

COOR

scott snyder

October 8, 2002

For on100-58-00

Contents

1	Notational Conventions	7
2	Overview	8
3	Summary of Requirements	11
4	Running COOR	13
4.1	Online Setup	13
4.2	Starting and Stopping COOR	13
4.3	Controlling COOR	13
4.4	Files	13
4.5	Simulation Mode	15
5	Running TAKER	18
6	Detector Model	19
7	Begin/End Run File Format	25
8	Protocols	30
8.1	COOR-Client Communication	30
8.2	COOR-Downloader Communication	39
8.2.1	Common Protocol	39
8.2.2	COMICS	43

8.2.3	Level 1	44
8.2.4	Level 2	47
8.2.5	Level 3	47
8.2.6	SDAQ	49
8.2.7	Data Logging	51
8.2.8	Calibration manager	52
8.3	COOR-Logbook Communication	54
8.4	Run Transition SES Messages	54
8.5	Name Server Messages	55
9	Run Transitions	57
10	Writing Trigger Configurations	58
10.1	Finding Configurations	58
10.1.1	Python Script	58
10.1.2	XML	58
10.1.3	Dumped States	59
10.1.4	Custom XML	59
10.2	XML	59
10.3	Example	60
10.4	Run modes	67
10.5	Generic Element Reference	68
10.5.1	Element <code>calibration</code>	68
10.5.2	Element <code>client</code>	69
10.5.3	Element <code>clients</code>	70
10.5.4	Element <code>configuration</code>	71
10.5.5	Element <code>crate_list</code>	73
10.5.6	Element <code>crateref</code>	74
10.5.7	Element <code>download</code>	74
10.5.8	Element <code>expogroup</code>	75
10.5.9	Element <code>l1em_refset</code>	76
10.5.10	Element <code>l1hadveto_refset</code>	77
10.5.11	Element <code>l1jet_refset</code>	78
10.5.12	Element <code>l1lt_refset</code>	79
10.5.13	Element <code>l1refsets</code>	80
10.5.14	Element <code>l1term_list</code>	80
10.5.15	Element <code>l1trigger</code>	81
10.5.16	Element <code>l2filter</code>	83

10.5.17	Element l2global	83
10.5.18	Element l2script	84
10.5.19	Element l2trigger	84
10.5.20	Element l3trigger	85
10.5.21	Element level2	86
10.5.22	Element sdaq	86
10.5.23	Element sdaq_l1trigger	87
10.5.24	Element stream	89
10.5.25	Element trigdef	90
10.5.26	Element triglist	91
10.6	Device Type Reference	91
10.6.1	Element Calpulser	92
10.6.2	Element Cal_ADC_Crate	93
10.6.3	Element Cal_TC_Crate	94
10.6.4	Element CFT_Crate	95
10.6.5	Element L1_Crate	96
10.6.6	Element L2_Crate	97
10.6.7	Element Muo_Crate	98
10.6.8	Element Null_Device	99
10.6.9	Element SMT_Crate	100
10.6.10	Element STT_Crate	101
10.6.11	Element Trig_Crate	102
10.7	Level 1 Trigger Term Reference	103
10.7.1	Element l1ctt	103
10.7.2	Element l1emcount	104
10.7.3	Element l1emcountr	104
10.7.4	Element l1emquad	105
10.7.5	Element l1jetcount	106
10.7.6	Element l1jetcountr	107
10.7.7	Element l1jetquad	108
10.7.8	Element l1ltcount	109
10.7.9	Element l1emetsum	109
10.7.10	Element l1fpd	110
10.7.11	Element l1fps	111
10.7.12	Element l1hdetsum	111
10.7.13	Element l1lum	112
10.7.14	Element l1misspt	113
10.7.15	Element l1muo	113

10.7.16	Element 11specterm	114
10.7.17	Element 11totetsum	114
10.8	Level 2 Preprocessor Reference	115
10.8.1	Element 12calem	115
10.8.2	Element 12caljet	116
10.8.3	Element 12calmet	117
10.8.4	Element 12cps	117
10.8.5	Element 12ctt	118
10.8.6	Element 12fps	118
10.8.7	Element 12muc	119
10.8.8	Element 12muf	119
10.9	Level 2 Tool Reference	120
10.9.1	Element 12commissiontool	120
10.9.2	Element 12emfilter	120
10.9.3	Element 12emtool	121
10.9.4	Element 12etafilter	121
10.9.5	Element 12etaphisepfilter	122
10.9.6	Element 12etasepfilter	123
10.9.7	Element 12failallfilter	124
10.9.8	Element 12htfilter	124
10.9.9	Element 12invmassfilter	125
10.9.10	Element 12jetfilter	125
10.9.11	Element 12jetfilter	126
10.9.12	Element 12metfilter	127
10.9.13	Element 12mettool	127
10.9.14	Element 12muonfilter	128
10.9.15	Element 12muontool	128
10.9.16	Element 12phifilter	129
10.9.17	Element 12phisepfilter	130
10.9.18	Element 12randompassfilter	130
10.9.19	Element 12trackfilter	131
10.9.20	Element 12tracktool	131
10.9.21	Element 12transmassfilter	132
10.10	Prescale Sets	133

11	Defining COOR Resources	134
11.1	Resource Definitions	134
11.2	Resource Element Reference	141

11.2.1	Element attribute	141
11.2.2	Element calpulser	142
11.2.3	Element crate	143
11.2.4	Element crates	144
11.2.5	Element device	144
11.2.6	Element devices	145
11.2.7	Element devtype	145
11.2.8	Element l1ct_emcount	146
11.2.9	Element l1ct_emcountr	147
11.2.10	Element l1ct_emquad	148
11.2.11	Element l1ct_jetcount	148
11.2.12	Element l1ct_jetcountr	149
11.2.13	Element l1ct_jetquad	150
11.2.14	Element l1ct_ltcount	151
11.2.15	Element l1ct_refset	151
11.2.16	Element l1ct_thresh	152
11.2.17	Element l1qual	153
11.2.18	Element l2global	154
11.2.19	Element l2input	155
11.2.20	Element l2mbt	155
11.2.21	Element l2parm	156
11.2.22	Element l2pp	157
11.2.23	Element l2ppcrate	158
11.2.24	Element l2tool	158
11.2.25	Element level1	159
11.2.26	Element level2	160
11.2.27	Element resources	160
11.2.28	Element subdevice	161
11.2.29	Element term	162
11.2.30	Element tieto	162
11.2.31	Element trigmgr	163

12	TAKER	165
12.1	Run Transition Dialogs	165
12.2	TAKER plugins	166
12.2.1	Finding plugins	166
12.2.2	Activating plugins	167
12.2.3	Taker proxy methods	167

12.2.4	Plugin interactions	168
13	The Name/Value Service	169
13.1	Introduction	169
13.2	The Python Name Service API	169
13.3	The C++ Name Service API	171
13.4	The Name Service Editor	172
A	Document Type Definitions	174
A.1	Trigger Configuration	174
A.1.1	trigger_config.dtd	174
A.1.2	11terms.dtd	181
A.2	Resources	184
A.2.1	resources.dtd	184
B	Run Mode Examples	190
B.1	External	190
B.2	FW-Only	190
B.3	PDAQ	191
B.4	Parasitic-SDAQ	192
B.5	FW-SDAQ	193
B.6	PDAQ-SDAQ	194
B.7	SDAQ	195
C	State Diagrams	197

1 Notational Conventions

Text in *this font* refers to features of COOR which are planned, but not yet implemented.

2 Overview

The COOR program is responsible for coordinating changes in state of the software and hardware components comprising the data acquisition system. Clients wishing to use pieces of the system must send requests to the COOR, which will ensure that the request does not conflict with those of other clients. COOR will then communicate with the rest of the system to put it in the state requested by the client.

Requests to begin and end runs are also considered state changes which should go through the COOR. In response to such a request, it will step the other components of the system through the proper sequence of actions. These requests may come not only directly from clients, but may also be generated by the data logging and alarm systems to automatically stop or pause a run due to an error condition.

Figure 1 illustrates the main communication paths to and from COOR. At the top of the figure are the *clients*, which make requests of COOR. These will most commonly be instances of the *TAKER* program, which is the primary user interface for controlling data taking. However, there will also be clients to display status information about COOR, and to provide expert-level control of COOR itself. At the bottom of the figure are the *targets*, with which COOR communicates in order to effect changes in the system. At present the set of targets includes the Level 1 and 2 triggers (through the L1TCC), the Level 3 trigger (through the L3 supervisor), the EPICS system (through a translator program), and the data logging system.

COOR also talks to the alarm system (illustrated on the left), not only to report its own errors, but also to report major state changes, such as the beginnings and ends of runs. The alarm system will then make these state change notifications available to any other components of the system which want them.

Note the following:

- COOR is not in the data path. It is responsible for setting up and terminating runs, but does not directly participate in them.
- The flow of commands is largely one-way, from the clients, through COOR, and finally down to the targets. The exception is that COOR may send a notification back to a client if its run has been asynchronously stopped or paused. Note in particular that this implies that if the alarm system or one of the targets wants to change the run state — such as

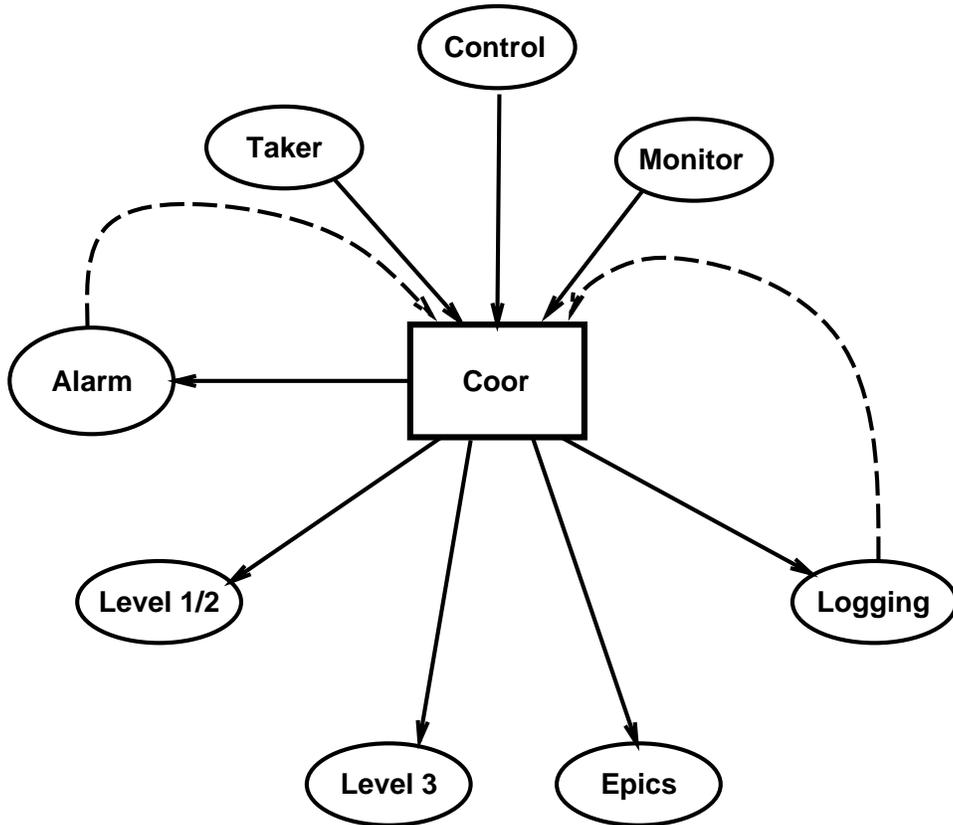


Figure 1: Primary communication paths to and from COOR.

pausing physics runs on a fatal alarm or stopping a run after a fixed number of events has been recorded — that component should make a separate client connection back to COOR to send the command. (This is illustrated by the dashed lines in Figure 1.)

- Some of the targets, such as Level 3 or the data logging system, are actually a collection of cooperating processes. For these targets, there should be a single process with which COOR communicates; this process is then responsible for distributing the commands from COOR to the rest of the subsystem. This strategy should help to insulate COOR from the internal subsystem details and make it easier to isolate the various pieces for testing and debugging.

3 Summary of Requirements

- COOR receives requests from clients to use pieces of the data acquisition system. If such a request is compatible with other concurrent users of the system, COOR communicates with the other system components to put them in the desired state.
- The subsystems which COOR should be able to configure include:
 - The Level 1, Level 2, and Level 3 triggers.
 - The data logging system.
 - The digitizing crates and VBDs.

Other components may be added as the need arises, or they may be controlled through other pathways. For example, in run I, calibration pulsers were controlled through COOR, while the high voltage, low voltage, and clock control subsystems were not.

- COOR receives requests from clients to begin and end runs. It again ensures that the request is compatible with other users and, if so, communicates with the triggers and the data logging system to start data flowing.
- COOR receives requests from clients to pause and resume ongoing runs. Pausing a run means disabling the Level 1 trigger bits used by that run. This halts the flow of data in such a way that it can be rapidly restarted.
- COOR may also receive requests to pause runs from the alarm system (due to the detection of an error which may compromise the quality of the data) or from the data logging system (if a limit was set on the number of events to be collected in a run).
- COOR should send notification of significant state changes (such as runs starting or ending) to the alarm system for distribution to any other interested parties.
- COOR should arrange to record in an appropriate database the information it has about each recorded run taken. (This may not actually be done by COOR, but instead by some other process with which it communicates.)

- COOR should be able to supply information to clients and monitoring programs concerning the current state of the detector components which it manages.
- Client programs should be provided for taking data runs and displaying the current status.

4 Running COOR

4.1 Online Setup

COOR is normally set up for running on the online cluster with the command `'setup d0online'`. You can also set up just `coor` (and the other products that it requires to run) with `'setup coor'`.

4.2 Starting and Stopping COOR

Start COOR with the command `'start_daq coor'`. This will attempt to start it up on the host given in the parameters file (`d0o1c` as of this writing). The script should refuse to start a new instance if there's already one running. You can also use `'startcoor'`. (The latter option does not require `d0online`.)

COOR may be stopped with the command `'stop_daq coor'` (from any host). Note that this requires that COOR be functioning sufficiently to process commands. If it is not, you'll need to use the `'kill'` command. You can also use `'stopcoor'`. (The latter option does not require `d0online`.)

4.3 Controlling COOR

A simple interactive client, `coortalk`, is available to control COOR. Start it with `coortalk`, exit with Control-D. Any of the commands listed in Sec. 8.1 may be sent. The responses from COOR are echoed to the terminal.

4.4 Files

Besides the files in the COOR product (which should not be altered), COOR reads files from the following locations.

- `/online/data/coor/coor.params` — The COOR parameters file. This file sets all of COOR's configurable parameters, including network addresses. This file is reread after a reinitialization; however, the values of some of the variables are used only on the initial startup.
- `/online/data/coor/resources/coor_resources.xml` — COOR's resource file. This file defines the static detector configuration — which crates are available, names for trigger terms, and so on. Its format is described in Sec. 11.

- `/online/data/coor/resources/init_devtypes.dtd` — This file is used for initialization, and should not be changed.
- `/online/data/coor/configurations` — This is the root of the tree of trigger configuration files. The format of these files is described in Sec. 10.
- `/online/data/coor/runnumber` — This file is used to keep track of the current run number. Don't change this unless you know exactly what you are doing — you can screw up offline processing if you make two runs with the same run number.
- `/online/data/coor/resources/excluded_devices.dat` — This file contains a list, one entry per line, of names of devices and crates that should be ignored (i.e., because they're not working). A hash mark (`#`) may be used for comments. This file is reread whenever any trigger configuration is loaded.
- `/online/data/coor/name_server_db.dat` — The database for the name service. This is a Python pickle file.

COOR writes the following files:

- `/online/log/coor/coor.out.timestamp` — This is where the standard output and error streams from COOR get written. This is useful mainly in cases where COOR fails to start or has crashed — this is where Python stack traces will appear.
- `/online/log/coor/yyyy/mm/coor_yyyy-mm-dd.hhmm.log` — This is COOR's log file. All activities are logged here, so it can get rather lengthy. Each day at midnight, the log file is closed and reopened with a new name. This can also be forced with the `reopen` command (see Sec. 8.1). In the event the name given above is not unique, COOR will append a suffix of the form `-n`.
- `/online/data/coor/state/runnnnnnnn.state` — For each recorded run, COOR writes out one of these state files, describing the state of the detector for that run. This is in a format that can be read back into COOR as a trigger configuration.

- `/online/data/coor/brun/brunnnnnnnn.dat` — At the beginning of each recorded run, COOR writes out one of these files, containing information to be entered into the database. The format of these files is given in Sec. 7.
- `/online/data/coor/brun/erunnnnnnnn.dat` — At the end of each recorded run, COOR writes out one of these files, containing information to be entered into the database. The format of these files is given in Sec. 7.
- `/online/data/coor/elog_spool/coorXXX.tolog` — These are temporary files used by COOR to hold logbook entries before they’re shipped to the logbook. The format is as outlined in Sec. 8.3. Other processes may also write files here with an extension of `.tolog`; they will eventually get picked up and sent to the logbook. (Be careful, though, not to give the file a `.tolog` extension while it is still being written. Best to write it first with some other name, then rename it.)
- `/online/data/coor/name_server_db.dat` — The database for the name service. This is a Python pickle file.

4.5 Simulation Mode

COOR can also be run in “simulation” mode. In this mode, COOR will not attempt to connect to any of the normal targets, will not listen for connections, and will not write files in the usual places. Instead, it will read a trigger configuration file and dump into the current directory logs of what it would have sent to the various processes, had it been running for real.

This is useful for verifying trigger configurations before using them and for generating input for the trigger simulator.

There are two variants of simulation: offline and online. They differ mainly in where COOR looks for resource information.

The offline simulation is set up as part of the DORunII setup (provided the release includes `coor`). Run it with `'coorsim trig-config'`, where *trig-config* is the trigger configuration file you want to process. COOR will look for a resource file in the library release tree, or you can specify a custom resource file using the `--resource` option to `coorsim`. Coorsim will simulate downloading the configuration and starting a run. It will write out in the current directory a collection of `.sim` files (`'11dn1.sim'`, `13dn1.sim`,

etc.) containing the commands that COOR would have sent to the various processes. It also writes a file 'coor.log', giving the verbose log of COOR's actions. The messages that COOR would have sent back to taker are sent to standard output.

The `coorsim` script takes the following options:

- `--online`: Use the online copy of COOR's resources and search the online configuration tree.
- `--resource=F`: Use F as COOR's top-level resource file. Look for other resource files in F 's directory.
- `--fetch-resources`: Try to fetch the latest version of the resources from the trigger database.

These three options are mutually exclusive.

The output written to the `.sim` file is exactly what COOR would have sent out except for the following:

- The initial command-id field is not present. In `logger.sim`, the initial 'COOR' is removed as well.
- If a message contains an embedded newline, the second and all subsequent lines of the message will have a single space prepended. This is to help in finding message boundaries in this case: the first lines of messages (and only those) will have a nonblank character in the first column.

The following `.sim` files are currently written:

- `calib.sim`
- `epics.sim`
- `level1.sim`
- `level2.sim`
- `level3.sim`
- `logger.sim`
- `sdaq.sim`

In addition, the `coorsim` scripts write out an additional file that is called `level2.sim-stripped`, which is the same as `level2.sim` except that only `l2script` messages are retained, and the `l2script` keyword itself is deleted.

An optional second parameter may be given to `coorsim`, giving a script of commands to feed to `COOR`. If it is omitted, a default script will be constructed, which loads the requested configuration and starts a run.

The online version will get set up when you do `'setup d0online'` (or `'setup coor'`). For this version, run `'coorsim_onl trig-config'`. In this case, `COOR` will read the current online resource definitions. In addition, it will look for the input `trig-config` first in the online trigger configuration tree (though it will also look in the current directory, or an absolute path can be used). Otherwise, it works like the offline version. (Note: `coorsim_onl` is a synonym for `coorsim --online`.)

5 Running `TAKER`

6 Detector Model

This section briefly summarizes the concepts COOR uses to model the current state of the system.

All objects used to model the system state are instances of a class deriving from `Object_Base.Object`. Each object has an *object name*. Objects are associated with a *registry*, deriving from `Object_Base.Object_Registry`. The `Object_Base.Object` class provides hooks for tracking all changes to the object contents. This is used for logging, monitoring, and to allow for undoing changes after discovering an error in a configuration.

Object names are conventionally divided into two parts: a *class tag*, and the name itself, written like *tag:name*. The class tag identifies the type of the object. Note that it is possible for two objects with different class tags to be implemented by the same Python class.

All objects which can be allocated derive from `Ownership.Ownable`, and those which can allocate other objects derive from `Ownership.Owner`.

All objects which represent things which must be configured derive from `Generic_Item.Item`. Such objects have a set of attributes which define their downloadable state. These are ordinary Python attributes of the object, but they follow a special naming convention. Downloadable attributes which may be changed at any time should start with ‘d_’. There are also ‘immutable’ attributes, which start with ‘i_’. These can be set when the item is first allocated, but not afterwards. These are usually used to represent structural relationships between items.

Two copies of the attributes are maintained. The actual Python attributes of the item represent COOR’s idea of the current state of the item. If the value of an attribute is `None`, that means the current state is unknown. Each item also maintains a *requested value* for each attribute. This is what has requested by the owner of the item, and is what COOR will try to make the current values reflect by means of a download.

