within the structure. Each field is given on a separate line, the format of which is:

```
type PARAMETER_NAME=default   from min to max
```

*PARAMETER_NAME* is the name of the field, e.g. 'EMFRAC' or 'DETA'. The data type of the field is given by the *type* argument. This must be one of 'int', 'bool', 'float', 'string' or 'ref' for integer, boolean, float, string or reference respectively. The default value of the field is given next. If there is no default for the given field then omit the equals sign; you may still specify a range. *Not supplying a default value will force all instances of the structure to explicitly specify this field.* Finally *min* and *max* give the range of allowed values for the field. Again if there is no minimum or maximum then omit the 'from' or 'to' keywords as required.

An example of a complete L2 tool structure is given below:

```
# Level 2 electron tool
define L2Electron
  preprocessors L2CFT,L2CAL
  float DETA=0.2    from 0. to 3.2
  float DPHI=0.2    from 0. to 3.2
  float EMFRAC=0.9  from 0. to 1.
endDefine
```

This example shows a possible L2 electron tool. Note the special field called 'INDEX'. Unlike other fields the parser will automatically generate and insert the correct value for this field in all instances of the tool. The '*' as a default value tells the parser that it should recognize this field as requiring special treatment and behave accordingly.

**Level 2 Pre-processors Only**

Every level 2 pre-processor includes a boolean L2READOUT field; if this is not included the parser will add it in automatically. The default value of this field should be zero (FALSE) as shown below:

```
bool L2READOUT=false
```

This field determines whether the pre-processor is required to write data to the level 2 global crate: 'false' means no readout, 'true' means readout required. The parser will automatically set this field based on the tools used by L2 global. However a pre-processor can be made to readout to global by manually setting this field to 'true'.

It is also possible to specify an array as a field. This is achieved using the syntax:

```
int SIZE=0     from 0 to 9
ref ARRAY[SIZE]
```

This specifies an array of references with 'SIZE' entries. As is shown the length of the array can be specified as a separate field or can be given as a constant value , e.g. ARRAY[10]. Note that the C/C++ style of array indexing is used, hence in the example above the maximum value of SIZE is 9 corresponding to an array of maximum length 10 (0→9).

### 4.2.1 L2ToolArgs

As well as special fields there are also two special structures. The first is called 'L2ToolArgs'. This structure defines the calling interface used by the script for *all* level 2 tools. An example is given below:

```
define L2ToolArgs
int   COUNT    from 1
float MINIMUM
float MAXIMUM
endDefine
```

The interpretation of the parameters is determined by the tool. For example "MINIMUM" could mean lowest $E_T$, $p_T$ or even $\phi$ value allowed. The reason for including such a structure, rather than hard coding it into the parser, is that it allows flexibility in the calling interface to level 2 tools, for example specifying an $\eta$ range, without needing to rewrite the parser.

### 4.2.2 L2Global

The second special structure is call 'Global'. This structure is very similar to the pre-processor structures and contains the configuration of the level 2 global code and must contain certain required fields.

- `version` - this sets the version number of the code and will be checked by the global software when the configuration is downloaded to it. If the global structure is changed in anyway at all the version number must also be changed.
- `neededQualifier` - this sets the global needed qualifier which is used by the level 2 pre-processors when determining whether or no to send data to global for a particular event.
- `NSCRIPT` - this is defined using the same syntax as a normal tool parameter i.e. type, default, min and max. The default and minimum should be set to zero and the maximum set to the largest number of scripts which the level 2 global code can handle, generally 128.

Following these field definitions there should be one field per level 2 tool type using the same name as that tool type. Each of these fields should have the same name as a corresponding tool, be of integer type and have a default value of zero. The minimum should also be zero and the maximum set to the largest number of tools of that type which the global code can stand for. An example of a global section definition is given below.

```
# Level 2 Global
define L2Global
  version 2    # Version number of global code
  neededQualifier L2GLOBAL_NEEDED
  int NSCRIPT=0     from 0 to 128 # number of scripts
  int NL2Electron=0 from 0 to 10  # no. electron tools
  int NL2Muon=0     from 0 to 15  # no. of muon tools
endDefine
```

## 4.3 L3 Definitions

The level 3 structure definitions consist of the tool types used in the level 3 trigger. Each definition starts and ends with the lines:

```
define L3ToolType    and    endDefine
```

The text after the "define" is the name of the tool type being defined. For all level 3 structures this must start with 'L3'. *(More stuff on L3 to add here!...)*

# 5. Object Definitions

Having now specified all the structures which will be used in the trigger configuration it is now necessary to actually fill the structures with the required values to achieve the triggers required. The middle section of the trigger configuration file therefore includes the specific object configurations for the level 2 and 3 triggers.

The tools are defined in two different sections. The level 2 objects must be placed after the command:

```
L2objects
```

Similarly the level 3 objects must be preceded by:

```
L3objects
```

To further differentiate between the level 2 and 3 objects, all object names must start with "L2_" or "L3_" for level 2 and level 3 tools respectively. The object definition syntax is shown below:

```
L2_ToolName L2ToolType(FIELD1=value1, FIELD2=value2)
```

In this case the line defines a level 2 tool, called "L2_ToolName", which is of type "L2ToolType". The text inside the brackets says that the default value (if any) of *FIELD1* should be replaced by *value1* and similarly for *FIELD2*. The remaining fields for the given tool type will be filled in using the default values. Any fields for this tool type which do not have a default value must be specified here or an error will be generated. The fields can be split over multiple lines but must always be enclosed inside brackets and separated from each other by commas. Examples of a couple of level 2 electron tools are given below:

```
L2_ETight L2Electron(DETA=0.1, DPHI=0.1)
L2_ELoose L2Electron(EMFRAC=0.8)
```

## Level 2 Pre-processors Only

The syntax for specifying L2 pre-processor configurations slightly different than that of other objects. Firstly the line starts with the label identifying the alpha to which the configuration data is to be sent followed by the name of the event

qualifier for which it is relevant. If the event qualifier is omitted then the configuration will be taken as the default for all events. For example the level 2 calorimeter pre-processor could be configured by the lines:

```
CalWorker L2Cal(CLUSTERSIZE=4)
CalWorker Small_Jet L2Cal(CLUSTERSIZE=3)
```

where "Small_Jet" is the name of a qualifier. Note that overriding a default parameter for two of more different qualifiers has *undefined results* i.e. if both qualifiers are set it is not specified which will have precedence.

# 6. Trigger Scripts

There is one trigger script format which sets up each of the 128 Level 1 trigger bits for all three trigger levels. Each trigger starts with the line:

```
trigger name
```

where *name* is a string enclosed in double quotes which can be used to refer to this trigger in the logger. The trigger definition is concluded with the line:

```
endTrigger
```

after which another trigger can be defined. Failure to end the definition like this will result in a syntax error.

Although there are 128 level 1 trigger bits there will generally be fewer trigger scripts. This is because the level 2 will only run one script per level 1 trigger bit and so to get two level 2 scripts to run on a single level 1 condition it must set two trigger bits. Hence although multiple level 2 scripts are allowed for each trigger script in the syntax the total number of level 2 scripts must not exceed 128 (or whatever value is defined in the L2Global structure definition).

## 6.1 L1 Configuration

The L1 section must be placed directly after the start of the block. The section is started and ended with the lines:

`L1config` and `endL1`

respectively. *(Details of the L1 configuration should be inserted here)*

## 6.2 L2 Configuration

The level 2 trigger script contains three types of objects: event qualifiers, pre-defined tools from the object definition section and high level tools which are created on demand. The high level tools are tools which use the results of other tools as their inputs, for example a mass tool. The reason that these tools are generated on demand is to reduce the complexity of script writing. The

disadvantage is that the results of these tools cannot be reused by other scripts in the same event, because each script will have its own tool. However it is unlikely that there will be two identical high level tools in different trigger scripts so the impact of this on performance should be minimal.