Each item can be in one of four states:

- **UNKNOWN** — The current state of the attributes in the external system is unknown.
- **VALID** — The current state of the attributes in the external system matches what has been requested.
- **DOWNLOADING** — There is a download operation in progress on this item,

to make the state of the attributes in the external system match what has been requested.

- `DOWNLOADING_INVALID` — There is a download operation in progress, but while it was in progress, this item has been invalidated. (Probably due to a lost connection to the download target.) When the download completes, the state of this item will become `UNKNOWN`.

When which has been owned is deallocated, the list of requested attribute values is cleared and the item state is changed to `UNKNOWN`. Also, the state of an owned object can be changed to `UNKNOWN` either by an explicit invalidation request or due to a broken connection to the download target.

For most item types, the set of attributes used is defined by the Python class representing that item. For most EPICS devices, however, a more generic representation is used, implemented by the class `Devices.Device`. Each device has a *device type* associated with it (class `Devices.Devtype`); the device type defines the set of attributes which the device holds. The device types are defined during `COOR` initialization.

Some items are *generic*, in that there is a set of identical items. Such items are usually implemented using the `Generic_Item.Numbered_Item` class. In this case, the name of the item is simply a number. Each distinct set of such objects must thus have a distinct class tag.

The registry into which the objects are collected is implemented by the class `Global_State.DO_State`. This class has high-level methods for allocating the various item types.

The following table gives the presently defined class tags, the Python classes used to implement them, and a brief description of each.

<code>calibclient</code>	<code>Calib.Calib_Client</code> These objects hold information sent to the calibration manager which is specific to a given configuration. It also provides an identifier to allow associating streams uniquely with configurations.
<code>calpulser</code>	<code>Devices.Calpulser</code> A calorimeter pulser device.
<code>calibstream</code>	<code>Calib.Calib_Stream</code> These objects shadow the <code>stream</code> objects. They are used to tell the calibration manager the correspondence between stream names and numbers.

<code>client</code>	<code>Clients.Client</code> Represents an external program requesting services of COOR. This includes instances of <code>TAKER</code> , as well as monitoring and control programs. This class derives from <code>Ownership.Owner</code> , so these objects can own others.
<code>conn</code>	<code>Config.Connection_Status_Reporter_Object</code> Used to send status information about the downloader connections to monitoring programs.
<code>crate</code>	<code>Devices.Device</code> A digitization crate; a geographic section.
<code>dev</code>	<code>Devices.Device</code> A generic EPICS device.
<code>global_info</code>	<code>Global_State.Global_Info</code> Holds miscellaneous global information, such as the store number
<code>l1eg</code>	<code>Level1.L1eg</code> A Level 1 exposure group.
<code>l1global</code>	<code>Level1.L1global</code> Holds global Level 1 information. This is where the current state of the Level 2 extra crate list is maintained.
<code>l1specterm</code>	<code>Level1.L1specterm</code> A Level 1 specific (named) and/or term. These are terms for which the definitions are fixed; they do not require downloading.
<code>l1ctt</code>	<code>Level1.L1trigmgr</code>
<code>l1fps</code>	<code>Level1.L1trigmgr</code>
<code>l1fpd</code>	<code>Level1.L1trigmgr</code>
<code>l1lum</code>	<code>Level1.L1trigmgr</code>
<code>l1muo</code>	<code>Level1.L1trigmgr</code> Level 1 and/or terms from trigger manager cards.
<code>l1emetsum</code>	<code>Level1.L1ct_Thresh</code>
<code>l1hdetsum</code>	<code>Level1.L1ct_Thresh</code>
<code>l1misspt</code>	<code>Level1.L1ct_Thresh</code>
<code>l1totetsum</code>	<code>Level1.L1ct_Thresh</code> Level 1 calorimeter trigger global energy threshold and/or terms.

l1em_refset	Level1.L1ct_Refset
l1hdveto_refset	Level1.L1ct_Refset
l1lt_refset	Level1.L1ct_Refset
l1jet_refset	Level1.L1ct_Refset
	Level 1 calorimeter trigger reference sets.
l1emcount0	Level1.L1ct_Count
l1emcount1	Level1.L1ct_Count
l1emcount2	Level1.L1ct_Count
l1emcount3	Level1.L1ct_Count
l1jetcount0	Level1.L1ct_Count
l1jetcount1	Level1.L1ct_Count
l1jetcount2	Level1.L1ct_Count
l1jetcount3	Level1.L1ct_Count
	Level 1 calorimeter trigger count threshold and/or terms.
l1emcountr0s	Level1.L1ct_Static_Count
l1emcountr0c	Level1.L1ct_Static_Count
l1emcountr0n	Level1.L1ct_Static_Count
l1emcountr1s	Level1.L1ct_Static_Count
l1emcountr1c	Level1.L1ct_Static_Count
l1emcountr1n	Level1.L1ct_Static_Count
l1jetcountr0s	Level1.L1ct_Static_Count
l1jetcountr0c	Level1.L1ct_Static_Count
l1jetcountr0n	Level1.L1ct_Static_Count
	Level 1 calorimeter trigger region-specific count threshold and/or terms.
l1ltcount0	Level1.L1ct_Static_Count
l1ltcount1	Level1.L1ct_Static_Count
l1ltcount2	Level1.L1ct_Static_Count
l1ltcount3	Level1.L1ct_Static_Count
l1ltcount4	Level1.L1ct_Static_Count
l1ltcount5	Level1.L1ct_Static_Count
l1ltcount6	Level1.L1ct_Static_Count
l1ltcount7	Level1.L1ct_Static_Count
	Level 1 calorimeter trigger large tile count threshold and/or terms.
l1emquad1	Level1.L1ct_Static_Count
l1emquad2	Level1.L1ct_Static_Count

l1emquad3	Level1.L1ct_Static_Count
l1emquad4	Level1.L1ct_Static_Count
l1jetquad1	Level1.L1ct_Static_Count
l1jetquad2	Level1.L1ct_Static_Count
l1jetquad3	Level1.L1ct_Static_Count
l1jetquad4	Level1.L1ct_Static_Count
	Level 1 calorimeter trigger quadrant matching and/or terms.
l1bit	Level1.L1bit Level 1 specific trigger bit.
l2bit	Level2.L2bit Level 2 trigger bit.
l2global	Level2.L2global Holds global Level 2 information. This object holds the global parameter values specified in l2global elements. It also holds the names used for the global processor.
l2pp	Level2.L2pp Represents a Level 2 preprocessor.
l2ppcrate	Level2.L2pp_Crate Holds the information common to all preprocessors in a crate.
l2tool	Level2.L2tool Represents a Level 2 tool/filter.
l2tooltype	Level2.L2tool_Type Holds the (static) information describing a Level 2 tool type.
l3l1shad	Level3.L3l1shad Level 3 'shadow' of Level 1 information. There is one of these objects for each l1bit object. Some information associated with a Level 1 trigger bit needs to be sent to Level 3 as well (such as the crate readout list). The attributes of this object keep track of that information.
l3l2shad	Level3.L3l2shad Level 3 'shadow' of Level 2 information. Similar to l3l1shad, but for Level 2.
l3bit	Level3.L3bit

	Level 3 trigger bit.
<code>l3client</code>	<code>Level3.Level3_Client</code> The level 3 system needs to be able to separate streams and filter bits by the individual configurations, in order to be able to resolve stream names which are not globally unique. These objects provide an identifier for correlating this information.
<code>l3clattribs</code>	<code>Level3.Level3_Client_Attributes</code> Additional attributes associated with a <code>l3client</code> . These are broken out into a separate object because level 3 gets confused if it sees a <code>set_client</code> command more than once.
<code>l3stream</code>	<code>Level3.L3stream</code> These objects shadow the <code>stream</code> objects. They are used to tell level 3 the correspondence between stream names and numbers.
<code>logclient</code>	<code>Logger.Logger_Client</code> The data logging system needs to be told some information about each client which could start a run, such as whether or not it has recording turned on (see Sec. 8.2.7). These objects have the attributes containing that information.
<code>logtriggers</code>	<code>Logger.Logger_Triggers</code> We also tell the data logging system the names of all trigger bits a client has allocated. These objects have attributes containing that information.
<code>sdaqclient</code>	<code>SDAQ.SDAQ_Client</code> These objects hold information sent to the SDAQ supervisor which is specific to a given configuration. It also provides an identifier to allow associating streams uniquely with configurations.
<code>sdaqstream</code>	<code>SDAQ.SDAQ_Stream</code> These objects shadow the <code>stream</code> objects. They are used to tell SDAQ the correspondence between stream names and numbers.
<code>stream</code>	<code>Logger.Stream</code> A recording stream.

7 Begin/End Run File Format

At the beginning of each recorded run, COOR writes a “begin run” (brun) file, containing information to be entered into the database. At the end of each recorded run, COOR also writes an “end run” (erun) file. See Sec. 4.4 for the location of these files. This section lists the contents of these files.

Each line of each type of file has the following form:

keyword: value

Possible *keywords* that may appear are listed below. Note that, as mentioned below, not all keywords will always be present, and some may be present more than once.

Here are the keywords that may appear in the brun file:

- **Calibtype** — The calibration type keyword. This is present only for calibration runs.
- **Comics_Runtype** — The value of the `comics_runtype` attribute of the configuration element.
- **Configname** — The name of the trigger configuration that was loaded.
- **Configtype** — The configuration type keyword, as specified in the trigger configuration.
- **Configvers** — The version of the trigger configuration that was loaded.
- **Crate** — One of these are emitted for each crate that this configuration has allocated (including ‘crates’ with the `novbd` flag set that don’t get read out). The value for this keyword has the form

crate-id name attribs

where *crate-id* is the geographic sector number of this crate and *name* is its name. *attribs* is a list of all the settings for the downloadable attributes of this crate, each in the form *name="value"*.

- **L1bit** — One of these is emitted for each Level 1 bit that this configuration uses. The value for this keyword has the form

number prescale name

where *number* is the number of the bit, as assigned by COOR, *prescale* is the prescale value set for this run, and *name* is the name of this bit, as set in the trigger configuration. The *prescale* value may have a percent sign on the end, to indicate a percentage prescale.

- **L1bit_{eg}** — One of these is emitted for each Level 1 bit that this configuration uses. The value for this keyword has the form

bit-number eg-number

where *bit-number* is the number of the bit, as assigned by COOR, and *eg-number* is the number of its exposure group. (This keyword is separate from **L1bit** for backwards compatibility reasons.)

- **L1bit_{terms}** — One of these is emitted for each Level 1 bit that this configuration uses. The value for this keyword has the form

bit-number term-list

where *bit-number* is the number of the bit, as assigned by COOR, and *term-list* is the and/or term list requested for this bit. The form of *term-list* is the same that COOR uses when communicating with Level 1 (see Sec. 8.2.3): a space-separated list of term numbers, preceded by a dash if the term is being vetoed upon. (Note: ‘0’ and ‘-0’ are different!)

- **L1_{eg}** — One of these is emitted for each Level 1 exposure group that this configuration uses. The value for this keyword has the form

number name

where *number* is the number of the exposure group, as assigned by COOR, and *name* its name, as set in the trigger configuration.

- **L1_{eg}crates** — One of these is emitted for each Level 1 exposure group that this configuration uses. The value for this keyword has the form

number crates

where *number* is the number of the exposure group, as assigned by COOR, and *crates* is a space-separated list of crate names read out by this exposure group.

- **L1egterms** — One of these is emitted for each Level 1 exposure group that this configuration uses. The value for this keyword has the form

number term-list

where *number* is the number of the exposure group, as assigned by COOR, and *term-list* is the and/or term list requested for this exposure group. The form of *term-list* is the same that COOR uses when communicating with Level 1 (see Sec. 8.2.3): a space-separated list of term numbers, preceded by a dash if the term is being vetoed upon. (Note: ‘0’ and ‘-0’ are different!)

- **L1term** — One of these is emitted for each Level 1 and/or term that this configuration uses. The value for this keyword has the form

number descrip

where *number* is the number of this and/or term. The string *descrip* is a description of the programming of this term, as follows:

- Direct-in term (**l1specterm**): The name of the term.
- Trigger manager term: A string of the form ‘*class-name*’, where *class* is the class name of this manager, as specified in the resource file (‘l1muo’, ‘l1ctt’, etc.) and *name* is the name of the requested input term for this manager.
- Calorimeter trigger count threshold terms: A string of the form ‘*class-count-rsnames*’. Here, *class* is one of the following:

- * ‘l1emcount*N*’
- * ‘l1jetcount*N*’
- * ‘l1l1tcount*N*’
- * ‘l1emcountr*N*’
- * ‘l1jetcountr*N*’
- * ‘l1emquad*N*’
- * ‘l1jetquad*N*’

where *N* is the number of the reference set that this term uses. The *count* field is the count threshold for this term, and *rsnames* gives the names of the reference sets used for this term, either a

single name for jet or large-tile terms, or two names, the EM and hadronic veto reference sets, separated by a dash for EM terms.

- Calorimeter trigger threshold terms: A string that has the form ‘*class-value*’, where *class* is either ‘`l1misspt`’ or ‘`l1totetsum`’, and *value* is the threshold value.

- **L2bit** — One of these is emitted for each Level 2 bit that this configuration uses. The value for this keyword has the form

number l1bit name

where *number* is the number of the bit, as assigned by COOR, *l1bit* is the number of the Level 1 bit on which this Level 2 bit depends, and *name* is the name of this bit, as set in the trigger configuration.

- **L3bit** — One of these is emitted for each Level 3 bit that this configuration uses. The value for this keyword has the form

number l2bit name

where *number* is the number of the bit, as assigned by COOR, *l2bit* is the number of the Level 2 bit on which this Level 3 bit depends, and *name* is the name of this bit, as set in the trigger configuration.

- **L3type** — The name of the requested Level 3 node type. This is present only for runs using Level 3. [Note: at present, for configurations specifying more than one level 3 type (with multiple `trigdef` elements), only the first level 3 type is reported.]
- **LBN** — The luminosity block number for the start of the run. All data in this run should have a LBN greater than or equal to this value. This number will be -1 if this run does not use the primary DAQ path.
- **Physics** — Either 0 or 1, depending on whether this configuration has the physics flag set.
- **Prescname** — The name of the last prescale set that was loaded. This may be blank if no prescale set has been loaded.
- **Recording** — Either 0 or 1, depending on whether recording is enabled for this run.

- **Run** — The run number.
- **Sdaqtype** — The SDAQ type keyword. This is present only for runs using SDAQ.
- **Store** — The current store number. This is present only if there is currently a store in progress.
- **Stream** — One of this is emitted for each stream that this configuration uses. The value is the name of the stream.
- **Stream_Scheme** — The value of the `stream_scheme` attribute of the `configuration` element.
- **Time** — The time at which the run started. This will be a string like this: ‘2000 Oct 26 02:16:25 UTC’. We use UTC rather than local time in order to avoid ambiguity when switching from daylight savings time to standard time.

Here are the keywords that may appear in the `erun` file:

- **LBN** — The luminosity block number for the end of the run. All data in this run should have a LBN less than to this value. This number will be `-1` if this run does not use the primary DAQ path. It may also be `-1` if COOR was unable to successfully retrieve the LBN from Level 1 at the end of the run.
- **Run** — The run number, as above.
- **Time** — The time at which the run ended. The format is the same as in the `brun` file.

Some additional keywords may also appear, beyond those listed here. A client may send keywords to be placed in these files along with the `start` and `stop` commands. Normally, these will be encoded in the `coor.params` variables `brun_desc` and `erun_desc`.

Also, any names in the name server database (see Sec. 13) with the ‘`brun`’ property set will be written to the `brun` file, and any names with the ‘`erun`’ property set will be written to the `erun` file. Such names may be distinguished because they will always start with a period.

8 Protocols

This section summarizes the protocols used in the communication between COOR and the various processes with which it communicates.

8.1 COOR-Client Communication

All communication between COOR and the clients is by way of ITC string messages. The usual scenario is for the client to send a command to COOR, then wait for a response. Commands should not be overlapped (except for `abort`, as noted below). There are also several responses which COOR may send asynchronously; these include `TEXT`, `MESG`, `UPDATE_MODATTR`, `UPDATE_NEWOBJ`, `UPDATE_DELOBJ`, and `CMND`, and are discussed further below.

Except as noted below, COOR will respond to each command with a message starting with either `DONE`, `FAIL`, or `ABORTED`. In each case, there may be further data following the keyword, as noted for the individual commands below. `DONE` means that COOR has made the state change requested by the command, `FAIL` or `ABORTED` means that it has not. (Note that this is not necessarily the same as whether the command completed successfully or not. However, modification commands that do not involve a state change should return `FAIL` if the change did not actually take effect.) `ABORTED` is sent after a download has been aborted, either explicitly by the user sending an `abort` command, or automatically, following a timeout.

Commands which initiate a download may take considerable time to complete. To confirm that the download has started, COOR will first reply with `WAIT`. This will then be followed by one of `DONE`, `FAIL`, or `ABORTED`, once the download is complete.

Before the `DONE` (or `FAIL` or `ABORTED`) reply, COOR may send any number of `TEXT` messages to the client. These will contain additional text after the keyword which should, in most cases, be displayed to the user. If COOR knows that the message is an error, it will be prefixed with the string `*bad*`. In addition, warning messages will be prefixed with the string `*warn*`. Client programs can use this information to highlight these messages.

At any time, COOR may send a `MESG` message to the client. This will contain additional text after the keyword which should, in most cases, be displayed to the user. The distinction between `MESG` and `TEXT` is that `TEXT` messages should be output generated by a command being executed, while `MESG` messages may arrive asynchronously (from, for example, an operator's

broadcast request). Some applications may request a report from COOR, then attempt to parse the TEXT messages containing the body of the report. Having a separate MESH type avoids the need to deal with the possibility of parsing random asynchronous messages interspersed with the report text.

Also at any time, COOR may send a STAT message to the client. This is just like MESH, except that the text is intended to be displayed in a short “status” area, if there is one. For example, periodic reports of the number of events in the run will use this.

If the run state is changed by COOR, it will send a CMND message to the client in order to notify it of the change. The messages of this type presently defined are:

- **CMND free** — COOR has freed all objects this client has allocated.
- **CMND pause** — COOR has paused the client’s run.
- **CMND stop** — COOR has stopped the client’s run.

Following are all the client commands which COOR recognizes. Note that the set of these that are allowed depends on what COOR thinks the current state of the client is — see the state diagrams for details.

- **abort** — Abort a download in progress. It should be sent after the WAIT reply, but before the final reply. Note that COOR will not generate a DONE or FAIL response for this command — when the download completes, the **abort** request is also considered ended.
- **alarm text** — Feed an (informational) alarm message through to the alarm system.
- **auto_pause runlist** — Request the runs given by *runlist* to pause. *Runlist* should be a space-separated list of integers; these should either be run numbers or client port numbers. If it is omitted, all runs in progress will be processed. A run will only actually pause if its configuration has the **autopause** flag turned on. The intention is that this message is sent by the alarm server to COOR when there is a fatal alarm.
- **broadcast text** — Send *text* to all clients as a TEXT message. The message will be prefixed with ‘-->’.

- `coor_auto_pause` — Used internally to implement the `auto_pause` command. This command should not be sent by clients.
- `coor_force_free` — Used internally to implement the `force_free` command. This command should not be sent by clients.
- `coor_force_pause` — Used internally to implement the `force_pause` command. This command should not be sent by clients.
- `coor_force_stop` — Used internally to implement the `force_stop` command. This command should not be sent by clients.
- `disconnect dnllist`— Force the connections to the download targets named in *dnllist* to go away. If *dnllist* is omitted, connections to all download targets will be dropped.
- `dump pattern` — Dump COOR’s internal configuration state to the client. The argument *pattern* is a regular expression; information about all objects matching *pattern* will be included in the dump. If *pattern* is omitted, information for all objects is dumped.

The information returned from COOR will be in a message starting with the keyword `DUMP`. The rest of the message will consist of a string which, when passed through the Python reader, will produce a Python dictionary containing the dump data.

The `DUMP` message will then be immediately followed by a `DONE` message.

- `dump_update pattern` — This combines the effect of `dump` and `update` in a single command. The advantage of using this rather than two separate commands is that it eliminates the chance that a change could occur between the `dump` and the `update`.
- `exitcoor` — Exit COOR.
- `flush` — Force a flush of COOR’s log file.
- `force_free runlist` — Force the runs given by *runlist* to stop, and for the clients to release all the resources they hold. *Runlist* should be a space-separated list of integers; these should either be run numbers or client port numbers. If it is ‘all’, all runs in progress will be affected. If it is omitted, the command has no effect.

- **force_invalidate** *pattern* — Invalidate all items which match the regular expression *pattern*. (If *pattern* is omitted, all items are considered to match.) Only items which are owned by some client are invalidated. This command differs from the **invalidate** command in that it does not require that the client issuing the command own the items being invalidated.
- **force_pause** *runlist* — Force the runs given by *runlist* to pause. *Runlist* should be a space-separated list of integers; these should either be run numbers or client port numbers. If it is ‘all’, all runs in progress will be affected. If it is omitted, the command has no effect.
- **force_reinit** — Reinitialize COOR. Connections to clients are left intact, but connections to download targets will be dropped and reestablished. The parameters and resources files will be reread, but note that not all parameters can be changed during a reinitialization. Any runs in progress will be forcibly stopped, and all clients will have their owned resources forcibly released.
- **force_stop** *runlist* — Stop the runs given by *runlist*. *Runlist* should be a space-separated list of integers; these should either be run numbers or client port numbers. If it is ‘all’, all runs in progress will be affected. If it is omitted, the command has no effect. The intention is that this message is sent by the data logging system to stop a run.
- **free** — Free all resources held by this client.
- **help** — Summarize valid commands.
- **info** *report-type* — Get back a formatted report of some aspects of the current state. This report will be sent back as TEXT messages (followed by a DONE message); it is intended to be human-readable. The report types available are:
 - **clients** — Print information about all clients presently connected to COOR.
 - **configs** [*path*] — Print the list of valid configuration names in *path*. If *path* is omitted, use the root directory. In addition to

the configuration names, this command will also print any subdirectories present in *path*. These are distinguished by ending in a slash.

- **configuration** — Return the current configuration in XML form.
- **crates** — Print information about all readout crates that are owned by some client.
- **downloaders** — Print information about the status of COOR's connections to the download targets (and the alarm system).
- **itc** — Print information about all of COOR's ITC connections.
- **11bits** — Print information about all presently defined Level 1 trigger bits.
- **11egs** — Print information about all presently defined Level 1 exposure groups.
- **12bits** — Print information about all presently defined Level 2 trigger bits.
- **12tools** — Print information about all presently defined Level 2 tools/filters.
- **13bits** — Print information about all presently defined Level 3 filter bits.
- **13clients** — Print information about all the presently defined Level 3 client objects.
- **local_crates** — Print information about all readout crates that are owned by this client.
- **local_11bits** — Print information about the Level 1 trigger bits owned by this client.
- **local_12bits** — Print information about the Level 2 trigger bits owned by this client.
- **local_12tools** — Print information about the Level 2 tools and filters owned by this client.
- **local_13bits** — Print information about the Level 3 filter bits owned by this client.
- **13clients** — Print information about the Level 3 client objects owned by this client.

- `local_l1egs` — Print information about the Level 1 exposure groups owned by this client.
 - `local_objects pattern` — Return the names of all objects owned by this client that match the regular expression *pattern*.
 - `local_streams` — Print information about the streams owned by this client.
 - `objects pattern` — Return the names of all objects that match the regular expression *pattern*.
 - `prescale_path prescname` — Print the pathname that would be used for the prescale set named *prescname* for the currently loaded configuration. Prints nothing if there's no configuration loaded, or if the configuration does not support prescale sets.
 - `prescale_sets` — Print the list of available prescale sets for the currently loaded trigger configuration.
 - `store` — Print information about the current store.
 - `streams` — Print information about all presently defined streams.
- `invalidate pattern` — Invalidate all items which match the regular expression *pattern* and are owned by this client. (If *pattern* is omitted, all items are considered to match.)
 - `limit limit` — Request that runs be stopped after approximately *limit* events have been recorded. If *limit* is 0 or omitted, no limit is imposed.
 - `load configname` — Load a new configuration, named *configname*. The details of this are described in Sec. 10. COOR will return information about the loaded configuration as an argument to the DONE message. This will be a string representation of a Python dictionary, that can be read with `eval()`. The keys in this dictionary can include:
 - `autopause` — True or false, depending on the autopause setting for this configuration.
 - `calib_reference` — The calibration reference set keyword specified in the configuration. Only present for calibration configurations.
 - `calib_type` — The calibration type keyword specified in the configuration. Only present for calibration configurations.

- `comics_runtype` — The default COMICS run type keyword specified in the configuration.
 - `configname` — The name of the configuration that was loaded.
 - `physics` — True or false, depending on the physics flag setting for this configuration.
 - `runtype` — The run type keyword string specified in the configuration.
 - `sdaq_type` — The SDAQ type keyword specified in the configuration. Only present for SDAQ configurations.
- `modify name` — Modify the currently downloaded configuration, based on the commands in *name*. This works like loading a configuration (see the `load` command), except that the configuration read should contain neither a `trigdef` nor a `sdaq` element. *The details of this are subject to change.*
 - `num_nodes l3client-num num-nodes` — Request that the level-3 client *l3client-num* be assigned to *num-nodes* nodes. If *num-nodes* is 0, all otherwise uncommitted nodes should be used. The client making this request must not have a run in progress, and must own the requested level-3 client.
 - `pause add-info` — Pause the client’s run. *add-info* should be a set of ‘*keyword: value*’ pairs, separated by newlines. (It may be blank.) This information will be sent to the significant event system, and may be entered in the logbook.
 - `prescale bitnumber prescale ...` — Change the prescale for Level 1 trigger bit *bitnumber* to *prescale*. Multiple *bitnumber–prescale* pairs may be given in the command. All trigger bits being modified must be owned by this client.

If a prescale value ends in a percent sign, it is interpreted as a percentage prescale. If it is zero, then the trigger is disabled. Otherwise, the prescale is interpreted as a ratio.
 - `prescale_set set-name` — Load a predefined prescale set named *set-name*. See Sec. 10.10 for more information.

- **pulser block** *pulser-name pattern* ... — Set the pulser pattern for the pulser *pulser-name* to *pattern*. Note that if *pattern* contains spaces, it should be quoted. Multiple *pulser-name, pattern* pairs may be given in a single command.

The pulser must have been allocated exclusively for this to work.

- **pulser word** *pulser-name/ext value* ... — Set COMICS device “*pulser-name/ext*” to *value*. Note that if *value* contains spaces, it should be quoted. Multiple *pulser-name/ext, value* pairs may be given in a single command. This is a low-level command, intended for hardware debugging.

The pulser must have been allocated exclusively for this to work.

- **reconnect** *dnllist*— If the connections to any of the download targets named in *dnllist* has been lost, this command will attempt to reestablish them. If *dnllist* is omitted, this will be done for all the download targets.
- **recording** *state* — Set the recording state for this client. The *state* argument should be either ‘on’ or ‘off’.
- **reenable** *l1bits* — Tell the trigger framework to reenable some Level 1 trigger bits. The parameters *l1bits* should be a space-separated list of trigger bit numbers. If it is empty, than all auto-disabled triggers owned by this client will be reenabled. This only has an effect if the bits were configured in auto-disable mode. The bits must be owned by this client.
- **reinit** — Reinitialize COOR. Connections to clients are left intact, but connections to download targets will be dropped and reestablished. The parameters and resources files will be reread, but note that not all parameters can be changed during a reinitialization. This command may only be issued if no objects are owned by any client (use **force_reinit** to avoid this restriction).
- **reopen** — Tell COOR to close its current log file and start writing a new one (with a new name).
- **resume** *add-info* — Resume the client’s run after a pause. *add-info* should be a set of ‘*keyword: value*’ pairs, separated by newlines. (It

may be blank.) This information will be sent to the significant event system, and may be entered in the logbook.

- **revalidate** — If any of the items owned by this client are marked as invalid, try to do the required downloads to make them valid again. This is done automatically before starting a run.
- **scl_init** — Request that the level-1 framework reinitialize the serial command links.
- **start *brun-info*** — Start a new run. The run number of the new run will be returned as an argument in the DONE message. *brun-info* should be a set of ‘*keyword: value*’ pairs, separated by newlines. (It may be blank.) This information will be added to the brun file for this run.
- **stop *erun-info*** — Stop the client’s run. *erun-info* should be a set of ‘*keyword: value*’ pairs, separated by newlines. (It may be blank.) This information will be added to the erun file for this run.
- **store_begin *store-number add-info*** — Declare that store *store-number* is beginning. *add-info* should be a set of ‘*keyword: value*’ pairs, separated by newlines. (It may be blank.) This information will be sent to the significant event system, and may be entered in the logbook.
- **store_end *add-info*** — Declare that the current store is ending. *add-info* should be a set of ‘*keyword: value*’ pairs, separated by newlines. (It may be blank.) This information will be sent to the significant event system, and may be entered in the logbook.

When this command is issued, any physics runs in progress will be automatically ended (as with **force_stop**).

- **timeout** — Force the download in progress to timeout. Unlike **abort**, it is global — **abort** will only affect a client’s own downloads. Like **abort**, it does not generate a DONE or FAIL response.
- **update *pattern*** — Request asynchronous updates for changes in COOR’s configuration database for all objects matching the regular expression *pattern*. If *pattern* is omitted, any previous update request is canceled. The messages sent back by COOR when updates occur are of one of three forms:

- `UPDATE_MODATTR` *objname attrs* — Some attributes of *objname* have changed. *Attrs* is a string which when passed through the Python reader will yield a Python dictionary containing the modified attributes.
- `UPDATE_NEWOBJ` *objname attrs* — The object *objname* has been added to the configuration. *Attrs* is a string which when passed through the Python reader will yield a Python dictionary containing the attributes of the new object.
- `UPDATE_DELOBJ` *objname* — The object *objname* has been removed from the configuration.

Only the pattern from the last `update` command is remembered.

- `username` *name [progrname]* — Set a username and optional program name for this client, for use in status displays.

8.2 COOR-Downloader Communication

There is a common protocol for communication between COOR and the targets. Within the context of this protocol are various target-specific commands. We first describe the common protocol, then summarize the target-specific parts.

8.2.1 Common Protocol

The design of the common downloading protocol was the result of several considerations:

- The protocol should be usable for all the download targets.
- It should allow the target to process requests concurrently. This implies that COOR should be able to send multiple download requests to the target without receiving a reply, and that the replies may be sent back to COOR in a different order than that in which the commands were received.
- It should allow the target to process requests in a “batched” fashion — to queue up all the commands for a particular configuration request, and only start processing them once all commands have been received.

This implies that there must be some way to mark the end of a configuration request.

- The protocol should be easy to test, debug, and extend.

The resulting protocol has the following characteristics:

- The ITC package is used for the underlying transport. One command or acknowledgment is sent per ITC message. All messages are of type `String_Message`. Where it makes sense, targets should not be case-sensitive.
- Commands are always sent one way: from COOR to the target. As discussed in Sec. 2, a target that needs to make asynchronous requests should explicitly open an additional command channel to COOR.
- Except as noted below, every command should result in an acknowledgment from the target back to COOR. In some cases, the order of acknowledgment may not be the same as the order in which the commands were issued. In order to keep straight the correspondence between commands and acknowledgments, each command has a “command id” which is sent with the command and returned with the acknowledgment. Targets should not assume anything about the format of this id, other than that it consists of printable characters, contains no whitespace, and is no longer than 32 characters. *Do we want to make stronger guarantees here? E.g., that it’s a monotonically increasing number?*
- Except as explicitly, noted, commands can be *batched*. A batch is implicitly started by the first batched command received. It is ended by the special command `configure`. When `configure` is received, the batched commands should describe a consistent configuration. Once the `configure` command is sent, no additional commands will be sent (except for `abort`, and, if the connection breaks, `init`) until every command in the batch (including the `configure` command) has been acknowledged.

The target can start processing commands at any time. It can process them as they are received, or it can queue them up and process them all once the `configure` command has been received. Acknowledgments can be sent before the `configure` command arrives. Except

for the `configure` command (which must be acknowledged last) *and for commands within a block*, commands may be acknowledged in any order.

Note that it is possible for COOR to send a `configure` command with no preceding commands. Targets should simply acknowledge and ignore these requests. (Actually, this should not happen any more, but it's still a good idea for targets to be able to deal with it.)

Commands which are not batched are called *immediate*. They will not be followed by a `configure` command, and in most cases, no other commands will follow (except for `abort` and possibly `init`) until the command is acknowledged.

The general format of a command sent by COOR to the targets is as follows:

command-id *command* [*args...*]

The target should reply to the command with a message of the form

command-id *status* [*text*]

The *status* should be one of the strings 'ok', 'bad', 'progress', or 'more'. The optional *text* is either status information being returned from the command (if *status* is 'ok') or an error message (if *status* is 'bad'). If *text* would take more than one line, each line should be sent separately, in order. For all except the last line, *status* should be 'more'. For the last line, *status* should be the final value (either 'ok' or 'bad').

If *status* is 'ok' and the command was not requesting any information, then *text* may be blank. If *status* is 'bad', *text* should contain a brief error message.

A *status* of 'progress' is similar to 'more', except that the message is displayed to the user immediately, instead of being saved until an 'ok' or 'bad' is received. When a 'progress' message is processed, it will also reset the download timeout. Thus, a downloader working on a lengthy operation can prevent COOR from timing out by periodically sending these messages. (In such a case, the *text* field should be a message reassuring the user that things are still working properly.)

There is a set of common commands which should be recognized by all targets. They might not have to do anything for some of them, but they should be able to recognize and acknowledge them:

- **configure** — As discussed above. A simple target which processes all commands as they are received can ignore these messages.
- **abort** — This command is somewhat special. It is sent to the targets by COOR when a download has been aborted. When this command is received, the target may discard any commands which it has queued, but has not yet processed. If there are no queued commands, the abort request should be ignored. (In particular, the target should *not* undo any commands which it has already reported as completing successfully.) The target need not respond to the **abort** command.

A simple target which processes all commands as they are received can ignore these messages.

- **init** — This is an immediate command. The target should immediately end all ongoing DAQ, release resources, and restore all programming to the default state.

It should not be necessary to reboot nodes, reload FPGAs, etc. in response to this command. The assumption being made is that the target system is still sane, but is in an unknown state.

This is sent by COOR on startup, and whenever a connection to a target has been broken and reestablished.

- **start_run** *runno spectrigrs* — This is an immediate command. Run number *runno* is starting. A successful reply to this command implies that the target is now ready for that run to start. *Spectrigrs* is a list of Level 1 specific triggers participating in the run. It is a space-separated list of integers. (*This may get abbreviated using a notation like FIRST:LAST to specify a range.*)

At the time this message is issued, all the triggers associated with the run in question will be disabled, so data for that run will not be flowing yet.

- **stop_run** *runno* — This is an immediate command. Run number *runno* is stopping. A successful reply to this command implies that the target has done what it needs to do in order to stop the run.

At the time this message is issued, all the triggers associated with the run in question will be disabled. *But there is presently nothing to synchronize flushing of any buffered data. Is this needed?*

- `begin_block`
- `end_block` — *These are not really commands, per se, and need not be acknowledged. Commands which occur between `begin_block` and `end_block` must be processed in the order in which they were sent. Only batched commands may appear within a block, and `configure` may not appear within a block.*
Not all order dependencies will be protected within a block. In general, if it doesn't make sense to reorder the commands they won't be put into a block. (This will probably only be used for EPICS downloads.)
- `pause_run runno`
- `resume_run runno` — These are immediate commands. Note that `run runno` is being paused or resumed. These messages are advisory only: separate commands will be sent to actually enable or disable data flow (for example, disabling level1 trigger bits).
- `begin_store storenum`
- `end_store storenum` — Note that a store is beginning or ending.

Note one exception to the above protocol: all messages sent by COOR to the data logging system will have the string 'COOR ' prefixed to them.

The following sections summarize the target-specific commands for each target. Note that these are not intended to be complete summaries of the commands which the targets can accept; rather, they document the subset of those commands which the present implementation of COOR will actually send.

8.2.2 COMICS

The process used to download EPICS devices is called COMICS. Every downloadable EPICS device has an object name, which consists of a *class-tag: name* pair. (See Sec. 6.) The downloadable state of a device consists of a set of named attributes, each of which has some value.

The command to request a download consists of the device name (without the class tag) followed by a list of attribute name-value pairs:

```
set name attr value ...
```

Any *value* which contains embedded whitespace should be enclosed in single quotes.

If the command string contains “VERIFY ’YES” , then this is a request for a verify transaction.

8.2.3 Level 1

The Level 1 target is presently used to configure both the Level 1 trigger framework and the Level 1 calorimeter trigger.

Here are the commands used to configure the Level 1 framework:

- `increment_lbn`

Request a new luminosity block number from Level 1. The TCC should increment its LBN count and return it (in decimal representation) as the body of the reply message, in the format

`<lbn>`

- `L1FW_Pause`

- `L1FW_Resume`

Pause or resume all trigger processing. At present, these messages are generated only around trigger enable or disable requests, to ensure that they take effect simultaneously. (But the pause/resume framing may be omitted if only a single bit is being enabled or disabled.)

- `L1FW_Expo_Group` *egnumber*

[`Deallocate`]

[`And_Or_List` *termstring*]

[`Geo_Sect_List` *geosect-string*]

Configure L1 exposure group *egnumber*. The list of associated and/or terms is given by *termstring*. This is a space-separated list of integers; a dash before an integer indicates that that particular term is to be vetoed. The list of geographical sections to read out is given by *geosect-string*. This is a space-separated list of integers, except that a range of consecutive integers from *first* to *last* inclusive may be written using the notation ‘*first:last*’.

If the `Deallocate` keyword is present, the exposure group should be reset to its default configuration.

- L1FW_spec_trig *bitnumber*
 - [deallocate]
 - [Prescale_Ratio *prescale*]
 - [Prescale_Percent *prescale*]
 - [L2_Unbiased_Sample *ratio*]
 - [L1_Qualifier *l1qualifiers*]
 - [Obey_FE_Busy]
 - [Auto_Disabled]
 - [Re_Enable]
 - [coor_enable]
 - [force_l2reject]
 - [Obey_Individual_Disable 0]
 - [expo_group *egnumber*]
 - [And_Or_List *termlist*]

Configure L1 specific trigger *bitnumber*. If *bitnumber* starts with a dash, any boolean options mentioned in the command are to be turned *off*; otherwise, they are to be turned *on*.

The keywords in the command have the following meanings:

- `deallocate` — Reset the bit to its default configuration.
- `Prescale_Percent prescale` — Set the bit's prescale to *prescale*, as a percentage. The *prescale* value should be between 0 and 100.
- `Prescale_Ratio prescale` — Set the bit's prescale to *prescale*, as a ratio. (I.e., pass 1 of *prescale* events.)
- `L2_Unbiased_Sample ratio` — Set the Level 2 unbiased sample ratio for this bit. This feature controls the assertion of the L2 unbiased sample L1 qualifier flag for some fraction of the events. (There is only one such L1 qualifier flag common to all 128 trigger bits.) This feature is implemented with a 24-bit counter, giving an allowed value range of $1-2^{24} = 16,777,216$. When a ratio of N is programmed, the counter is initialized with a random number between 0 and $N - 1$. The counter is decremented every time the bit fires. The L2 unbiased sample L1 qualifier flag will be set for the one event where this counter reaches the value zero. A ratio of 1 corresponds to asserting the qualifier for every event for which this trigger bit fires. This feature cannot be disabled; instead, set the ratio to its maximum value.

- `L1_Qualifier` *l1qualifiers* — Set the bit’s L1 qualifier mask to *l1qualifiers*, which should be a space-separated list of integers.
- `Obey_FE_Busy` — Turn on/off whether or not the bit is disabled on front-end-busy.
- `Auto_Disabled` — Turn on/off auto-disable (one-shot) mode.
- `Re_Enable` — Reenable an auto-disabled bit.
- `coor_enable` — Enable/disable this trigger bit.
- `force_l2reject` — If true, then this bit should not be processed by level 2. Instead, just treat it as if it was rejected by level 2.
- `Obey_Individual_Disable` 0 — If true, this bit can be disabled by level 3. Otherwise, this bit ignores disables from level 3.
- `expo_group` *egnumber* — Set the exposure group associated with this trigger bit to *egnumber*, which should be an integer.
- `And_Or_List` *termlist* — Set the list of and/or terms for this bit to *termlist*. This is a space-separated list of integers; a dash before an integer indicates that that particular term is to be vetoed.

- `L2_Global_Ignored`

Specify that L2 is not to be used for making trigger decisions. Instead, the L2 framework should wait a fixed amount of time ($\sim 100 \mu s$) and automatically accept all the specific triggers present in the L1 specific trigger fired mask (except for those specific triggers programmed with the “Force_L2Reject” message).

COOR will send this message only immediately after an `init`.

- `L2_Global_Obeyed`

Specify that L2 is to be used for making trigger decisions.

COOR will send this message only immediately after an `init`.

- `SCL_Initialize`

Reinitialize the serial command links. This is an immediate command.

Here are the commands used to configure the Level 1 calorimeter trigger:

- **L1CT_Energy_Threshold** *type* **Comparator** *number* **Value** *thresh*
Set a threshold for comparator *number* (an integer) of type *type* to *value* (a floating point number). The possibilities for *type* are ‘EM_Et’, ‘HD_Et’, ‘TOT_Et’, and ‘Miss_Pt’.
- **L1CT_Ref_Set** *type* *number* *contents*
Set the reference set *number* of type *type* to the string *contents*. (This string is passed through COOR uninterpreted, except that separate lines are sent as separate messages.) The *type* keyword may be one of the following: ‘EM_Et_Ref_Set’, ‘HD_Veto_Ref_Set’, ‘TOT_Et_Ref_Set’, and ‘Large_Tile_Ref_Set’.
Alternatively, if *contents* is the string ‘Deallocate’, then this reference set is being deallocated.
- **L1CT_Count_Threshold** *type* **Ref_Set** *refset* **Comparator** *number* **Value** *value*
Set the count threshold *number* (an integer) of type *type* associated with reference set *refset* (an integer) of that type to *value* (an integer). The possibilities for *type* are: ‘EM_Et_Towers’ and ‘Tot_Et_Towers’.

8.2.4 Level 2

Here are the commands used to send Level 2 programming information:

- **L2Crate_List** *crate-list* Declare the set of Level 2 crates that are to be used. *crate-list* is a space-separated list of names. This message gets sent before a run transition.
- **l2script** *script-text* Send the text in *script-text* to Level 2.