The L2 configuration data starts with the line:

```
L2script name
```

where *name* is the name of the script. Following this there are the low level tool call definitions. These take the form:

```
label L2_ToolName(COUNT=num, MIN=low, MAX=high)
```

The first field, 'label', is used to reference this call to a tool. The actual tool called is specified by 'L2_ToolName'. Following the tool name there are a list of arguments which are passed to the tool. These are used to determine whether the tool will return a pass or fail. The arguments are the same as those defined and described in the L2ToolArgs structure given in the L2structures section of the file.

The high level, or second order, tools have a very similar calling syntax:

```
label L2ToolType(TOOL=[label1,label2]) \
                (COUNT=num, MIN=low, MAX=high)
```

In this case we don't have a tool name so now L2ToolType(… ) refers to the actual tool type and provides parameters to create it with. In this case the parameters show that the array named 'TOOL' is to be filled with the references to the tool calls 'label1' and 'label2'. Following the specification of the high level tool there is the usual list of arguments for any level 2 tool. These are identical to the arguments specified for low level (first order) tools.

An example of a complete L2 trigger definition block is included below for the decay $Z \rightarrow e^+e^-$:

```
L2script Z->ee
Ze1 L2_ETight(COUNT=1, MIN=25.)
Ze2 L2_ETight(COUNT=2, MIN=10.)
Zee L2Mass(TOOL=[Ze1,Ze2])(COUNT=1, MIN=80., MAX=100.)
endL2
```

Note that the tool "L2_ETight" must be defined in the L2 object definition section. As is shown the end of the L2 configuration is denoted by "endL2".

As discussed above there can be multiple level 2 scripts included in a single trigger script. However in reality the parser will separate these out and generate a single trigger bit configuration for each L2script defined.

## 6.3 L3 Configuration

The final section of the trigger script describes the L3 configuration. Unlike the L1 and L2 the flexibility of the L3 trigger design truly allows multiple scripts to be associated with a single trigger bit. Accordingly there can be multiple L3 filters included in a single trigger bit script almost without limitation. The level 3 filter will be associated with the level 2 script directly preceding it .

Each of level 3 filter section begins with:

```
L3filter  name
```

where name is the name of the particular filter. Following this there are a list of tools, much the same as the L2 definition. *(More detail on L3 syntax needed here….)*

The end of each of the L3 filter blocks is denoted by the line:

```
endL3
```

# Appendix A:
## Level 2 Event Qualifiers

The level 2 event qualifiers are designed to control the readout of level 2 crates on an event by event basis while ensuring that the level 2 global MBT receives data from all input sources for every event which requires a level 2 decision. This synchronization is required because the Magic Bus Transceiver card (MBT) cannot switch input channel configurations on an event by event basis.