8.2.5 Level 3

Here are the commands COOR sends to configure Level 3:

- **clear_client** *client-number*
Delete all configuration information for client *client-number*. The number may then be reused in subsequent configuration messages.

- `define_trigger` *l3bit client-number l1bit l2bit filter-name*
 Define L3 bit number *l3bit* for client *client-number*, associated with the L1 bit *l1bit* and the L2 bit *l2bit*, with name *filter-name*.
- `dsm_addr` *dsm-host dsm-port*
 This tells Level 3 that the DAQ State Manager can be found at network address *dsm-host:dsm-port*. This is an immediate command, and is sent just after the `init` message.
- `farm_nodes` *client-number type-name num-nodes*
 The configuration being requested by client *client-number* wants to be assigned *num-nodes* level-3 nodes of type *type-name*. A *typename* of “REGULAR” will be used for normal running. If *num-nodes* is nonzero, then that many nodes should be allocated for the exclusive use of this client. Otherwise, if *num-nodes* is 0, this client wants to share all otherwise unassigned nodes in the farm with other users. (For normal running, *num-nodes* should be 0.)
- `l1bit` *l1bit l1bit-name geosect-list*
 Note that Level 1 bit *l1bit* is named *l1bit-name* and that it reads out the crates specified by *geosect-list*. The latter is a space-separated list of integers, except that a range of consecutive integers from *first* to *last* inclusive may be written using the notation ‘*first:last*’.
- `l2bit` *l2bit l2bit-name*
 Note that Level 2 bit *l2bit* is named *l2bit-name*.
- `runinfo` *client-number runnumber*
 Declare that run number *runnumber* is being started by client *client-number*. This is an immediate command, and is sent just before the `start_run` command.
- `set_client` *client-number configname*
 Start definitions for a new client *client-number*. *Configname* gives the name of the configuration which this client is loading.

- **stream** *stream-number client-number stream-name*

Define a new stream for client *client-number*. The number of the stream is *stream-number*. This is a small integer, and is globally unique. The name of the stream is given by *stream-name*; this should be unique among all streams for a given client, but will not necessarily be globally unique.

- **trigger_list** *client-number trigger-list-text*

Provide the trigger list text for client *client-number*. The trigger list should refer to Level 1, Level 2, and Level 3 bits and streams by name; it is the responsibility of Level 3 to map these to the appropriate numbers using the information contained in the other messages.

If *trigger-list-text* is missing, this should be interpreted as a request to pass all events to all defined streams.

8.2.6 SDAQ

Here are the commands COOR sends to configure the secondary data acquisition system.

- **clear_client** *client-number*

Delete all configuration information for client *client-number*. The number may then be reused in subsequent configuration messages.

- **dsm_addr** *dsm-host dsm-port*

This tells SDAQ that the DAQ State Manager can be found at network address *dsm-host:dsm-port*. This is an immediate command, and it sent just after the **init** message.

- **11bit** *client-number bitnumber...*

Declare that the configuration for client *client-number* is using the trigger framework with level-1 bit number *bitnumber*. For such configurations, COOR will not send **sdaq_run** or **sdaq_stop** commands. If the configuration uses multiple level-1 bits, they all will be listed in this message (separated by spaces). Note that it is also possible for this message to contain *no* level-1 bit numbers. This can happen, for example, in the case where a SDAQ run is parasitic off of a PDAQ run set up from a different configuration.

- **runinfo** *client-number runnumber*
 Declare that run number *runnumber* is being started by client *client-number*. This is an immediate command, and is sent just before the **start_run** command.
- **sdaq_crates** *client-number crate-list*
 Declare the list of crates to be read out by *client-number*. *Crate-list* is the list of geographical sectors to use. It is a space-separated list of integers, except that a range of consecutive integers from *first* to *last* inclusive may be written using the notation '*first:last*'.
- **sdaq_run** *runnumber*
 SDAQ should start data flowing for run *runnumber*. This is an immediate command.
- **sdaq_stop** *runnumber*
 SDAQ should stop the data flow for run *runnumber*. This is an immediate command.
- **sdaq_type** *client-number type-string*
 Declare the SDAQ type string. The *type-string* parameter is passed through unmodified from the input configuration.
- **set_client** *client-number configname*
 Start definitions for a new client *client-number*. *Configname* gives the name of the configuration which this client is loading.
- **stream** *stream-number client-number stream-name*
 Define a new stream for client *client-number*. The number of the stream is *stream-number*. This is a small integer, and is globally unique. The name of the stream is given by *stream-name*; this should be unique among all streams for a given client, but will not necessarily be globally unique.

8.2.7 Data Logging

Since data logging configuration information is associated with particular clients of COOR, we must have some way of identifying these clients to the logging system. For this purpose, each client needing logger configuration is assigned a small integer “client number.” The commands sent are as follows:

- `clear_client` *client-number*
Delete all configuration information for client *client-number*. The number may then be reused in subsequent configuration messages.
- `l1bit` *client-number l1bit-number bit-name*
Declare that client *client-number* is using level-1 trigger bit number *l1bit-number* named *bit-name*.
- `l2bit` *client-number l2bit-number l1bit-number bit-name*
Declare that client *client-number* is using level-2 trigger bit number *l2bit-number*, depending on level-1 trigger bit number *l1bit-number* and named *bit-name*.
- `l3bit` *client-number l3bit-number l2bit-number bit-name*
Declare that client *client-number* is using level-3 trigger bit number *l3bit-number*, depending on level-2 trigger bit number *l2bit-number*, and named *bit-name*.
- `lbn` *client-number lbn*
This message is sent before begin-run and end-run commands. Before a begin-run command, it means that all events for the new run for *client-number* will have a luminosity block number greater than or equal to *lbn*. Before an end-run command, it means that all events for the run being ended will have a luminosity block number less than *lbn*. The *lbn* field will be -1 if the run does not use level-1 or if there was an error retrieving it from level-1.
- `QUERY RUN`
Produce a report summarizing the status of all runs in progress. This should be appended to the end of the reply message (the *text* field). For each run, there should be a space-separated triple of numbers, giving

the run number, number of events received for this run, and event rate in Hz. The triples for different runs are separated by newlines.

- **runinfo** *client-number runnumber*

Declare that run number *runnumber* is being started by client *client-number*. This is an immediate command, and is sent just before the **start_run** command.

- **set_client** *client-number* [**recording on**] [**recording off**]
[*configname configname*]

Change configuration information for client *client-number*. *Configname* gives the name of the configuration which this client has loaded. The strings ‘**recording on**’ and ‘**recording off**’ toggle recording on and off for this client.

- **stream** *stream-number client-number rebrate stream-name family-name family-rate*

Define a new stream for client *client-number*. The number of the stream is *stream-number*. This is a small integer, and is globally unique. The name of the stream is given by *stream-name*; this should be unique among all streams for a given client, but will not necessarily be globally unique. The file family name is given by *family-name*. The parameter *rebrate* is a floating-point number, giving the relative expected data rate for this stream. The logging system can use this as a hint for assigning resources to streams. (The stream messages for a given client will be sorted in order or descending *rebrate*.) *family-rate* is the sum of *rebrate* over all of this client’s streams in family *family-name*.

8.2.8 Calibration manager

Here are the commands COOR sends to configure the calibration manager.

- **calib_ref** *client-number reference-string*

Declare the calibration reference string. The *reference-string* parameter is passed through unmodified from the input configuration.

- **calib_type** *client-number type-string*

Declare the calibration type string. The *type-string* parameter is passed through unmodified from the input configuration.

- **clear_client** *client-number*
Delete all configuration information for client *client-number*. The number may then be reused in subsequent configuration messages.
- **coor_addr** *coor-host coor-port*
This tells the calibration manager that COOR's client port is at network address *coor-host:coor-port*. This is an immediate command, and it sent just after the `init` message.
- **crates** *client-number crate-id-list*
Declare the list of crates to be read out by *client-number*. *Crate-id-list* gives the crate ID numbers of the crates participating in the calibration. This is a space-separated list of integers, except that a range of consecutive integers from *first* to *last* inclusive may be written using the notation '*first:last*'.
- **dsm_addr** *dsm-host dsm-port*
This tells the calibration manager that the DAQ State Manager can be found at network address *dsm-host:dsm-port*. This is an immediate command, and is sent just after the `init` message.
- **runinfo** *client-number runnumber*
Declare that run number *runnumber* is being started by client *client-number*. This is an immediate command, and is sent just before the `start_run` command.
- **set_client** *client-number configname*
Start definitions for a new client *client-number*. *Configname* gives the name of the configuration which this client is loading.
- **stream** *stream-number client-number stream-name*
Define a new stream for client *client-number*. The number of the stream is *stream-number*. This is a small integer, and is globally unique. The name of the stream is given by *stream-name*; this should be unique among all streams for a given client, but will not necessarily be globally unique.

8.3 COOR-Logbook Communication

The communication protocol between COOR and the logbook is based on XML. COOR sends to the logbook an XML-formatted log entry, as described below. The response from the logbook should be one of the following strings, along with a terminating newline:

- `<SUCCESS/>` — The entry was successfully committed to the logbook.
- `<FAIL/>` — There was a transient failure in committing the entry. The operation might succeed if retried later.
- `<ERROR/>` — There was a permanent failure in committing the entry (i.e., the message sent was formatted incorrectly). If the same operation is retried, it would fail again.

The logbook entry message should consist of a single `MESSAGE` element, containing the elements `OPERATOR`, `CATEGORY`, `TOPIC`, `KEYWORD`, and `TEXT`. The `MESSAGE` element has a `TYPE` attribute, which should be set to `"text"`. Here is an example:

```
<MESSAGE TYPE="text">
<OPERATOR>Coor</OPERATOR>
<CATEGORY>DAQ_Shift/Log</CATEGORY>
<TOPIC>General_Log</TOPIC>
<KEYWORD>begin_run</KEYWORD>
<TEXT><![CDATA[2001 Jul 02 17:15:41 CDT Start run 1355
    calmuon-1.0
    Config type: test Comics type: data
    Recording on Store: 12345
    Owned crates:
        ecnse(0x4a) cmesc(0x3b) trgfr(0x1f) l3wakeup(0x7f)
    Writing streams:
        daq_test
]]></TEXT>
</MESSAGE>
```

8.4 Run Transition SES Messages

For each run transition, COOR sends a message to the significant event system (SES). These are messages of type `SE_Run_Message`. The body of the

message has the following form:

type number lbn d

In the above, *type* is one of the following strings:

- `run_started`
- `run_ended`
- `run_paused`
- `run_resumed`
- `store_started`
- `store_ended`

The field *number* is either the run number or the store number, and *lbn* is the luminosity block number at the transition. The field *d* is a string representation of a python dictionary. This dictionary will contain both the run attribute information returned after a `load` command (see Sec. 8.1) as well as any information that was entered by the operator via a `TAKER` dialog.

8.5 Name Server Messages

COOR also provides a simple name/value service. This service is accessed by connecting to the port given by `d0online_names.COOR_NAME_SERVER_ADDR`.

Names are hierchical, with components being separated by a `'.'`. This, one can have names like `' .coor.storenum'` or `' .cal.ccne.peds'`. The value of a name is thus either a string or a set of other names. In addition to the primary value, a name can have a set of string-valued properties associated with it.

The messages sent to the server are itc `String_Message`'s, with the format as below. A reply is sent to each message, also a `String_Message` (except for "getall"); the first word in this string is either `'ok'` or `'bad'`.

Here is a list of the messages recognized by the name server.

- `get name` — Return the value of *name*. If *name* has a simple string value, this returns

ok name value

If *name* is a directory, this returns

`ok name. namelist`

(note the trailing period on *name*). The field *namelist* is a space-separated list of the names in the directory. Any of these which are themselves directories will have a trailing period.

Otherwise, this returns ‘bad’.

- `set name value` — Set *name* (which must not already be a directory) to *value*. Any nonexistent directories specified in *name* are automatically created. Returns either ‘ok’ or ‘bad’.
- `getprop name prop` — Return the property *prop* of *name*. The property must have been previously set with “setprop.” Returns either ‘ok *value*’ or ‘bad’.
- `setprop name prop value` — Set property *prop* (which must not start with an underscore) of *name* (which must already exist) to *value*. Returns either ‘ok’ or ‘bad’.
- `del name` — Delete *name*. If *name* is a directory, delete the entire tree under it. Returns either ‘ok’ or ‘bad’.
- `getall name` — Returns *name*, and everything underneath it (if it’s a directory) as a Python dictionary wrapped in a `Py_Message`. If there’s a problem, `None` will be returned instead.
- `checkpoint` — Forces the database to be written to disk immediately. Returns either ‘ok’ or ‘bad’

9 Run Transitions

Here we summarize the actions taken when starting and stopping runs.

Starting a run:

- Do a revalidation step. If this fails, the run cannot start.
- Request a new luminosity block number from Level 1 (only for runs using primary DAQ). If this fails, the run cannot start.
- Send a `start_run` message to all targets. Wait until all have responded. If any report an error, the run cannot start. Once all have responded, we take care of all the other actions taken at start-run: writing the brun file, notifying the significant event system, and so on.
- Tell the L1 framework to enable the trigger bits for this run.

Stopping a run:

- Tell the L1 framework to disable the trigger bits for this run.
- Request a new luminosity block number from Level 1 (only for runs using primary DAQ).
- Send a `stop_run` message to all targets. Wait until all have responded. Once all have responded, we take care of all the other actions taken at end-run: writing the erun file, notifying the significant event system, and so on.

10 Writing Trigger Configurations

Trigger configurations describe how to configure the trigger system and how devices are to be downloaded. This section discusses the language used to describe these configurations.

10.1 Finding Configurations

COOR locates trigger configurations using a configuration name. These configurations can come from different places and be in different formats. COOR maintains a list of “configuration loaders.” Given a configuration name, COOR asks each loader in turn to load that configuration, until one succeeds. The configuration loaders presently defined are listed below.

10.1.1 Python Script

Given *configname*, if a file *configname.py* exists in the directory given by the configuration parameter `trigger_config_path` (normally the value of the environment variable `COOR_CONFIG_ROOT`), it is read as a python script. This file should define a function called *configname*, which takes the clientstat object as an argument and returns the download list. Note that the details of this are subject to change; for most trigger configurations, using the XML format (described below) is recommended.

10.1.2 XML

Given *configname*, if a file *configname.xml* exists in the directory given by the configuration parameter `trigger_config_path` (normally the value of the environment variable `COOR_CONFIG_ROOT`), it is read as an XML format trigger configuration. The details of this format are described in the remainder of this section.

Before reading a configuration file, COOR scans for files in the `readouts` subdirectory of the configuration tree. Any files found there with a `.xml` extension and names that are legal XML names are made available as XML external entities. For example, if there is a file `readouts/allcrates.xml`, you can include its contents in a trigger configuration with `&allcrates;`

Note also that COOR will not search the `readouts` subdirectory for trigger configurations.

Implementation note: The XML reader works by first transforming the XML into a DOM tree; all COOR-specific processing is done using the DOM tree. The XML/DOM library from the XML SIG is used. This implies that the tool which extracts the trigger configuration files from the database might best work by constructing a DOM tree and then converting that to XML. If, in the future, it is desired to get rid of the intermediate file and have COOR access the database directly, this would make pasting the two pieces together easier.

10.1.3 Dumped States

Whenever a recorded run is started, COOR dumps out its state in a file called `runnnnnnnnn.state`, where `nnnnnnnn` is the run number. These files can be read back in to reproduce previous configurations.

Given *configname*, if a file `configname.state` exists in the directory given by the configuration parameter `trigger_config_path` (normally the value of the environment variable `COOR_CONFIG_ROOT`), it is read as a dumped state. Note that *configname* should have the format `runnnnnnnnn`.

10.1.4 Custom XML

A configuration can also be defined by sending an XML string directly to COOR, instead of reading it out of a file. For this to happen, the “configuration name” should start with the string ‘`custom:`’; the remainder of the string is the trigger configuration itself.

10.2 XML

The trigger configuration syntax is based on the extended markup language (XML) from the WWW Consortium. XML is (almost) a subset of SGML, and is thus also related to HTML. For a full description of XML and its use from Python, follow links from <http://www.python.org/topics/xml>. Here, we simply outline a few of its essential features.

An XML document can be viewed as a tree of *elements*. An element can contain other element; it can also contain text. Each element has a set of attribute-value pairs associated with it. Elements are classed into named types; the set of allowed attributes for an element is determined by its type. An element type may also have restrictions on the types or ordering

of the elements that it contains. This information about the element types is contained in a “document type definition” (DTD).

The syntax for writing an element in an XML file looks like this:

```
<element-name attribute=value ... >
  element-contents
</element-name>
```

Note that XML is case-sensitive. For the special case of an empty element (no contents), this may be abbreviated:

```
<element-name attribute=value ... />
```

See Sec. 10.3 for examples.

10.3 Example

This section presents a couple annotated examples of trigger configurations. For the full details on the element types used, see the following reference sections. In addition, the DTDs used to read the trigger configurations are reproduced in Appendix A.

Here is an example of a configuration using the primary DAQ readout:

```
1 <?xml version="1.0"?> [1]
2 <!DOCTYPE configuration SYSTEM "trigger_config.dtd"> [1]
3
4 <!-- A configuration request for testing. --> [2]
5 <configuration name="comics_test1" version="1.0"> [3]
6
7   <download> [4]
8     <Cal_ADC_Crate name="ecnnw" blsmode="normal"/>
9     <Cal_ADC_Crate name="ecnsw" blsmode="normal" ownmode="exclusive"/> [4]
10  </download>
11  <download name="muocrates">
12    <Muo_Crate name="fmsm"/>
13    <Muo_Crate name="fmmn"/>
14  </download> [4]
15  <download name="l2crates">
```

```

16     <L1_Crate          name="l1cal"/>
17     <L2_Crate          name="l2cal"/>
18     <L2_Crate          name="l2glb"/>
19 </download>
20 <crate_list name="calcrates"> 5
21     <crateref ref="ecnnw"/>
22     <crateref ref="ecnsw"/>
23 </crate_list>
24 <crate_list name="allcrates">
25     <crateref ref="calcrates"/>
26     <crateref ref="muocrates"/>
27     <crateref ref="l2crates"/>
28 </crate_list> 5
29
30 <level2> 6
31     <l2calem pt="10"/> 7
32
33     <l2emtool name="pt3" ieta="1" iphi="1" minet="3" requiretrack="-1"/> 8
34
35     <l2emfilter name="pt10"
36         emfrac="0.15" minet="10" isofrac="0.25" tool="pt3"/> 9
37 </level2> 6
38
39 <l1refsets> 10
40     <l1jet_refset name="jet10">
41         Value 10.0
42     </l1jet_refset>
43 </l1refsets> 10
44
45 <trigdef> 11
46     <expogroup name="eg1" readout="allcrates"> 12
47         <l1termlist> 13
48             <l1specterm name="fastz"/>
49         </l1termlist> 13
50         <l1trigger name="l1bit1" prescale="5"> 14
51             <l1termlist> 13
52                 <l1specterm name="fastz"/> 15
53                 <l1specterm name="pbar_halo" require="veto"/> 16

```

```

54         <l1jetcount jet_refset="jet10" count="1"/> [17]
55         <l1muo name="mu2pt2c1lo"/> [18]
56     </l1termlist> [13]
57     <l2trigger name="l2bit1"> [19]
58         <l2script>
59             <l2filter name="pt10"/>
60         </l2script>
61
62         <l3trigger name="l3bit1"/> [20]
63     </l2trigger>
64 </l1trigger>
65 </expogroup>
66 <triglist> [21]
67     This is some trigger list text.
68 </triglist>
69 </trigdef>
70
71 <stream name="xstr" relrate="2.4"/> [22]
72 </configuration> [23]

```

-
- [1] Lines 1 and 2: These two lines are required at the start of every trigger configuration file. They identify the file as using XML and give the DTD to which the file conforms.
 - [2] Line 4: This shows the syntax of an XML comment.
 - [3] Line 5: This line is the start of the top-level element of the file, named `configuration`. It identifies the name and version of this configuration. Additional options may also be specified here; see the reference section for details. Note that if the configuration is read from a file, the file name must match the name and version specified here. For this example, the file must be called “`comics_test1-1.0.xml`”.
 - [4] Lines 7–14: This is the `download` element, in which the downloads to EPICS devices are defined. The `download` element contains one element for each device to be allocated and downloaded; the `name` attribute gives the name of the device being requested. Each device also has a device type, and the name of the element used to reference a device

must match its type. Devices can be allocated as either “shared” or “exclusive” — if “shared,” then other configurations can also allocate this device, provided that the requests are compatible. Shared is the default, but may be overridden by explicitly specifying the `ownmode` attribute, as in line 9. Note that if you are going to want to make changes to the device after the initial download, it should be allocated in exclusive mode. The additional parameters depend on the device type, and are described in Sec. 10.6.

Note that multiple `download` elements are allowed.

- 5 Lines 20–28: The next section contains a set of `crate_list` elements. These are a way of assigning a name to a set of readout crates, in order to be able later to specify sets of crates to read out. Each `crate_list` element has a `name` attribute, which is the name of the list being defined. It then contains a set of `crateref` elements, each of which names either a readout crate defined in the `download` section or another `crate_list`. All the referenced crates are merged together and duplicates eliminated; the ordering doesn’t matter. Note that circular `crate_list` references are not allowed. Empty lists are allowed (but are probably not very useful).

Note that you can also define a crate list by using the `name` attribute on the `download` element. Also, anywhere where you could specify a crate list name, you can also just give a space-separated list of crate (or crate list) names.

- 6 Lines 30–37: This element contains definitions for level 2 tools and filters that may be used in this configuration.
- 7 Line 31: This element defines a level 2 preprocessor, saying that we want to use the `12ca1` preprocessor. This requires that we have already allocated the `12ca1` readout crate above.
- 8 Line 33: This element defines a level 2 tool. The tool is of type “`12emtool`” and is named “`pt3`”. This name is used only in order to identify this tool from elsewhere within this configuration; it is not sent to level 2. The remaining attributes are the tool attributes.
- 9 Line 36: This element defines a level 2 filter. Actually, as far as `COOR` is concerned, there is no difference between filters and tools; they just

have different names. In the attributes for this filter, note the reference to the tool defined earlier.

- [10] Lines 39–43: This element contains definitions for the level 1 calorimeter trigger reference sets. A reference set associates a threshold with each trigger tower in the calorimeter. The precise syntax for the contents of a reference set is specified in the calorimeter trigger programming document. Reference sets come in several flavors: electromagnetic, electromagnetic veto, jet (total energy), and large tile. Each reference set also has a name, allowing it to be referenced from the trigger term definitions below. The example shown here defines a jet reference set named “jet10” with a uniform threshold of 10 GeV.
- [11] Line 45: This is the start of a `trigdef` element, which groups together the trigger programming for a group of level 3 nodes. You may have multiple `trigdef` elements if you have a configuration that uses more than one level 3 node group. A `trigdef` element has optional attributes to specify the size of the node group and the node type desired. A `trigdef` element contains a set of `expogroup` elements, followed by a `triglist` element, which contains the level 3 filter programming.
- [12] Line 46: Trigger definitions are organized as a tree, with the leaves being the level 3 filters. Each level 3 filter is associated with a level 2 bit, and each level 2 bit is associated with a level 1 bit. Each level 1 bit, in turn, is associated with an exposure group.

The exposure groups are the basis for calculating luminosity. Level 1 bits with the same crate readout list and the same set of exposure-sensitive trigger terms are grouped into exposure groups. (This is because the resources required to maintain a scalar for each bunch for each trigger bit are prohibitive; but it is feasible to do it for the smaller number of exposure groups.)

Each exposure group should have a name, for monitoring and logging purposes. The `readout` attribute should name a `crate_list`; it gives the crates which the level 1 bits in this exposure group will read out. An `expogroup` element contains a `l1termlist` element, giving the trigger terms for this group, followed by a set of `l1trigger` elements.

An `expogroup` element may also appear outside of a `trigdef` element, if level 3 is not to be used for these triggers. In this case, the exposure

group should not contain any `l2trigger` definitions.

- [13] Lines 47–49 and 51–56: These are examples of level 1 term lists. Term lists are associated both with exposure groups and with level 1 triggers. A term list consists of a list of level 1 terms; the currently defined elements for level 1 terms are described in Sec. 10.7.

Every level 1 term element supports the attribute `require`, which can be set to either `require` (the default) or `veto`.

- [14] Line 50: The start of a level 1 trigger definition. Each such definition must specify a `name` attribute. Additional, optional, attributes are described in Sec. 10.5.

A `l1trigger` element contains a `l1termlist` element, giving the terms required for this trigger bit. Note that this list must be a superset of the term list for the level 1 bit’s exposure group. The `l1termlist` element is followed by the list of level 2 triggers associated with this level 1 bit.

- [15] Line 52: This is a “specific” named and/or term — here named “fastz”.
- [16] Line 53: Another specific and/or term, but here we require that the term *not* be asserted.
- [17] Line 54: This is an example of a calorimeter trigger term. It requires one trigger tower with a total energy exceeding that required by the “jet10” reference set.

- [18] Line 55: This is an example of a muon trigger term.

- [19] Line 57: The start of a level 2 trigger definition. Each `l2trigger` element must have a `name` attribute specified, to supply a name for the bit. At the start of the `l2trigger` element is a `l2script` element. This contains the level 2 programming for this bit, specified as a list of `l2filter` elements. to see. If the `l2script` element is omitted, level 2 will always pass this bit. Following the `l2script` element is the list of level 3 triggers associated with this level 2 bit.

If a `l1trigger` element contains no `l2trigger` elements, then that level 1 bit is programmed to always generate level 2 rejects.

- [20] Line 62: A level 3 trigger definition. Each such element must contain a `name` attribute, to name the bit. No other information is supplied at this point; the actual filter definitions are contained in the trigger list supplied in the `triglist` element.
- [21] Line 66: The `triglist` element contains the level 3 trigger list script. This text is passed through to level 3 untouched by `COORD`. Note that if it contains characters which are special to XML, you might want to put it inside a `CDATA` construction.
- [22] Line 71: Finally, a set of `stream` elements defines the recording streams. Each takes a `name` attribute; the optional `relrate` attribute provides information to the logging system for load balancing.
- [23] Line 72: This line is the end of the top-level element, and thus the end of the trigger configuration.

For comparison, here is another example using the secondary DAQ system:

```

1 <?xml version="1.0"?>
2 <!DOCTYPE configuration SYSTEM "trigger_config.dtd">
3
4 <configuration name="comics_test2" version="1.0">
5   <download>
6     <SMT_Crate name="smt0_0"/>
7   </download>
8   <sdaq type="sdaqtype" readout="smt0_0"/> [1]
9   <stream name="xstr" relrate="2.4"/>
10 </configuration>

```

- [1] Line 8: This element identifies the configuration as using the secondary DAQ system. The `readout` attribute gives the list of crates to be read out. The `type` attribute is a string which is passed through to the SDAQ supervisor.

10.4 Run modes

There are numerous ways in which the primary and secondary data acquisition systems may be combined in a given trigger configuration. The primary DAQ may be used in one of three modes:

- None: Primary DAQ is not used.
- FW-Only: The trigger framework generates level 1 accepts, but then always generates level 2 rejects. This is indicated by having a `l1trigger` element that contains no `l2trigger` elements. Usually, the exposure group containing this trigger should not be in a `trigdef` element.
- Full: Full use of primary DAQ. In this case, the level 1 framework crate must be read out, and it is automatically added to readout lists. In addition, trigger accepts are sent to the dummy geographic sector “l3wakeup”.

(Note, however, that it is possible to program both “FW-Only” and “Full” triggers in a single configuration.)

The secondary DAQ system may also be used in one of three modes:

- None: Secondary DAQ is not used.
- “Parasitic:” Secondary DAQ is being driven by the trigger framework. This may have been programmed either in the same configuration or in a different configuration.
- Full: The secondary DAQ system generates triggers itself.

There are seven legal combinations of these modes, enumerated below. Examples of configurations for all these modes are given in Appendix B.

- External: Neither PDAQ nor SDAQ is used. This can be used for the case where data are being fed into the collector/router from an external source. One should probably specify an explicit (large) stream ID for such configurations.
- FW-Only: PDAQ in FW-Only mode, no SDAQ. Here, we generate L1 accepts but always force L2 rejects. This is set up by including a `l1trigger` element that contains no `l2trigger` elements. The `expogroup` containing this element should not be in a `trigdef` element. This may be useful for tasks like standalone crate testing.

- PDAQ: Full PDAQ, no SDAQ. Normal primary DAQ running.
- Parasitic-SDAQ: No PDAQ, parasitic SDAQ. This type of configuration contains a `sdaq` element, but no primary DAQ definitions. The `parasitic` attribute of the `sdaq` element must be set to “yes”. This can be useful for using SDAQ to parasitically read out monitoring information for another run.
- FW-SDAQ: FW-Only PDAQ, parasitic SDAQ. Here, readout is via SDAQ, triggered by the trigger framework. These configurations should include both a `l1trigger` definition as in the “FW-Only” case and also a `sdaq` element. This is used, for example, for tracking calibration runs.
- PDAQ-SDAQ: Full PDAQ, parasitic SDAQ. Normal primary DAQ running, but also reading out using SDAQ (for example, to read out monitoring information).
- SDAQ: No PDAQ, full SDAQ. Reading out with SDAQ only (no trigger framework involvement).

The other two combinations are not allowed.

10.5 Generic Element Reference

This section describes the “generic” elements used to describe the trigger configuration — that is, the elements which do not depend on the available resources. The other elements types (for device types and level 1 term types) are described in the following sections.

10.5.1 Element calibration

ATTRIBUTES

Name	Type	Default
<code>type</code>	CDATA	#REQUIRED
<code>reference</code>	CDATA	"None"
<code>only_streams</code>	CDATA	""
<code>readout</code>	CDATA	#REQUIRED

PARENTS

configuration

CONTENTS

EMPTY

DESCRIPTION

Including this element signals that this configuration is to be used for calibration and the calibration manager should therefore be involved. The `type` attribute is a keyword which is passed through to the calibration manager. *The possible values of this keyword remain to be defined.* The `reference` attribute gives the reference set name to pass to the calibration manager. The `readout` attribute is the list of crates which are to participate in the calibration; it should be a space-separated list of names, each of which is either the name of a `crate_list` or else the name of an individual crate defined in the `download` element. Note that the crates may need to be separately programmed for calibration mode.

If the `only_streams` attribute is given, it should be a space-separated list of stream names. The calibration manager will be told only about the streams given in this list.

10.5.2 Element client

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	#REQUIRED
<code>run_number</code>	CDATA	#REQUIRED
<code>recording</code>	(yes no)	#REQUIRED

PARENTS

clients

CONTENTS

(configuration*)

DESCRIPTION

This element is not normally used for trigger configurations, but it is used in the state output COOR produces at the start of a run.

This describes a single client, named **name** (usually the username and hostname where the client is running). The number of the run this client presently has in process is **run_number**; this is 0 if this client does not have a run in progress. The attribute **recording** tells whether this client has recording turned on.

The contents of this element is the configuration which this client has requested.

10.5.3 Element clients

ATTRIBUTES

Name	Type	Default
		(None.)

PARENTS

None.

CONTENTS

(client*)

DESCRIPTION

This element is not normally used for trigger configurations, but it is used in the state output COOR produces at the start of a run.

This is the top-level element in the state output; it contains a list of the active clients.

10.5.4 Element configuration

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
version	CDATA	"0"
autopause	(yes no)	"no"
physics	(yes no)	"no"
type	CDATA	"test"
comics_runtype	CDATA	"data"
stream_scheme	CDATA	" "

PARENTS

client

CONTENTS

```
((download|crate_list)*,  
l1refsets?,  
(level2?,trigdef+)?,  
expogroup*,  
sdaq?,  
calibration?,  
stream*)
```

DESCRIPTION

This is the top-level element for a trigger description. The attributes `name` and `version` serve to identify this configuration; the name by which COOR knows the configuration is formed by joining together the name and version, separated by a dash. If the configuration is being read from a file, the name of the file must match this name.

The `type` attribute gives the configuration type: `global` for normal physics running, `special` for special physics running, and `test` for diagnostic (non-physics) running. *The precise meaning*

of these types remains to be defined. The set may also be enlarged later.

If `autopause` is `yes`, then the run will automatically be paused if a fatal alarm occurs. (*Once this is implemented in the alarm system.*)

The `physics` attribute tags this configuration as being a primary physics configuration, for purposes of recording luminosity, downtime, etc. Runs tagged as ‘physics’ get the following special treatment:

- Pause/resume transitions are recorded in the logbook.
- A recorded physics run cannot be started unless a store is in progress.
- Only one recorded physics run may be in progress at any one time.
- `TAKER` will ask for confirmation before starting a physics run with recording off.
- `TAKER` will ask for confirmation before starting a physics run with no prescale set loaded.
- When a store ends, recorded physics runs will be automatically stopped (as in `force_stop`).

The attribute `comics_runtype` gives a default to supply for the `runtype` parameter for all `COMICS` devices that have that attribute.

The attribute `stream_scheme` gives the stream scheme name, to be passed through to the database.

The contents of this element start with `download`, in which the crate downloads are given, followed by a set of `crate_list` elements, giving names to lists of crates for readout.

For a configuration using primary DAQ, the `level1` reference sets and level 2 resources may be specified with the `l1refsets` and `level2` elements, followed by a set of `trigdef` elements, one for each level 3 node group being used.

For framework-only configurations, `expogroup` elements may appear at this level.

For configurations using secondary DAQ, a `sdaq` element should be included. A `l1refsets` element may also be specified, but it is ignored unless the trigger framework is being used.

If neither a `trigdef` nor a `sdaq` element is present the logger will be told to receive data, but no one will be told to generate it. This may be useful for recording data from sources outside of COOR's control, such as Sidet.

If a `calibration` element is present, the calibration manager will be told to handle streams for this run.

Finally, a set of `stream` elements define the recording streams.

10.5.5 Element `crate_list`

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	#REQUIRED

PARENTS

`configuration`

CONTENTS

(`crateref*`)

DESCRIPTION

The `crate_list` element provides a way of associating a symbolic name (`name`) with a set of readout crates. This name can then be referenced from a `readout` attribute.

This element contains a set of `crateref` elements, each of which can reference either another `crate_list` or an individual readout crate. The final list is the union of all of the `crateref` elements. Note that ordering does not matter, and duplicates are ignored. Also, recursive `crate_list` elements are not allowed.

Duplicate `crate_list` elements are allowed, as long as their contents are identical. A crate list may also be defined by supplying the `name` attribute to a `download` element.

10.5.6 Element `craterref`

ATTRIBUTES

Name	Type	Default
<code>ref</code>	CDATA	#REQUIRED

PARENTS

`crate_list`

CONTENTS

EMPTY

DESCRIPTION

These elements are used inside of `crate_list` elements to refer to readout crates or other crate lists. The `name` attribute should match either the name of a crate defined inside the `download` element or the name of a `crate_list`.

10.5.7 Element `download`

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	""

PARENTS

`configuration`

CONTENTS

(Device type elements)

DESCRIPTION

The `download` element contains a description of the EPICS settings to be made for this configuration (excluding any made implicitly as part of the trigger configuration). It contains an element for each device to be downloaded. These element types are drawn from the set of known device types. These are described in Sec. 10.6.

If the `name` attribute is defined, then this element also defines a crate list with the given name containing the contents (as would be done with a `crate_list` element. In this case, all the elements within this download should refer to crates.

10.5.8 Element `expogroup`

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	#REQUIRED
<code>number</code>	CDATA	""
<code>readout</code>	CDATA	#REQUIRED
<code>other_gs</code>	CDATA	""

PARENTS

`trigdef,configuration`

CONTENTS

`(11term1ist?,11trigger*)`

DESCRIPTION

This element defines a level 1 exposure group named `name`. The `readout` attribute should be a space-separated list of names, each of which is either the name of a `crate_list` or else the name of an individual crate defined in the `download` element. This is the set of crates to be read out by all of the level 1 bits associated with this exposure group. If the exposure group contains any

`l2trigger` elements, then the trigger framework crate (“trgfr”) is automatically added to the readout list.

If `other_gs` is specified, it should also refer to a crate list, in the same manner as for `readout`. The trigger framework will be told to send accepts to these crates, but Level 3 will not be told about them. If the exposure group contains any `l2trigger` elements, then the dummy internal sector “l3wakeup” is automatically added to this list.

If the `number` attribute is specified, it gives the number of the exposure group to allocate. Otherwise, COOR will choose one.

The element contents start with a `l1termList` element, which gives the level 1 trigger terms to which this exposure group is sensitive. The term list is followed by a list of `l1trigger` elements, describing the level 1 bits associated with this exposure group.

The `name` attribute may be the same as a previous exposure group. In this case, no new exposure group is allocated (unless an explicit `number` attribute was given). Instead, any new triggers defined here are associated with existing exposure group. The `readout`, `other_gs`, and contained `l1termList` for the new exposure group definition must be the same as for the previous one. This can be useful if trigger bits in a single exposure group must be assigned to different level 3 node groups (and must therefore appear in separate `trigdef` elements).

10.5.9 Element `l1em_refset`

ATTRIBUTES

Name	Type	Default
<code>ownmode</code>	<code>(exclusive shared)</code>	<code>"shared"</code>
<code>name</code>	ID	<code>#REQUIRED</code>
<code>number</code>	CDATA	<code>""</code>

PARENTS

`l1refsets`

CONTENTS

(#PCDATA)

DESCRIPTION

This element defines an EM reference set to the level-1 calorimeter trigger. The contents of this element are sent unmodified to the trigger as the programming for the reference set (except that it is flattened to a single line).

The `name` attribute is an identifier which is used to refer to this reference set in the term definitions.

The `ownmode` attribute gives the ownership mode for this reference set. This can be either `exclusive`, meaning that only this configuration can have this reference set allocated, or `shared`, meaning that another configuration can also allocate this reference set, provided that the requests are compatible.

If the `number` attribute is provided, it specifies the specific reference set which should be allocated. Otherwise, COOR will assign the numbers. (Note that if you specify the number explicitly, it must match that of the corresponding `l1hadveto_refset`.)

10.5.10 Element `l1hadveto_refset`

ATTRIBUTES

Name	Type	Default
<code>ownmode</code>	<code>(exclusive shared)</code>	<code>"shared"</code>
<code>name</code>	ID	<code>#REQUIRED</code>
<code>number</code>	CDATA	<code>""</code>

PARENTS

`l1refsets`

CONTENTS

(#PCDATA)

DESCRIPTION

This element defines an hadronic veto reference set to the level-1 calorimeter trigger. The contents of this element are sent unmodified to the trigger as the programming for the reference set (except that it is flattened to a single line).

The `name` attribute is an identifier which is used to refer to this reference set in the term definitions.

The `ownmode` attribute gives the ownership mode for this reference set. This can be either `exclusive`, meaning that only this configuration can have this reference set allocated, or `shared`, meaning that another configuration can also allocate this reference set, provided that the requests are compatible.

If the `number` attribute is provided, it specifies the specific reference set which should be allocated. Otherwise, COOR will assign the numbers. (Note that if you specify the number explicitly, it must match that of the corresponding `l1em_refset`.)

10.5.11 Element `l1jet_refset`

ATTRIBUTES

Name	Type	Default
<code>ownmode</code>	<code>(exclusive shared)</code>	<code>"shared"</code>
<code>name</code>	ID	<code>#REQUIRED</code>
<code>number</code>	CDATA	<code>""</code>

PARENTS

`l1refsets`

CONTENTS

`(#PCDATA)`

DESCRIPTION

This element defines a jet (total E_T) reference set to the level-1 calorimeter trigger. The contents of this element are sent unmodified to the trigger as the programming for the reference set (except that it is flattened to a single line).

The `name` attribute is an identifier which is used to refer to this reference set in the term definitions.

The `ownmode` attribute gives the ownership mode for this reference set. This can be either `exclusive`, meaning that only this configuration can have this reference set allocated, or `shared`, meaning that another configuration can also allocate this reference set, provided that the requests are compatible.

If the `number` attribute is provided, it specifies the specific reference set which should be allocated. Otherwise, COOR will assign the numbers.

10.5.12 Element `l1lt_refset`

ATTRIBUTES

Name	Type	Default
<code>ownmode</code>	<code>(exclusive shared)</code>	<code>"shared"</code>
<code>name</code>	ID	#REQUIRED
<code>number</code>	CDATA	<code>""</code>

PARENTS

`l1refsets`

CONTENTS

`(#PCDATA)`

DESCRIPTION

This element defines a large tile reference set to the level-1 calorimeter trigger. The contents of this element are sent unmodified

to the trigger as the programming for the reference set (except that it is flattened to a single line).

The `name` attribute is an identifier which is used to refer to this reference set in the term definitions.

The `ownmode` attribute gives the ownership mode for this reference set. This can be either `exclusive`, meaning that only this configuration can have this reference set allocated, or `shared`, meaning that another configuration can also allocate this reference set, provided that the requests are compatible.

If the `number` attribute is provided, it specifies the specific reference set which should be allocated. Otherwise, COOR will assign the numbers.

10.5.13 Element `l1refsets`

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

configuration

CONTENTS

```
((l1em_refset|l1hadveto_refset|  
l1jet_refset|l1lt_refset)*)
```

DESCRIPTION

This element collects together definitions for level 1 reference sets.

10.5.14 Element `l1termlist`

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

expogroup, l1trigger, sdaq_l1trigger

CONTENTS

(Level 1 term elements)

DESCRIPTION

The `l1termlist` element contains a set of level 1 trigger term elements. The element types are drawn from the set of known level 1 term types; these are described in Sec. 10.7. Each of these elements contains a `require` element, which can be either `require` or `veto`. It is an error for a term to appear with both `require` and `veto`.

Every term list will always require the “`always_on`” term, and will always veto on the “`skip_next_n_0`” term.

10.5.15 Element `l1trigger`

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	<code>#REQUIRED</code>
<code>number</code>	CDATA	<code>"</code>
<code>prescale</code>	CDATA	<code>"1"</code>
<code>l2_unbiased_ratio</code>	CDATA	<code>"16777216"</code>
<code>obey_feb</code>	(yes no)	<code>"yes"</code>
<code>auto_disabled</code>	(yes no)	<code>"no"</code>
<code>l1_qualifiers</code>	CDATA	<code>"0"</code>

PARENTS

expogroup

CONTENTS

(`l1termlist`, `l2trigger*`)

DESCRIPTION

This element defines a level 1 trigger bit named **name**. The **prescale** attribute gives the initial prescale value for this bit. If the value of this attribute ends with a percent sign ('%'), then this is considered a percentage prescale; the value in front of the percent sign should be an integer in the range 0–100. Otherwise, the value should be a non-negative integer. If it is non-zero, then it is interpreted as a ratio; i.e., pass pass 1-of-*n* events). Specifying 0 for the prescale forces the trigger to always be disabled (until the prescale is changed). If **obey_feb** is **no**, this bit will not be disabled by a front-end busy assertion. If **auto_disabled** is **yes**, this bit will be automatically disabled every time it fires, until explicitly reset (one-shot mode). The **l1_qualifiers** attribute is a space-separated list of level 1 qualifier names (as specified by **l1qual** elements in the resource file). Alternatively, it may be an integer, in which case it is interpreted directly as the qualifier bit mask.