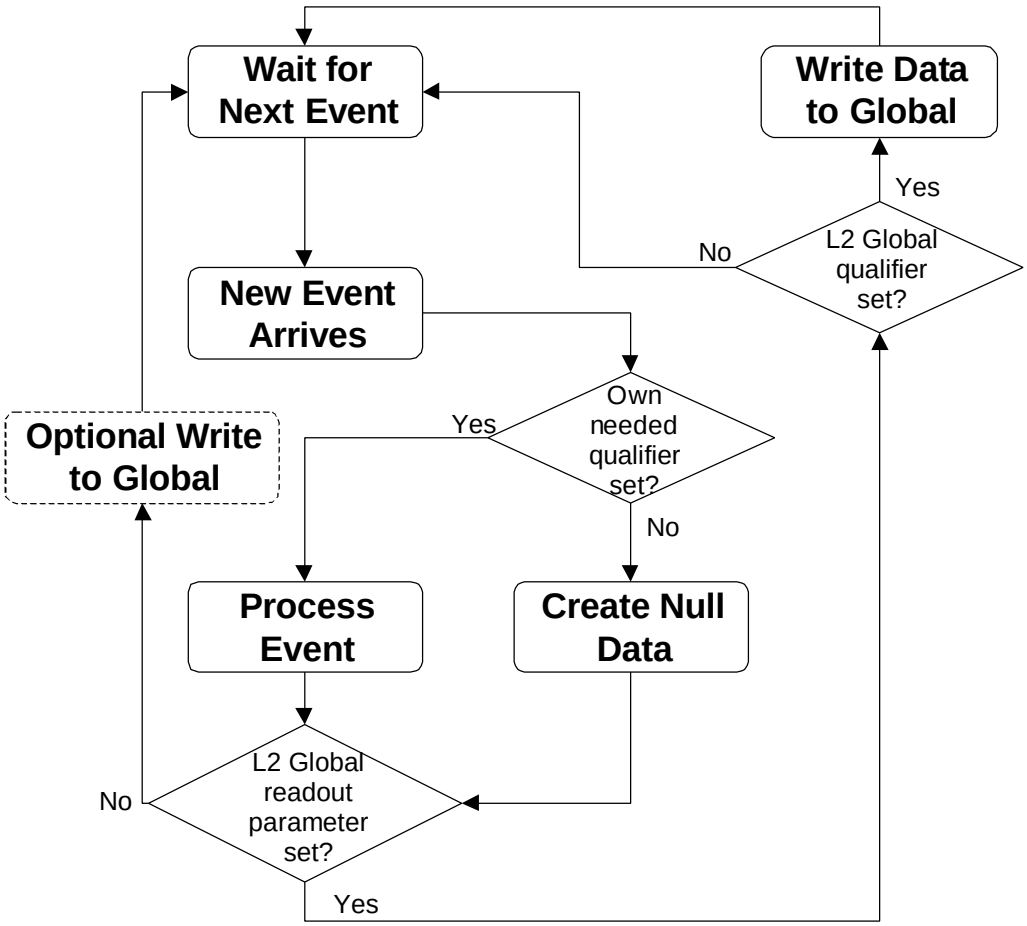


**Figure 3:** *Flowchart showing the output decision making process for level 2 pre-processor crates.*

There is one qualifier for each level 2 crate i.e. one for each pre-processor and one for global. The pre-processor qualifiers denote whether the crate in question needs to process data for the associated event. If the qualifier is not set the crate does not need to spend time processing the event but may still needs to write empty headers to the level 2 global. The global crate qualifier is set for all events where the global crate is readout.
The decision making process to decide whether to write to the level 2 global crate is shown in Figure 3. The result of the decision depends on three different quantities:

i) Level 2 readout parameter
ii) Pre-processor needed qualifier
iii) Level 2 global qualifier

The last two quantities are determined on an event by event basis whereas the level 2 readout parameter is set in the start-of-run configuration data which each crate receives and remains fixed for all events.

First the crate decides whether it needs to process the event in question. This is determined solely by its own event qualifier. If the qualifier is set then the crate must process the event, if it is not set the crate may choose not to process the event, and just produce empty headers.

After producing the processed event data, or empty headers, the pre-processor must examine its level 2 readout parameter. If the level 2 readout parameter is not set then the pre-processor in question should probably not write any data to the level 2 global crate¶, otherwise it must write data to the global crate every time the global event qualifier is set.

Irrespective of the decision to write to L2 global, if the crate sees an L2 accept for an event it *must* write either empty headers or data to the level 3.

**Requirement on COOR**

Unfortunately the restrictions which the L2 Global MBT places on the readout of pre-processor crates cannot be handled entirely within the bounds of the trigger definition language. The language only controls the trigger configuration and does not say anything about the crate readout. For this reason the COOR program must ensure that all pre-processor crates which have their level 2 readout parameter set must be readout for every event in which the level 2 global qualifier is set (and hence the L2 global crate is readout). This is required because if a crate is not in the readout list for a given event it is not notified of the event and hence cannot act on any relevant qualifiers. If a pre-processor, which had its level 2 readout parameter set, were not in the readout list for such an event it would not write any data to the level 2 global. This would result in a buffer synchronization error in the global MBT and the global crate would request an SCL_INITIALIZE for every such event!