The attribute **l2_unbiased_ratio** gives a value for the Level 2 unbiased sample ratio for this trigger bit. This feature controls the assertion of the L2 unbiased sample L1 qualifier flag for some fraction of the events. (There is only one such L1 qualifier flag common to all 128 trigger bits.) This feature is implemented with a 24-bit counter, giving an allowed value range of $1-2^{24} = 16,777,216$. When a ratio of *N* is programmed, the counter is initialized with a random number between 0 and *N* – 1. The counter is decremented every time this trigger bit fires. The L2 unbiased sample L1 qualifier flag will be set for the one event where this counter reaches the value zero. A ratio of 1 corresponds to asserting the qualifier for every event for which this trigger bit fires. This feature cannot be disabled; instead, set the ratio to its maximum value.

If the **number** attribute is specified, it gives the number of the trigger bit to allocate. Otherwise, COOR will choose one.

The first child of this element is a **l1termlist** element, giving the level 1 term requirements for this bit. This list must be a superset of the term list given in this bit's exposure group. This

is followed by a set of `l2trigger` elements to define the level 2 trigger bits associated with this level 1 bit. If no `l2trigger` elements are included, then this trigger bit will be programmed to always generate level 2 rejects.

10.5.16 Element `l2filter`

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	#REQUIRED
<code>count</code>	CDATA	'1'

PARENTS

`l2script`

CONTENTS

EMPTY

DESCRIPTION

This element defines a single filter to be used in a level 2 script. The `name` attribute is the name of a tool/filter defined in the `level2` section of the configuration. The `count` attribute is the number of objects to require to pass that tool/filter.

10.5.17 Element `l2global`

ATTRIBUTES

Name	Type	Default
<code>USEL1BITS</code>	CDATA	"0"

PARENTS

`level2`

CONTENTS

EMPTY

DESCRIPTION

This element can be used to specify some global parameters for the level 2 system.

Documentation pending hearing from l2 folks about exactly what attributes this should have...

10.5.18 Element `l2script`

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

`l2bit`

CONTENTS

`(#PCDATA|l2filter)*`

DESCRIPTION

This element defines the filters for a single level 2 trigger. The contents consist of a list of `l2filter` elements; these requirements must all be satisfied for the trigger to fire. (A script with no filters always passes.)

[Previous versions required writing the script text explicitly here. In this version, that syntax is still accepted, but it is deprecated, and will be removed in a future version.]

10.5.19 Element `l2trigger`

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	<code>#REQUIRED</code>
<code>number</code>	CDATA	<code>""</code>

PARENTS

11trigger

CONTENTS

(12script?,13trigger*)

DESCRIPTION

This element defines a level 2 trigger bit named **name**. It contains a set of 13**trigger** elements defining the level 3 triggers associated with this level 2 bit.

If the **number** attribute is specified, it gives the number of the trigger bit to allocate. Otherwise, COOR will choose one.

10.5.20 Element 13trigger

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
number	CDATA	""

PARENTS

12trigger

CONTENTS

EMPTY

DESCRIPTION

This element defines a level 3 trigger bit named **name**. Note that all this element does is tell COOR that there is a level 3 bit called **name**. The actual programming for the bit is embedded in the trigger list in the **triglist** element.

If the **number** attribute is specified, it gives the number of the trigger bit to allocate. Otherwise, COOR will choose one.

10.5.21 Element `level2`

ATTRIBUTES

Name	Type	Default
(None)		

PARENTS

`configuration`

CONTENTS

(`l2global?`,
(*Level 2 preprocessor elements, Level 2 tool elements*))

DESCRIPTION

This element contains level-2 information not associated with a particular trigger bit, including tool and filter definitions.

The attributes of this element are deprecated, and should not be specified in new trigger configurations. They will be removed in a future version.

This element starts with an optional `l2global` element, which specifies global parameters affecting the level 2 system. It then contains a list of elements requesting level 2 preprocessors. These elements are described in Sec. 10.8. Note that for each preprocessor referenced, the corresponding readout crate must have been allocated.

Following the preprocessor definitions is a list of level 2 tool and filter definitions. These elements are described in Sec. 10.9.

10.5.22 Element `sdaq`

ATTRIBUTES

Name	Type	Default
<code>type</code>	CDATA	#REQUIRED
<code>only_streams</code>	CDATA	""
<code>readout</code>	CDATA	#REQUIRED

PARENTS

configuration

CONTENTS

((sdaq_l1trigger)?)

DESCRIPTION

This element is used in configurations using the secondary DAQ system. The `readout` attribute is the list of crates which should be read out; it should be a space-separated list of names, each of which is either the name of a `crate_list` or else the name of an individual crate defined in the `download` element. Note that the crates may need to be separately programmed in the `download` section to expect SDAQ readout.

The `type` attribute is a string passed to the SDAQ supervisor. *The possible values of this remain to be defined.*

If the `only_streams` attribute is given, it should be a space-separated list of stream names. The SDAQ manager will be told only about the streams given in this list.

The `sdaq_l1trigger` element was previously used to specify the trigger framework programming for a configuration where the framework generates triggers for SDAQ. This usage is now deprecated; use a separate `expogroup` element instead.

10.5.23 Element `sdaq_l1trigger`

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	#REQUIRED
<code>number</code>	CDATA	""
<code>prescale</code>	CDATA	"1"
<code>l2_unbiased_ratio</code>	CDATA	"16777216"
<code>obey_feb</code>	(yes no)	"yes"
<code>auto_disabled</code>	(yes no)	"no"
<code>l1_qualifiers</code>	CDATA	"0"
<code>expogroup_number</code>	CDATA	""

PARENTS

sdaq

CONTENTS

(11term1ist?)

DESCRIPTION

The use of this element is now deprecated. A separate `expogroup` element should be used instead.

This element defines a level 1 trigger bit named `name`, to be used for triggering SDAQ runs. The `prescale` attribute gives the initial prescale value for this bit. If the value of this attribute ends with a percent sign (`%`), then this is considered a percentage prescale; the value in front of the percent sign should be an integer in the range 1–100. If the value is 0, then this trigger will be disabled. Otherwise, the value should be a positive integer. In this case, the prescale is interpreted as a ratio; i.e., pass pass 1-of- n events). If `obey_feb` is `no`, this bit will not be disabled by a front-end busy assertion. If `auto_disabled` is `yes`, this bit will be automatically disabled every time it fires, until explicitly reset (one-shot mode). The `11_qualifiers` attribute is a space-separated list of level 1 qualifier names (as specified by `11qual` elements in the resource file). Alternatively, it may be an integer, in which case it is interpreted directly as the qualifier bit mask.

The attribute `12_unbiased_ratio` gives a value for the Level 2 unbiased sample ratio for this trigger bit. This feature controls the assertion of the L2 unbiased sample L1 qualifier flag for some fraction of the events. (There is only one such L1 qualifier flag common to all 128 trigger bits.) This feature is implemented with a 24-bit counter, giving an allowed value range of $1-2^{24} = 16,777,216$. When a ratio of N is programmed, the counter is initialized with a random number between 0 and $N - 1$. The counter is decremented every time this trigger bit fires. The L2 unbiased sample L1 qualifier flag will be set for the one event where this counter reaches the value zero. A ratio of 1 corresponds

to asserting the qualifier for every event for which this trigger bit fires. This feature cannot be disabled; instead, set the ratio to its maximum value.

If the `number` attribute is specified, it gives the number of the trigger bit to allocate. Otherwise, COOR will choose one.

The corresponding exposure group is allocated automatically. If the `expogroup_number` attribute is specified, it gives the number of the exposure group to allocate. Otherwise, COOR will choose one.

The child of this element is a `l1termlist` element, giving the level 1 term requirements for this bit.

10.5.24 Element stream

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	#REQUIRED
<code>number</code>	CDATA	""
<code>family</code>	CDATA	"default"
<code>relrate</code>	CDATA	"1.0"

PARENTS

`configuration`

CONTENTS

EMPTY

DESCRIPTION

This element defines a data logging stream named `name`, associated with file family `family`. The `relrate` attribute is a floating point number giving the expected data rate to this stream, relative to an (*at this point*) arbitrary normalization. This information may be used by the data logging system for load balancing.

Note that this element declares that stream `name` exists, but the information about what gets sent to the stream is embedded in the trigger list in the `triglist` element.

To specify that this stream should use a specific stream ID, use the `number` attribute. This should not normally be used (COOR assigns stream IDs automatically). One case where it may be useful is to instruct the data logging system to receive data from a source not under the control of COOR (Sidet, for example).

10.5.25 Element `trigdef`

ATTRIBUTES

Name	Type	Default
<code>l3type</code>	CDATA	'REGULAR'
<code>num_nodes</code>	CDATA	'0'

PARENTS

`configuration`

CONTENTS

(`expogroup*`, `triglist?`)

DESCRIPTION

This element is used in configurations using the primary DAQ system. There should be one of these for every group of level 3 nodes being used.

The attribute `l3type` may be used to request a non-standard level 3 node type; the default of `REGULAR` should be appropriate for normal running. The attribute `num_nodes` can be used to request that a group of level 3 nodes be exclusively allocated to this configuration. The default of 0 means that all otherwise unassigned nodes will be used (and shared with other configurations).

These elements contain the lists of level 1 exposure groups (which in turn contain the trigger definitions), and then the level 3 trigger

list. The level1 reference set and level 2 definitions should precede the trigdef elements.

10.5.26 Element `triglist`

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

`trigdef`

CONTENTS

(#PCDATA)

DESCRIPTION

This is the level 3 trigger list. The contents of this element are passed through to level 3 uninterpreted. It is expected that this will refer to objects such as level 2 and level 3 bits and streams by name; level 3 should use the additional information it has from `coor` about the name to number mappings of these objects to complete the programming.

Level 3 should understand a blank trigger list as meaning “pass everything.” If this element is omitted, it is equivalent to a blank string.

Be careful about characters which are special to XML. It may be useful to put this string inside a CDATA construction.

10.6 Device Type Reference

This section describes the device type elements. These are the elements that can be used inside a `download` element.

All device type elements have three common attributes. The `name` attribute is the name of the device requested. It is an error if the device requested doesn't exist, or if it's actual type does not match the element

type used. The `ownmode` attribute gives the ownership mode for this device. This can be either `exclusive`, meaning that only this configuration can have this device allocated, or `shared`, meaning that another configuration can also allocate this device, provided that the requests are compatible. Note, however, that you cannot modify a device if it has been allocated by more than one configuration.

The third common attribute is `inhibit`. If this is set to `on`, then this device will not actually be downloaded. This is intended for commissioning purposes.

For crate devices, if an attribute value contains a `'` character, then it will be treated as a printf-style string and the geographic sector number will be substituted.

10.6.1 Element Calpulser

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	<code>#REQUIRED</code>
<code>ownmode</code>	<code>(exclusive shared)</code>	<code>"shared"</code>
<code>inhibit</code>	<code>(yes no)</code>	<code>"no"</code>
<code>pattern</code>	CDATA	<code>'off'</code>

PARENTS

`download`

CONTENTS

EMPTY

DESCRIPTION

A calorimeter pulser. The pattern to load into the pulser is given by `pattern` (6 32-bit words). If `pattern` is the special string `'off'`, the pulser will be turned off.

Defined devices of this type:

<u>Name</u>
cal_cccp_pls00
cal_cccp_pls01
cal_cccp_pls02
cal_cccp_pls03
cal_cccp_pls04
cal_cccp_pls05
cal_cccp_pls06
cal_cccp_pls07
cal_cccp_pls08
cal_cccp_pls09
cal_cccp_pls10
cal_cccp_pls11
cal_cccp_pls12

10.6.2 Element Cal_ADC_Crate

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
ownmode	(exclusive shared)	"shared"
inhibit	(yes no)	"no"
runtype	CDATA	" "
vbdtype	CDATA	"NONE"
blsmode	CDATA	"NONE"
adcmode	CDATA	"NONE"
cccatype	CDATA	"NONE"
cccttype	CDATA	"NONE"
pulsetype	CDATA	"NONE"
cccptype	CDATA	"NONE"
pedtype	CDATA	"NONE"
adcctype	CDATA	"NONE"
ccctmode	CDATA	"NONE"
ccctdiag	CDATA	"NONE"

PARENTS

download

CONTENTS

EMPTY

DESCRIPTION

A calorimeter ADC crate. The `runtype` attribute specifies the run type string. If this attribute is not specified, it defaults to the value of the `comics_runtype` attribute of the `configuration` element.

Despite being asked, the calorimeter group has not bothered to provide any documentation or explanation as to what the other parameters mean.

Note: In the current version, `inhibit` defaults to `on`. This will be changed once COMICS settles down.

Defined crates of this type:

Name	ID (decimal)	ID (hex)	Location
<code>ecnnw</code>	64	40	M312-3
<code>ecnsw</code>	65	41	M312-2
<code>ccnw</code>	66	42	M312-1
<code>ccsw</code>	67	43	M306-1
<code>ecsnw</code>	68	44	M306-2
<code>ecssw</code>	69	45	M306-3
<code>ecsse</code>	70	46	M308-3
<code>ecsne</code>	71	47	M308-2
<code>ccse</code>	72	48	M308-1
<code>ccne</code>	73	49	M310-1
<code>ecnse</code>	74	4a	M310-2
<code>ecnne</code>	75	4b	M310-3

10.6.3 Element `Cal_TC_Crate`

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
ownmode	(exclusive shared)	"shared"
inhibit	(yes no)	"no"
runtype	CDATA	" "

PARENTS

download

CONTENTS

EMPTY

DESCRIPTION

The calorimeter timing and control crate. The `runtype` attribute specifies the run type string. If this attribute is not specified, it defaults to the value of the `comics_runtype` attribute of the `configuration` element.

The calorimeter group added some other parameters. But, despite being explicitly asked, they have not provided any documentation of explanation as to what they are or what they mean.

Note: In the current version, `inhibit` defaults to on. This will be changed once COMICS settles down.

Defined crates of this type:

Name	ID (decimal)	ID (hex)	Location
caltc	76	4c	M310-0

10.6.4 Element CFT_Crate

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
ownmode	(exclusive shared)	"shared"
inhibit	(yes no)	"no"
runtype	CDATA	" "

PARENTS

download

CONTENTS

EMPTY

DESCRIPTION

A fiber tracker readout crate. The `runtype` attribute specifies the run type string. If this attribute is not specified, it defaults to the value of the `comics_runtype` attribute of the `configuration` element.

Note: In the current version, `inhibit` defaults to on. This will be changed once COMICS settles down.

Defined crates of this type:

Name	ID (decimal)	ID (hex)	Location
<code>cftax</code>	80	50	M212-0
<code>cftst</code>	81	51	M212-1
<code>cps</code>	82	52	M213-0
<code>fps</code>	83	53	M213-1

10.6.5 Element L1_Crate

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	#REQUIRED
<code>ownmode</code>	(exclusive shared)	"shared"
<code>inhibit</code>	(yes no)	"no"

PARENTS

download

CONTENTS

EMPTY

DESCRIPTION

For level 1. *This is a placeholder until someone tells me what needs to be downloaded here.*

Defined crates of this type:

Name	ID (decimal)	ID (hex)	Location
11cal	16	10	M101-0
11lum	17	11	M115-0
11fpd	18	12	M115-1
11ctt	19	13	M322-2
11ctm	20	14	M321-2
11muc	22	16	M321-0
11mun	23	17	M321-1
11mus	24	18	M321-1
11mtm	25	19	M322-1

10.6.6 Element L2_Crate

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
ownmode	(exclusive shared)	"shared"
inhibit	(yes no)	"no"

PARENTS

download

CONTENTS

EMPTY

DESCRIPTION

For level 2. *This is a placeholder until someone tells me what needs to be downloaded here.*

Defined crates of this type:

Name	ID (decimal)	ID (hex)	Location
12glb	32	20	M121-1
12muc	33	21	M324-0
12muf	34	22	M324-1
12cal	35	23	M121-2
12ps	36	24	M200-2
12ctt	37	25	M200-1

10.6.7 Element Muo_Crate

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
ownmode	(exclusive shared)	"shared"
inhibit	(yes no)	"no"
runtype	CDATA	" "

PARENTS

download

CONTENTS

EMPTY

DESCRIPTION

A muon readout crate. The `runtype` attribute specifies the run type string. If this attribute is not specified, it defaults to the value of the `comics_runtype` attribute of the `configuration` element.

Note: In the current version, `inhibit` defaults to on. This will be changed once COMICS settles down.

Defined crates of this type:

Name	ID (decimal)	ID (hex)	Location
fmsm	48	30	M314-0
fmm	49	31	M314-1

Name	ID (decimal)	ID (hex)	Location
fms s	50	32	M316-0
fmns	51	33	M316-1
cmwtp	52	34	M317-0
cmwsp	53	35	M317-1
cmwbp	54	36	M317-2
cmwsc	55	37	M319-0
cmetp	56	38	M318-0
cmesp	57	39	M318-1
cmebp	58	3a	M318-2
cmesc	59	3b	M319-1

10.6.8 Element Null_Device

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
ownmode	(exclusive shared)	"shared"
inhibit	(yes no)	"no"

PARENTS

download

CONTENTS

EMPTY

DESCRIPTION

A device type which does not do any downloading. Mainly for testing.

Defined crates of this type:

Name	ID (decimal)	ID (hex)	Location
seq0	0	00	PC03-0
seq1	1	01	PC03-1
seq2	2	02	PC04-0

Name	ID (decimal)	ID (hex)	Location
seq3	3	03	PC04-1
seq4	4	04	PC19-0
seq5	5	05	PC19-1
seq6	6	06	PC20-0
seq7	7	07	PC20-1
seq8	8	08	???
calt	9	09	DAB1
12ta	10	0a	FCH2
12tb	11	0b	FCH2
12tc	12	0c	FCH2
12td	13	0d	FCH2

Defined devices of this type:

Name
test

10.6.9 Element SMT_Crate

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
ownmode	(exclusive shared)	"shared"
inhibit	(yes no)	"no"
runtype	CDATA	""

PARENTS

download

CONTENTS

EMPTY

DESCRIPTION

A silicon tracker readout crate. The `runtype` attribute specifies the run type string. If this attribute is not specified, it defaults to the value of the `comics_runtype` attribute of the `configuration` element.

Note: In the current version, `inhibit` defaults to `on`. This will be changed once COMICS settles down.

Defined crates of this type:

Name	ID (decimal)	ID (hex)	Location
smt0_0	96	60	M205-0
smt0_1	97	61	M205-1
smt1_0	98	62	M206-0
smt1_1	99	63	M206-1
smt2_0	100	64	M208-0
smt2_1	101	65	M208-1
smt3_0	102	66	M209-0
smt3_1	103	67	M209-1
smt4_0	104	68	M210-0
smt4_1	105	69	M210-1
smt5_0	106	6a	M211-0
smt5_1	107	6b	M211-2

10.6.10 Element STT_Crate

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	<code>#REQUIRED</code>
<code>ownmode</code>	<code>(exclusive shared)</code>	<code>"shared"</code>
<code>inhibit</code>	<code>(yes no)</code>	<code>"no"</code>

PARENTS

download

CONTENTS

EMPTY

DESCRIPTION

For the silicon tracker trigger crates. *This is a placeholder until someone tells me what needs to be downloaded here.*

Defined crates of this type:

Name	ID (decimal)	ID (hex)	Location
stt0	112	70	M202-0
stt1	113	71	M202-1
stt2	114	72	M203-0
stt3	115	73	M203-1
stt4	116	74	M204-0
stt5	117	75	M204-1

10.6.11 Element Trig_Crate

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
ownmode	(exclusive shared)	"shared"
inhibit	(yes no)	"no"

PARENTS

download

CONTENTS

EMPTY

DESCRIPTION

For the trigger framework. *This is a placeholder until someone tells me what needs to be downloaded here.*

Defined crates of this type:

Name	ID (decimal)	ID (hex)	Location
trgfr	31	1f	M124

10.7 Level 1 Trigger Term Reference

This section describes the level 1 trigger term elements. These are the elements which can be used inside a `l1termlist` element.

All term elements have two common attributes. The `require` attribute, which can have a value of either `require` or `veto`, tells whether this term is to be required or rejected. The `ownmode` attribute gives the ownership mode for this term. This can be either `exclusive`, meaning that only this configuration can have this term allocated, or `shared`, meaning that another configuration can also allocate this term, provided that the requests are compatible.

This will be expanded later.

10.7.1 Element `l1ctt`

ATTRIBUTES

Name	Type	Default
<code>require</code>	<code>(require veto)</code>	<code>"require"</code>
<code>ownmode</code>	<code>(exclusive shared)</code>	<code>"shared"</code>
<code>name</code>	CDATA	<code>#REQUIRED</code>
<code>number</code>	CDATA	<code>" "</code>

PARENTS

`l1termlist`

CONTENTS

EMPTY

DESCRIPTION

This element references a term from a central tracker trigger manager card. The `name` attribute identifies the particular input term to the manager card which is to be mapped to the output term. *The list of available input terms remains to be documented.*

If the `number` attribute is specified, it gives the number of the term to allocate. Otherwise, COOR will choose one.

10.7.2 Element `l1emcount`

ATTRIBUTES

Name	Type	Default
<code>require</code>	(require veto)	"require"
<code>ownmode</code>	(exclusive shared)	"shared"
<code>number</code>	CDATA	" "
<code>count</code>	CDATA	#REQUIRED
<code>em_refset</code>	IDREF	#REQUIRED
<code>hadveto_refset</code>	IDREF	#REQUIRED

PARENTS

`l1termlist`

CONTENTS

EMPTY

DESCRIPTION

This term is satisfied if there are at least `count` towers with EM E_T above that specified in `em_refset` but with hadronic E_T below that specified in `hadveto_refset`.

The values of the reference set attributes should match names given in earlier `l1em_refset` and `l1hadveto_refset` elements. If those elements specified the `number` attribute, they must match; otherwise, COOR will try to automatically allocate matching reference sets.

If the `number` attribute is specified, it gives the number of the comparator to allocate. Otherwise, COOR will choose one.

10.7.3 Element `l1emcountr`

ATTRIBUTES

Name	Type	Default
require	(require veto)	"require"
ownmode	(exclusive shared)	"shared"
number	CDATA	" "
count	CDATA	#REQUIRED
region	(c n s)	#REQUIRED
em_refset	IDREF	#REQUIRED
hadveto_refset	IDREF	#REQUIRED

PARENTS

11term1ist

CONTENTS

EMPTY

DESCRIPTION

This term is satisfied if there are at least `count` towers in a particular region of the calorimeter with EM E_T above that specified in `em_refset` but with hadronic E_T below that specified in `hadveto_refset`.

The `region` parameter should be one of 'c' (central), 'n' (north), or 's' (south).

The values of the reference set attributes should match names given in earlier `11em_refset` and `11hadveto_refset` elements. If those elements specified the `number` attribute, they must match; otherwise, COOR will try to automatically allocate matching reference sets.

A `number` attribute is present for consistency, but its value is ignored.

10.7.4 Element 11emquad

ATTRIBUTES

Name	Type	Default
require	(require veto)	"require"
ownmode	(exclusive shared)	"shared"
number	CDATA	""
count	CDATA	#REQUIRED
quadrant	(1 2 3 4)	#REQUIRED

PARENTS

11term1ist

CONTENTS

EMPTY

DESCRIPTION

This term is satisfied if there are at least `count` towers in a particular quadrant of the calorimeter with EM E_T above hardwired thresholds.

The `quadrant` parameter should be '1', '2', '3', or '4'.

A `number` attribute is present for consistency, but its value is ignored.

10.7.5 Element 11jetcount

ATTRIBUTES

Name	Type	Default
require	(require veto)	"require"
ownmode	(exclusive shared)	"shared"
number	CDATA	""
count	CDATA	#REQUIRED
jet_refset	IDREF	#REQUIRED

PARENTS

11term1ist

CONTENTS

EMPTY

DESCRIPTION

This term is satisfied if there are at least `count` towers with total E_T above that specified in `refset`.

The value of the reference set attribute should match the name given in an earlier `l1jet_refset` element.

If the `number` attribute is specified, it gives the number of the comparator to allocate. Otherwise, COOR will choose one.

10.7.6 Element `l1jetcountr`

ATTRIBUTES

Name	Type	Default
<code>require</code>	<code>(require veto)</code>	"require"
<code>ownmode</code>	<code>(exclusive shared)</code>	"shared"
<code>number</code>	CDATA	""
<code>count</code>	CDATA	#REQUIRED
<code>region</code>	<code>(c n s)</code>	#REQUIRED
<code>jet_refset</code>	IDREF	#REQUIRED

PARENTS

`l1termlist`

CONTENTS

EMPTY

DESCRIPTION

This term is satisfied if there are at least `count` towers in a particular region of the calorimeter with total E_T above that specified in `refset`.

The `region` parameter should be one of ‘c’ (central), ‘n’ (north), or ‘s’ (south).

The value of the reference set attribute should match the name given in an earlier `l1jet_refset` element.

A `number` attribute is present for consistency, but its value is ignored.

10.7.7 Element `l1jetquad`

ATTRIBUTES

Name	Type	Default
<code>require</code>	(require veto)	"require"
<code>ownmode</code>	(exclusive shared)	"shared"
<code>number</code>	CDATA	""
<code>count</code>	CDATA	#REQUIRED
<code>quadrant</code>	(1 2 3 4)	#REQUIRED

PARENTS

`l1termlist`

CONTENTS

EMPTY

DESCRIPTION

This term is satisfied if there are at least `count` towers in a particular quadrant of the calorimeter with total E_T above hardwired thresholds.

The `quadrant` parameter should be ‘1’, ‘2’, ‘3’, or ‘4’.

A `number` attribute is present for consistency, but its value is ignored.

10.7.8 Element `l1l1tcount`

ATTRIBUTES

Name	Type	Default
<code>require</code>	<code>(require veto)</code>	"require"
<code>ownmode</code>	<code>(exclusive shared)</code>	"shared"
<code>number</code>	CDATA	""
<code>count</code>	CDATA	#REQUIRED
<code>lt_refset</code>	IDREF	#REQUIRED

PARENTS

`l1termlist`

CONTENTS

EMPTY

DESCRIPTION

This term is satisfied if there are at least `count` large tiles with total E_T above that specified in `refset`.

The value of the reference set attribute should match the name given in an earlier `l1l1t_refset` element.

A `number` attribute is present for consistency, but its value is ignored.

10.7.9 Element `l1emetsum`

ATTRIBUTES

Name	Type	Default
<code>require</code>	<code>(require veto)</code>	"require"
<code>ownmode</code>	<code>(exclusive shared)</code>	"shared"
<code>number</code>	CDATA	""
<code>value</code>	CDATA	#REQUIRED

PARENTS

`l1termlist`

CONTENTS

EMPTY

DESCRIPTION

This term is satisfied if the scalar E_T sum over the electromagnetic calorimeter is greater than `value` (a floating-point number).

If the `number` attribute is specified, it gives the number of the term to allocate. Otherwise, COOR will choose one.

10.7.10 Element 11fpd

ATTRIBUTES

Name	Type	Default
<code>require</code>	<code>(require veto)</code>	"require"
<code>ownmode</code>	<code>(exclusive shared)</code>	"shared"
<code>name</code>	CDATA	#REQUIRED
<code>number</code>	CDATA	" "

PARENTS

11termlist

CONTENTS

EMPTY

DESCRIPTION

This element references a term from a forward proton detector trigger manager card. The `name` attribute identifies the particular input term to the manager card which is to be mapped to the output term. *The list of available input terms remains to be documented.*

If the `number` attribute is specified, it gives the number of the term to allocate. Otherwise, COOR will choose one.

10.7.11 Element `l1fps`

ATTRIBUTES

Name	Type	Default
<code>require</code>	<code>(require veto)</code>	"require"
<code>ownmode</code>	<code>(exclusive shared)</code>	"shared"
<code>name</code>	CDATA	#REQUIRED
<code>number</code>	CDATA	""

PARENTS

`l1termlist`

CONTENTS

EMPTY

DESCRIPTION

This element references a term from a forward preshower trigger manager card. The `name` attribute identifies the particular input term to the manager card which is to be mapped to the output term. *The list of available input terms remains to be documented.*

If the `number` attribute is specified, it gives the number of the term to allocate. Otherwise, COOR will choose one.

10.7.12 Element `l1hdetsum`

ATTRIBUTES

Name	Type	Default
<code>require</code>	<code>(require veto)</code>	"require"
<code>ownmode</code>	<code>(exclusive shared)</code>	"shared"
<code>number</code>	CDATA	""
<code>value</code>	CDATA	#REQUIRED

PARENTS

`l1termlist`

CONTENTS

EMPTY

DESCRIPTION

This term is satisfied if the scalar E_T sum over the hadronic calorimeter is greater than `value` (a floating-point number).

If the `number` attribute is specified, it gives the number of the term to allocate. Otherwise, COOR will choose one.

10.7.13 Element `l1lum`

ATTRIBUTES

Name	Type	Default
<code>require</code>	<code>(require veto)</code>	<code>"require"</code>
<code>ownmode</code>	<code>(exclusive shared)</code>	<code>"shared"</code>
<code>name</code>	CDATA	<code>#REQUIRED</code>
<code>number</code>	CDATA	<code>" "</code>

PARENTS

`l1termlist`

CONTENTS

EMPTY

DESCRIPTION

This element references a term from a luminosity trigger manager card. The `name` attribute identifies the particular input term to the manager card which is to be mapped to the output term. *The list of available input terms remains to be documented.*

If the `number` attribute is specified, it gives the number of the term to allocate. Otherwise, COOR will choose one.

10.7.14 Element `l1misspt`

ATTRIBUTES

Name	Type	Default
<code>require</code>	<code>(require veto)</code>	<code>"require"</code>
<code>ownmode</code>	<code>(exclusive shared)</code>	<code>"shared"</code>
<code>number</code>	CDATA	<code>""</code>
<code>value</code>	CDATA	<code>#REQUIRED</code>

PARENTS

`l1termlist`

CONTENTS

EMPTY

DESCRIPTION

This term is satisfied if the vector E_T sum over the entire calorimeter is greater than `value` (a floating-point number).

If the `number` attribute is specified, it gives the number of the term to allocate. Otherwise, COOR will choose one.

10.7.15 Element `l1muo`

ATTRIBUTES

Name	Type	Default
<code>require</code>	<code>(require veto)</code>	<code>"require"</code>
<code>ownmode</code>	<code>(exclusive shared)</code>	<code>"shared"</code>
<code>name</code>	CDATA	<code>#REQUIRED</code>
<code>number</code>	CDATA	<code>""</code>

PARENTS

`l1termlist`

CONTENTS

EMPTY

DESCRIPTION

This element references a term from a muon trigger manager card. The `name` attribute identifies the particular input term to the manager card which is to be mapped to the output term. *The list of available input terms remains to be documented.*

If the `number` attribute is specified, it gives the number of the term to allocate. Otherwise, COOR will choose one.

10.7.16 Element `l1specterm`

ATTRIBUTES

Name	Type	Default
<code>require</code>	<code>(require veto)</code>	"require"
<code>ownmode</code>	<code>(exclusive shared)</code>	"shared"
<code>name</code>	CDATA	#REQUIRED

PARENTS

`l1termlist`

CONTENTS

EMPTY

DESCRIPTION

This element references the fixed trigger term `name`. *The list of available terms remains to be documented.*

10.7.17 Element `l1totetsum`

ATTRIBUTES

Name	Type	Default
<code>require</code>	<code>(require veto)</code>	"require"
<code>ownmode</code>	<code>(exclusive shared)</code>	"shared"
<code>number</code>	CDATA	" "
<code>value</code>	CDATA	#REQUIRED

PARENTS

l1termlist

CONTENTS

EMPTY

DESCRIPTION

This term is satisfied if the scalar E_T sum over the entire calorimeter is greater than `value` (a floating-point number).

If the `number` attribute is specified, it gives the number of the term to allocate. Otherwise, COOR will choose one.

10.8 Level 2 Preprocessor Reference

10.8.1 Element `l2calem`

ATTRIBUTES

Name	Type	Default
<code>ieta_min</code>	CDATA (int)	"0"
<code>ieta_max</code>	CDATA (int)	"39"
<code>box_size</code>	CDATA (int)	"3"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Parameters:

- `ieta_min` — Minimum value of `ieta` to search for clusters. Allowable range: 0–160.

- `ieta_max` — Maximum value of `ieta` to search for clusters. Allowable range: 0–160. Must be larger than `ieta_min`.
- `box_size` — Size of the box (for isolation). Must be at least 1.

No further documentation is available.

10.8.2 Element `l2caljet`

ATTRIBUTES

Name	Type	Default
<code>ieta_min</code>	CDATA (int)	"0"
<code>ieta_max</code>	CDATA (int)	"39"
<code>jet_size</code>	CDATA (int)	"3"
<code>keep_corner</code>	CDATA (int)	"1"
<code>keep_row</code>	CDATA (int)	"1"
<code>dclear</code>	CDATA (int)	"1"

PARENTS

`level2`

CONTENTS

EMPTY

DESCRIPTION

Parameters:

- `ieta_min` — Minimum value of `ieta` to search for clusters. Allowable range: 0–160.
- `ieta_max` — Maximum value of `ieta` to search for clusters. Allowable range: 0–160. Must be larger than `ieta_min`.
- `jet_size` — Size of the jet box to use. Must be at least 1.
- `keep_corner` — Keeps jets with overlapping corners. Must be 0 or 1.

- `keep_row` — Keep jets with overlapping rows. Must be 0 or 1.
- `dclear` — Distance needed to ensure jets are separate. Must be at least 1.

No further documentation is available.

10.8.3 Element `12calmet`

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

`level2`

CONTENTS

EMPTY

DESCRIPTION

To be documented.

10.8.4 Element `12cps`

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

`level2`

CONTENTS

EMPTY

DESCRIPTION

To be documented.

10.8.5 Element 12ctt

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

To be documented.

10.8.6 Element 12fps

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

To be documented.

10.8.7 Element 12muc

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

To be documented.

10.8.8 Element 12muf

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

To be documented.

10.9 Level 2 Tool Reference

10.9.1 Element l2commissiontool

ATTRIBUTES

Name	Type	Default
major_version	CDATA (int)	"0"
minor_version	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Makes one dummy object so that the commissioning filters can have an input list of objects.

10.9.2 Element l2emfilter

ATTRIBUTES

Name	Type	Default
emfrac	CDATA (float)	"0."
minet	CDATA (float)	"0."
isofrac	CDATA (float)	"1.0"
tool	CDATA (tool)	#REQUIRED
major_version	CDATA (int)	"0"
minor_version	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Filter which takes a list of electromagnetic objects created by `l2emtool` and applies cuts. These cuts are on the E_T , electromagnetic fraction, and isolation of the objects.

10.9.3 Element `l2emtool`

ATTRIBUTES

Name	Type	Default
<code>cpswindowieta</code>	CDATA (int)	"3"
<code>cpswindowiphi</code>	CDATA (int)	"3"
<code>trackwindowiphi</code>	CDATA (int)	"5"
<code>minet</code>	CDATA (float)	"0."
<code>requiretrack</code>	CDATA (int)	"0"
<code>requirecps</code>	CDATA (int)	"0"
<code>major_version</code>	CDATA (int)	"0"
<code>minor_version</code>	CDATA (int)	"0"

PARENTS

`level2`

CONTENTS

EMPTY

DESCRIPTION

Creates a list of electromagnetic objects based on an input list from the `l2calem` preprocessor, with optional matching to objects from the central tracker or preshower. A cut on the E_T of the object is applied.

10.9.4 Element `l2etafilter`

ATTRIBUTES

Name	Type	Default
nregions	CDATA (int)	"1"
ietamin	CDATA (int)	"0"
ietamax	CDATA (int)	"160"
ietamin2	CDATA (int)	"0"
ietamax2	CDATA (int)	"160"
ietamin3	CDATA (int)	"0"
ietamax3	CDATA (int)	"160"
ietamin4	CDATA (int)	"0"
ietamax4	CDATA (int)	"160"
filter	CDATA (tool)	#REQUIRED
major_version	CDATA (int)	"0"
minor_version	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Takes a list of objects and cuts on their η values. Passes objects which are in the allowed η region.

10.9.5 Element l2etaphisepfilter

ATTRIBUTES

Name	Type	Default
nfilters	CDATA (int)	"1"
ietaminsep	CDATA (int)	"0"
iphiminsep	CDATA (int)	"0"
filter0	CDATA (tool)	#REQUIRED
filter1	CDATA (tool)	#REQUIRED
filter2	CDATA (tool)	#REQUIRED
filter3	CDATA (tool)	#REQUIRED
major_version	CDATA (int)	"0"
minor_version	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Undocumented.

10.9.6 Element `l2etasepfilter`

ATTRIBUTES

Name	Type	Default
<code>ietaminsep</code>	CDATA (int)	"0"
<code>ietamaxsep</code>	CDATA (int)	"0"
<code>nfilters</code>	CDATA (int)	"1"
<code>filter0</code>	CDATA (tool)	#REQUIRED
<code>filter1</code>	CDATA (tool)	#REQUIRED
<code>major_version</code>	CDATA (int)	"0"
<code>minor_version</code>	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Takes two filter lists as input and calculates the η separation between all combinations. It passes both objects of any pair that falls in the allowed η separation window.

10.9.7 Element l2failallfilter

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

This filter always fails.

10.9.8 Element l2htfilter

ATTRIBUTES

Name	Type	Default
htmin	CDATA (float)	"0."
nfilters	CDATA (int)	"1"
filter0	CDATA (tool)	#REQUIRED
filter1	CDATA (tool)	#REQUIRED
filter2	CDATA (tool)	#REQUIRED
major_version	CDATA (int)	"0"
minor_version	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Takes three filter lists as input and calculates the H_T of all the objects. The H_T is the scalar sum of the E_T 's of the objects.

10.9.9 Element `l2invmassfilter`

ATTRIBUTES

Name	Type	Default
<code>mininvmass</code>	CDATA (float)	"0."
<code>maxinvmass</code>	CDATA (float)	"0."
<code>filter0</code>	CDATA (tool)	#REQUIRED
<code>filter1</code>	CDATA (tool)	#REQUIRED
<code>major_version</code>	CDATA (int)	"0"
<code>minor_version</code>	CDATA (int)	"0"

PARENTS

`level2`

CONTENTS

EMPTY

DESCRIPTION

Takes two lists of objects and calculates the invariant masses of all pairs. It passes both objects of any pair that has an invariant mass within the allowed window.

10.9.10 Element `l2jetfilter`

ATTRIBUTES

Name	Type	Default
<code>minet</code>	CDATA (float)	"0."
<code>tool</code>	CDATA (tool)	#REQUIRED
<code>major_version</code>	CDATA (int)	"0"
<code>minor_version</code>	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Takes a list of jet objects created by the `l2jettool` and applies a cut on the E_T of the objects.

10.9.11 Element `l2jetfilter`

ATTRIBUTES

Name	Type	Default
<code>minet</code>	CDATA (float)	"0."
<code>tool</code>	CDATA (tool)	#REQUIRED
<code>major_version</code>	CDATA (int)	"0"
<code>minor_version</code>	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Takes a list of jet objects created by the `l2jettool` and applies a cut on the E_T of the objects.

10.9.12 Element l2metfilter

ATTRIBUTES

Name	Type	Default
minmet	CDATA (float)	"0."
tool	CDATA (tool)	#REQUIRED
major_version	CDATA (int)	"0"
minor_version	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Takes the one object created by the l2mettool and applies a minimum $\#_T$ cut.

10.9.13 Element l2mettool

ATTRIBUTES

Name	Type	Default
major_version	CDATA (int)	"0"
minor_version	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Makes an met object with the E_T from the l2calmet preprocessor.

10.9.14 Element l2muonfilter

ATTRIBUTES

Name	Type	Default
minet	CDATA (float)	"0."
quality	CDATA (int)	"0"
prompt	CDATA (int)	"0"
sign	CDATA (int)	"0"
tool	CDATA (tool)	#REQUIRED
major_version	CDATA (int)	"0"
minor_version	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Takes a list of muon objects created by l2muontool and applies cuts. These cuts are on the quality, sign, "promptness" (time of flight), and E_T of the objects.

10.9.15 Element l2muontool

ATTRIBUTES

Name	Type	Default
l1ptthresh	CDATA (int)	"0"
trackwindowiphi	CDATA (int)	"5"
requiretrack	CDATA (int)	"0"
major_version	CDATA (int)	"0"
minor_version	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Creates a list of muon objects based on an input list from the 12muc and 12muf preprocessors, with an optional match to the central tracker. A cut on the Level 1 E_T of the object is applied.

10.9.16 Element 12phifilter

ATTRIBUTES

Name	Type	Default
iphimin	CDATA (int)	"0"
iphimax	CDATA (int)	"0"
filter	CDATA (tool)	#REQUIRED
major_version	CDATA (int)	"0"
minor_version	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Takes a list of objects and cuts on their ϕ values. Passes objects which are in the allowed ϕ region.

10.9.17 Element l2phisepfilter

ATTRIBUTES

Name	Type	Default
iphiminsep	CDATA (int)	"0"
iphimaxsep	CDATA (int)	"0"
nfilters	CDATA (int)	"1"
filter0	CDATA (tool)	#REQUIRED
filter1	CDATA (tool)	#REQUIRED
major_version	CDATA (int)	"0"
minor_version	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Takes a list of objects as input and calculates the ϕ separation between all combinations. It passes both objects of any pair that falls in the allowed ϕ separation window.

10.9.18 Element l2randompassfilter

ATTRIBUTES

Name	Type	Default
passpercent	CDATA (float)	"1"
tool	CDATA (tool)	#REQUIRED
major_version	CDATA (int)	"0"
minor_version	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

This filter is intended mainly for commissioning. It passes a desired percentage of events using a random number generator. It takes the list of objects generated by `l2commissiontool` as input. This is really just one dummy object.