¶ in this case the global's MBT will be configured to ignore that input channel so any data written will be safely ignored. However it may still be useful to write the data in any case for performance testing without global present

# Appendix B:

## An Example Trigger File

Given below is an example trigger file. The file is not meant to show the correct parameters and fields for each object but instead is meant to show the overall structure of the file and to act as a template for real configuration files.

```
# Example Trigger Definition File
# -------------------------------
#
#**********************************
#*     Structure Definitions      *
#**********************************

# Include some other file
include "somefile.tdl"

# ----------Level 1 Structures----------
L1structures

# L1 Qualifiers
define L1QUALIFIERS
  bit 0  L2CAL_NEEDED
  bit 1  L2CFT_NEEDED
  bit 2  L2MUON_NEEDED
  bit 3  L2GLOBAL_NEEDED
  bit 10 L2CAL_LORES
endDefine

# L1 Calorimeter total energy refset
define L1TT_TotalRefset
  float TTTOT[-20:20][32]=0.  from 0. to 1000.
endDefine

# L1 Calorimeter EM energy refset
define L1TT_EMRefset
  float TTEM[-20:20][32]=0.   from 0. to 1000.
  float TTHAD[-20:20][32]=0.  from 0. to 1000.
endDefine

# L1 Calorimeter large tile refset
define L1LT_TotalRefset
  float LTTOT[-20:20][32]=0.  from 0. to 1000.
endDefine

# Generic L! calorimeter cut tool
```

```
define L1CalET
  ref REFSET
  int COUNT=1  from 0 to 1280
endDefine

# L1 calorimeter missing ET tool
define L1MissingET
  float MINMET=0.   from 0. to 1000.
endDefine

# L1 calorimeter global EM energy tool
define L1GlobalEM
  float MINET=0.    from 0. to 1000.
endDefine

# L1 calorimeter global Hadronic energy tool
define L1GlobalHAD
  float MINET=0.    from 0. to 1000.
endDefine

# L1 calorimeter global Total energy tool
define L1GlobalTotal
  float MINET=0.    from 0. to 1000.
endDefine

# L1 Muon tool
define L1Muon
  int    COUNT=1    allowed 1,2,3
  int    MINP=2     allowed 2,4,7,11
  int    HIMINP=4   allowed 2,4,7,11
  string ETARNG="x" allowed "r1","r2","x"
  bool   TRKTRIG=true
  bool   SCINT=false
  bool   WIRECH=false
endDefine

# ----------Level 2 Structures----------
L2structures

# Level 2 calorimeter pre-processor
define L2PPCal
  version 1
  L1qualifier L2CAL_NEEDED
  int MAXCLUSTER=10 from 1 to 100
  int CLUSTERSIZE=2 from 1 to 5
endDefine
```

```
# Level 2 CFT pre-processor
define L2PPCFT
  version 2
  L1qualifier L2CFT_NEEDED
  int MAXTRACKS=10  from 1 to 100
endDefine

# Level 2 Muon pre-processor
define L2PPMuon
  version 2
  L1qualifier L2MUON_NEEDED
  int MAXMUONS=10   from 1 to 100
endDefine

# Level 2 Global (special structure)
define L2Global
  version 2
  L1qualifier L2GLOBAL_NEEDED
  int NSCRIPT=0     from 0 to 128
  int L2Electron=0  from 0 to 20
  int L2Muon=0      from 0 to 15
  int L2Mass=0      from 0 to 10
endDefine

# Level 2 Electron tool
define L2Electron
  version 3
  L1qualifier L2CAL_LOWRES
  preprocessors L2PPCal,L2PPCFT
  float DETA=0.2    from 0. to 3.2
  float DPHI=0.2    from 0. to 3.2
  float EMFRAC=0.9  from 0. to 1.
endDefine

# Level 2 Muon tool
define L2Muon
  version 2
  preprocessors L2Muon,L2CFT
  float DETA=0.25   from 0. to 3.2
  float DPHI=0.25   from 0. to 3.2
endDefine

# Level 2 Mass tool
define L2Mass
  version 1
  preprocessors  # None required directly by this tool
  int NTOOL=0       from 0 to 4
```

```
  ref TOOL[NTOOL]
endDefine

# Level 2 tool arguments (special structure)
define L2ToolArgs
  int   COUNT=1     from 0
  float MINIMUM
  float MAXIMUM
endDefine


# ----------Level 3 Structures----------
L3structures

# Level 3 Electron tool
define L3Electron
  float DETA=0.05        from 0. to 3.2
  float DPHI=0.05        from 0. to 3.2
  float EMFRAC=0.95      from 0. to 1.
  float ISOLATION=0.5    from 0. to 3.2
endDefine

#**********************************
#*       Object Definitions       *
#**********************************

# -----------Level 1 Objects-----------
L1objects

# L1 Calorimeter refsets
L1LowJet L1TT_TotalRefset(TTTOT[-20:-1][]=5., \
                          TTTOT[0:19][]=8.)
L1HighEM L1TT_EMRefset(TTEM[][]=20.,TTHAD[][]=1000.)
L1CleanEM l1TT_EMRefset(TTEM[][]=10.,TTHAD[][]=5.)

# L1 calorimeter frequently used cuts
L1Clean2e L1CalET(REFSET=L1CleanEM,COUNT=2)

# L1 Muon tools
L1SingleMu L1Muon(COUNT=1,MINP=2,WIRECH=true)
L1DoubleMu L1Muon(COUNT=2,MINP=2,ETARNG="r1")

# -----------Level 2 Objects-----------
L2objects

# Pre-processors
CalWorker L2Cal(L2READOUT=1, MAXCLUSTER=15, CLUSTERSIZE=3)
```

```
CalWorker L2CAL_LOWRES L2Cal(CLUSTERSIZE=4)
CFTWorker L2CFT(L2READOUT=1)
ForwardMuonWorker L2Muon(MAXMUON=20)
CentralMuonWorker L2Muon(MAXMUON=15)

# Tools
L2_ETight L2Electron(DETA=0.1, DPHI=0.1)
L2_ELoose L2Electron(EMFRAC=0.8)


# -----------Level 3 Objects-----------
L3objects

L3_ETight L3Electron(EMFRAC=0.99, ISOLATION=0.2)
L3_Eloose L3Electron(DETA=0.1, DPHI=0.1)



#**********************************
#*        Trigger Scripts         *
#**********************************

# Trigger on Z going to electron plus something
trigger Z->eX

L1config clean2e
  L1Clean2e
  !L1MissingET(MINMET=10.)
  L1GlobalEM(MINET=60.)
  L1SingleMu
endL1

L2script Z->ee
  L1prescale 10
  L1ubsOneOf 10000
  Ze1 L2_ETight(COUNT=1, MIN=25.)
  Ze2 L2_ETight(COUNT=2, MIN=10.)
  Zee L2Mass(TOOL=[Ze1,Ze2]) \
            (COUNT=1, MIN=80., MAX=100.)
endL2

L3filter Zee1
# First L3 filter
endL3

L3filter Zee2
# Second L3 filter
```

```
endL3

L2script Z->emu
  L1prescale 3
  L1ubsOneOf 20000
  L1qualifier L2CAL_LOWRES
  Ze3 L2_ETight(COUNT=1, MIN=10.)
  Zmu1 L2_MuLoose(COUNT=1, MIN=10.)
  Zemu L2Mass(TOOL=[Ze3,Zmu1]) \
             (COUNT=1, MIN=80., MAX=100.)
endL2

L3filter Zemu1
# Third L3 filter
endL3

endTrigger


# Trigger on SUSY event
trigger SUSY

L1config
  L1MissingET(MINMET=20.)
endL1

# Trigger script goes here
endTrigger
```