10.9.19 Element `l2trackfilter`

ATTRIBUTES

Name	Type	Default
<code>minet</code>	CDATA (float)	"0."
<code>tool</code>	CDATA (tool)	#REQUIRED
<code>major_version</code>	CDATA (int)	"0"
<code>minor_version</code>	CDATA (int)	"0"

PARENTS

`level2`

CONTENTS

EMPTY

DESCRIPTION

Takes a list of track objects created by the `l2tracktool` and applies a cut on the object E_T .

10.9.20 Element `l2tracktool`

ATTRIBUTES

Name	Type	Default
<code>minet</code>	CDATA (float)	"0."
<code>major_version</code>	CDATA (int)	"0"
<code>minor_version</code>	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Creates a list of track objects based on an input list from the l2ctt preprocessor. A cut on the E_T of the objects is applied.

10.9.21 Element l2transmassfilter

ATTRIBUTES

Name	Type	Default
mintransmass	CDATA (float)	"0."
filter0	CDATA (tool)	#REQUIRED
filter1	CDATA (tool)	#REQUIRED
major_version	CDATA (int)	"0"
minor_version	CDATA (int)	"0"

PARENTS

level2

CONTENTS

EMPTY

DESCRIPTION

Takes two lists of objects and calculates the transverse mass of all pairs. It passes both objects of any pair that has a transverse mass within the allowed window.

10.10 Prescale Sets

Each trigger configuration may have a collection of “prescale sets” associated with it. A prescale set is a predefined specification of prescale values for a set of Level 1 trigger bits that can be loaded by the user in a single request. This can be used to construct canned prescale configurations to be used, for example, for different luminosities.

The way in which prescale sets are found depends on the loader that was used to load the trigger configuration. At present, only XML-based trigger configurations may have associated prescale sets.

Given a configuration pathname of the form *configname.xml*, any associated prescale sets should be in files with names of the form *configname-prescname.prescales*, where *prescname* is the name of the prescale set. Prescale files may also be put in a subdirectory of the directory where the trigger configuration resides, named *configname-prescales*. The files in this directory should have names of the form *prescname.prescales*.

In the prescale set files, a hash sign (#) introduces a comment; all text from the hash sign to the end of the line is removed. Any lines that are empty after comment removal are ignored. All remaining lines should consist of two fields separated by whitespace: a Level 1 trigger name and a prescale. Prescale values that end with a percent sign are interpreted as percentage prescales. Otherwise, the values should be integers, and are interpreted as ratios. But if the prescale value is 0, then the trigger is disabled.

11 Defining COOR Resources

When COOR starts up, it needs to find out the available detector resources. This is done by reading a XML-based file that defines these resources.

11.1 Resource Definitions

Resource definitions are contained in the file given by the configuration parameter `resource_file` (normally `coor_resources.xml`), in the directory given by the configuration parameter `resource_dir` (normally, the value of the environment variable `COOR_RESOURCES`).

The syntax of the resource definitions is based on XML (see Sec. 10.2). The element types which may be used are described below (Sec. 11.2).

Below is an example of a resource definition file.

```
1 <?xml version="1.0"?> [1]
2 <!DOCTYPE resources SYSTEM "resources.dtd"> [1]
3
4 <resources> [2]
5
6 <devtype name="MUO_Crate" comics_prefix="MUO."> [3]
7   <attribute name="runtype" xmltype="CDATA"/>
8 </devtype> [3]
9
10 <devtype name="SMT_Crate" comics_prefix="SMT.">
11   <attribute name="runtype" xmltype="CDATA"/>
12 </devtype>
13
14 <devtype name="Pulser"> [4]
15   <attribute name="mode" xmltype="(on|off)"
16     default="off" onfree="off"/>
17   <attribute name="pattern" xmltype="CDATA"/>
18 </devtype> [4]
19
20 <devtype name="Null_Device"/> [5]
21
22 <devices> [6]
```

```

23   <device name="pulser1"      type="Pulser">
24     <subdevice comics_name="pattern1" attribs="pattern"/> [7]
25   </device>
26 </devices> [6]
27
28 <crates> [8]
29   <crate name="seq2" type="Null_Device"
30     geosect="0x02" novbd="yes"/> [9]
31
32   <crate name="smt0_0" type="SMT_Crate" geosect="0x60"> [10]
33     <tieto name="seq2"/>
34   </crate> [10]
35
36   <crate name="l2glb" type="Null_Device" geosect="0x20"/>
37   <crate name="l2muc" type="Null_Device" geosect="0x21"/>
38
39   <crate name="cmwtp" type="Comics_Test_Dev" geosect="0x34"/>
40   <crate name="cmwsp" type="Comics_Test_Dev" geosect="0x35"/>
41   <crate name="cmwbp" type="Comics_Test_Dev" geosect="0x36"/>
42   <crate name="cmwsc" type="Comics_Test_Dev" geosect="0x37"/>
43
44   <crate name="cmesc" type="Comics_Test_Dev" geosect="0x3b"/>
45
46   <crate name="trgfr" type="Trig_Crate" geosect="0x1f"/> [11]
47   <crate name="l3wakeup" type="Null_Device"
48     geosect="0x7f" novbd="yes"/> [11]
49
50 </crates> [8]
51
52 <level1 n_expogroups="3" n_bits="5"> [12]
53   <term name="term0" number="0"/> [13]
54   <term name="term1" number="1"/>
55   <term name="term2" number="2"/>
56   <term name="term3" number="3"/> [13]
57
58   <trigmgr class="l1muo" first_term="48" device="muo_mgr2"> [14]
59     <term name="loose_muon" number="2"/>
60     <term name="tight_muon" number="3"/>

```

```

61 </trigmgr> [14]
62
63 <l1ct_thresh class="l1emetsum" first_term="128" n_terms="8"/> [15]
64
65 <term name="always_on" number="255"/> [16]
66
67 <l1qual number="02" name="qual02"/> [17]
68 </level1>
69
70
71 <level2> [18]
72
73 <l2ppcrate crate="l2muc"> [19]
74   <l2input number="1" crates="cmwtp cmwsp cmwsc"/>
75   <l2input number="2" crates="cmwbp cmwsc"/>
76   <l2mbt>
77   L2MUC MUCWORKER PILOTMBT {
78     VMESLOT = 21,      # VME slot number of the card
79     CHAN0 = %1,       # Mapping for chan 0
80     CHAN1 = %2,       # Mapping for chan 1
81     CHAN2 = %3,       # Mapping for chan 2
82     CHAN3 = %4,       # Mapping for chan 3
83     CHAN4 = %5,       # Mapping for chan 4
84     CHAN5 = %6,       # Mapping for chan 5
85     CHAN6 = %4,       # Mapping for chan 6
86     CHAN7 = 0,        # Mapping for the L1 SCL data
87     DISP_CHAN = 7,    # Channel to display on the MBT front panel
88     TEST_SCL = 0,    # Disable testing mode
89     CYCLE_BUFFERS = 1, # cycle the broadcast slot buffers
90     ENABLE_HIST = 1, # Turn on monitoring histograms
91     L1SCL_DATA_TYPE = 0 # Set L1 SCL datatype for the global
92   }
93   </l2mbt>
94 </l2ppcrate> [19]
95
96 <l2ppcrate crate="l2glb" l2cratename="L2GBL">
97   <l2input number="2" crates="l2muc"/>
98   <l2mbt>

```

```

99  L2GBL GBLWORKER PILOTMBT {
100     VMESLOT = 21,      # VME slot number of the card
101     CHAN0 = %1,       # Mapping for chan 0
102     CHAN1 = %2,       # Mapping for chan 1
103     CHAN2 = %3,       # Mapping for chan 2
104     CHAN3 = %4,       # Mapping for chan 3
105     CHAN4 = %5,       # Mapping for chan 4
106     CHAN5 = %6,       # Mapping for chan 5
107     CHAN6 = %4,       # Mapping for chan 6
108     CHAN7 = 0,        # Mapping for the L1 SCL data
109     DISP_CHAN = 7,    # Channel to display on the MBT front panel
110     TEST_SCL = 0,     # Disable testing mode
111     CYCLE_BUFFERS = 1, # cycle the broadcast slot buffers
112     ENABLE_HIST = 1,  # Turn on monitoring histograms
113     L1SCL_DATA_TYPE = 0 # Set L1 SCL datatype for the global
114 }
115 </l2mbt>
116 </l2ppcrate>
117
118 <l2global crate="l2glb" [20]
119     exe_name="GBLWORKER"
120     l2_name="GLOBALWORKER"
121     fail_tooltype="l2fail">
122     <l2parm name="USEL1BITS" type="int" default="0"/>
123 </l2global> [20]
124
125 <l2pp name="l2muc" [21]
126     crate="l2muc" exe_name="MUCWORKER" l2_name="MUCWORKER"
127     l1qual="qual02">
128     <l2parm name="pt" type="float"/>
129 </l2pp> [21]
130
131 <l2tool name="l2muon" pp="l2muc"> [22]
132     <l2parm name="qual" type="int" default="0"/>
133     <l2parm name="pt" type="float"/>
134 </l2tool> [22]
135
136 <l2tool name="l2mucut"> [23]

```

```
137 <l2parm name="pt" type="float"/>
138 <l2parm name="etamax" type="float"/>
139 <l2parm name="tool" type="tool"/>
140 </l2tool> ②
141
142 <l2tool name="l2fail"/>
143
144 </level2>
145
146
147 </resources>
```

- ① Lines 1 and 2: The boilerplate to identify the file as XML and identify the DTD to use.
- ② Line 4: The top-level element of the resource definition file.
- ③ Lines 6 and 8: The first thing we need to do is to define the device types we're going to use, for the devices that get downloaded through COMICS. Each type has a name and a set of attributes, as specified here. One can also associate some additional information with device types; for example, the `comics_prefix` is a string that gets added to the front of the names of devices of this type when the names get sent to COMICS.
- ④ Lines 14 and 18: This shows a slightly more complicated device types definition. For an attribute, you can specify how it should appear in XML, as the `xmltype` attribute. Useful values for this are `CDATA` and enumerations, as shown in the example. You can specify a default with `default`. If the default is missing (or is set to `#REQUIRED`), then there is no default, and the attribute must always be specified. If `onfree` is set to something nonblank, then when the last owner frees this device, a download will be generated to set this attribute to the specified value.
- ⑤ Line 20: And this is a trivial device type, with no attributes.
- ⑥ Lines 22 and 26: These define the devices which are downloadable through COMICS (excluding readout crates). Each such device is iden-

tified by a name and a type; the type must have earlier been defined by a `devtype` element.

- [7] Line 24 declares a subdevice for this device. This says that the attribute `pattern` should be sent to the COMICS device named `pattern1`, rather than to `pulser1`.
- [8] Lines 28 and 50: These define readout crates. Like other downloadable devices, each crate has a name and a type. In addition, each crate has an (integral) geographic sector number.
- [9] Line 30: This defines a sequencer crate. It has no attributes. Further, this crate does not get read out to Level 3; this is indicated by setting the “`novbd`” attribute.
- [10] Lines 32 to 34: This is a definition of a readout crate. Further, this declares a “`tieto`” for this crate — meaning that when we read out `smt0_0`, we also need to include the sequencer crate `seq0`.
- [11] Lines 46 and 48: These two definitions are required. The first declares the trigger framework crate; the second defines a special geographic section that the framework uses for sending trigger notifications to Level 3.
- [12] Line 52: This element contains the definitions for the level-1 resources. The attributes of this element give the number of available exposure groups (`n_expogroups`) and level-1 bits (`n_bits`).
The various and/or terms available are defined by the elements contained within this one.
- [13] Lines 53 and 56: These show definitions of specific, named and/or terms. Each of these elements associates a name with a particular and/or term number.
- [14] Lines 58 and 61: This is a definition for a level-1 trigger manager card; the number of the card’s first output and/or term is 48. The `class` attribute says that this manager card is part of the muon system, and the `device` attribute gives the name to be used when downloading the card.

The `term` elements contained within the `trigmgr` element assign names to the manager's input terms.

- [15] Line 63: This element defines a set of level-1 calorimeter trigger threshold terms. These terms compute sums over the calorimeter and compare the result with a specified threshold. This example declares a group of eight `11emetsum` terms, which means that up to eight distinct thresholds may be set for this type of sum (scalar E_T sum over the electromagnetic calorimeter).
- [16] Line 65: This definition is required for proper operation. It defines a trigger term that is guaranteed to be always asserted.
- [17] Line 67: This assigns a name to a Level 1 qualifier. The 'number' here is the bit position.
- [18] Line 71: This element contains the definitions for the Level 2 resources.
- [19] Lines 73 to 94: This defines a Level 2 preprocessor crate. Level 2 can have several preprocessor objects in one preprocessor crate (for example, the EM, missing E_T , and jet preprocessors all live in the `12cal` crate), so we separate out the definitions that are common to all preprocessors in a crate. The `crate` attribute names the preprocessor crate we're defining; it should match the name of one of the readout crates defined above. By default, the name sent to Level 2 will be an upcased version of this name, but this may be changed using the optional `12cratename` attribute. A `12ppcrate` element can contain `12input` elements, describing the inputs to the preprocessor. Each input has a number and a list of crates feeding it. To enable an input, all of the listed crates must be active.

To enable or disable inputs, COOR sends to Level 2 the string contained in the `12mbt` element. Here, a construction like `"%N?onstr:offstr%"` (where N is a string of decimal digits) will be replaced with `onstr` if input N is enabled and with `offstr` otherwise. The notation `"%N"` (with no following `'`) is equivalent to `"%N?N:-1%"`.

- [20] Lines 118 and 123: This makes some global definitions for the Level 2 system. The attribute `crate` should name a crate given in an earlier `12ppcrate` declaration, describing the Level 2 global crate. (The inputs

to this crate should be the preprocessors.) The attributes `exe_name` and `l2_name` give the name of the executable, and also the name of the object for global parameter settings. The Level 2 system may have some global parameters that can be set in the trigger configuration; those parameters are defined by the `l2parm` elements contained within this element. Here, we define a single, integer, parameter, and give it a default. Note that if we don't provide a default here, then this parameter must be specified in every trigger configuration that uses Level 2. The attribute `fail_tooltype` must name a Level 2 tool type that is guaranteed to always fail.

- [21] Lines 125 to 129: This defines a Level 2 preprocessor, with a name given by the `name` attribute. The crate it lives in is given by the `crate` attribute; this must match one of the earlier `l2ppcrate` definitions. The `exe_name` and `l2_name` attributes give the executable and object names to use to set parameters for this preprocessor. The `l1qual` attribute gives the name of the Level 1 qualifier to use to request that this crate process the current event.

A preprocessor may have parameters that can be set from the trigger configuration. These are described by the contained `l2parm` elements.

- [22] Lines 131 to 134: This defines a Level 2 tool type; its name is given by `name`. The `pp` attribute gives the list of preprocessors that this tool requires in order to run. A tool may have parameters; these are described by the contained `l2parm` elements.

- [23] Lines 136 to 140: This defines a second tool. Note that this tool has a parameter “`tool`” that refers to other tools.

11.2 Resource Element Reference

This section describes the elements used to describe the detector resources to COOR. The DTD used is given in Sec. A.2.1.

11.2.1 Element attribute

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
xmltype	CDATA	"CDATA"
default	CDATA	"#REQUIRED"
onfree	CDATA	""

PARENTS

devtype

CONTENTS

EMPTY

DESCRIPTION

Define one attribute of a device type. The name of the attribute is `name`. The XML type of this attribute is given by `xmltype`; useful values for this are 'CDATA' and XML enumerations. The default value for the attribute is given by `default`. If this is set to '#REQUIRED', then there is no default (and the attribute must always be specified). If `onfree` is not empty, then when a device of this type is freed by the last owner, then a download will be generated to set this attribute to the given value.

11.2.2 Element calpulser

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
comics_name	CDATA	#REQUIRED
downloader	CDATA	""

PARENTS

devices

CONTENTS

(tieto*)

DESCRIPTION

Define a calorimeter pulser called **name**. The name of the COMICS device is **comics_name**; if this is omitted, it is taken to be the same as **name**. If the **downloader** attribute is given, it names a specific downloader to which downloads for this pulser must be sent.

11.2.3 Element crate

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
type	CDATA	#REQUIRED
downloader	CDATA	""
geosect	CDATA	#REQUIRED
novbd	(yes no)	"no"

PARENTS

crates

CONTENTS

((subdevice|tieto)*)

DESCRIPTION

Define a readout crate. The crate name is **name**, and the name of its type is **type**. The attribute **geosect** (an integer) gives the geographic sector of the crate. The crate name must be unique. If the **downloader** attribute is given, it names a specific downloader to which downloads for this device must be sent.

If the **novbd** attribute is set, then this element names a geographic sector that is an SCL receiver but which does not have a VBD. Therefore, requests for such a “crate” will be passed to the level-1 framework, but not to level-3.

11.2.4 Element crates

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

resources

CONTENTS

(crate*)

DESCRIPTION

This element contains the set of available readout crates.

11.2.5 Element device

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
type	CDATA	#REQUIRED
downloader	CDATA	""

PARENTS

devices

CONTENTS

((subdevice|tieto)*)

DESCRIPTION

Define a downloadable device (which is not a readout crate). The device name is **name**, and the name of its type is **type**. The device name must be unique. If the **downloader** attribute is given, it names a specific downloader to which downloads for this device must be sent.

11.2.6 Element devices

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

resources

CONTENTS

((device|calpulser)*)

DESCRIPTION

This element contains the set of available downloadable devices (excluding readout crates).

11.2.7 Element devtype

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
comics_prefix	CDATA	""
default_inhibit	(yes no)	"no"

PARENTS

resources

CONTENTS

(attribute*)

DESCRIPTION

Define a device type `name`. The type's attributes are defined by the contained `attribute` elements. The `comics_prefix` attribute gives a string to add to the front of names for devices of this type when they get sent to COMICS. If `default_inhibit` is 'yes', then devices of this type will have their `inhibit` attribute default to 'yes'.

11.2.8 Element `l1ct_emcount`

ATTRIBUTES

Name	Type	Default
<code>refsetno</code>	CDATA	#REQUIRED
<code>first_term</code>	CDATA	#REQUIRED
<code>n_terms</code>	CDATA	#REQUIRED

PARENTS

`level1`

CONTENTS

EMPTY

DESCRIPTION

A set of level-1 calorimeter trigger electromagnetic count threshold trigger terms. These terms count the number of trigger towers which are above programmable thresholds set in associated *reference sets*; a term fires if this count is above a given threshold.

Actually, there are two reference sets for these terms. The term fires if the electromagnetic energy in a tower is greater than the threshold set by the first reference set and the hadronic energy in the tower is less than the threshold set by the second reference set.

The attribute `refsetno` gives the number of the associated reference sets (of classes `l1em_refset` and `l1hadveto_refset`).

The attribute `n_terms` gives the number of available and/or terms of this type (and thus the number of distinct values which may be set). The number of the first and/or term of this type is given by the attribute `first_term`. The attributes `refset1` and `refset2` give the full names (class plus number) of the associated reference sets.

11.2.9 Element `l1ct_emcountr`

ATTRIBUTES

Name	Type	Default
<code>refsetno</code>	CDATA	#REQUIRED
<code>count</code>	CDATA	#REQUIRED
<code>number</code>	CDATA	#REQUIRED
<code>region</code>	(c n s)	#REQUIRED

PARENTS

`level1`

CONTENTS

EMPTY

DESCRIPTION

A set of level-1 calorimeter trigger electromagnetic count threshold trigger terms, with a region restriction. These terms count the number of trigger towers which are above programmable thresholds set in associated *reference sets* within a given region of the detector; a term fires if this count is above `count`.

Actually, there are two reference sets for these terms. The term fires if the electromagnetic energy in a tower is greater than the threshold set by the first reference set and the hadronic energy in the tower is less than the threshold set by the second reference set.

The attribute `refsetno` gives the number of the associated reference sets (of classes `l1em_refset` and `l1hadveto_refset`).

The attribute `number` gives the and/or term number.

The attribute `region` is the region of the detector to which this term is sensitive: 'c' (central), 'n' (north), or 's' (south).

11.2.10 Element `l1ct_emquad`

ATTRIBUTES

Name	Type	Default
<code>count</code>	CDATA	#REQUIRED
<code>number</code>	CDATA	#REQUIRED
<code>quadrant</code>	(1 2 3 4)	#REQUIRED

PARENTS

`level1`

CONTENTS

EMPTY

DESCRIPTION

A set of level-1 calorimeter trigger electromagnetic count threshold trigger terms, with a quadrant restriction. These terms count the number of trigger towers which are above a fixed threshold (burned into PROMS) within a given quadrant of the detector; a term fires if this count is above `count`.

The attribute `number` gives the and/or term number.

The attribute `quadrant` is the quadrant of the detector to which this term is sensitive, 1-4.

11.2.11 Element `l1ct_jetcount`

ATTRIBUTES

Name	Type	Default
<code>refsetno</code>	CDATA	#REQUIRED
<code>first_term</code>	CDATA	#REQUIRED
<code>n_terms</code>	CDATA	#REQUIRED

PARENTS

level1

CONTENTS

EMPTY

DESCRIPTION

A set of level-1 calorimeter trigger total count threshold trigger terms. These terms count the number of trigger towers which are above programmable thresholds set in associated *reference sets*; a term fires if this count is above a given threshold.

The attribute `refsetno` gives the number of the associated reference set (of class `l1jet_refset`).

The attribute `n_terms` gives the number of available and/or terms of this type (and thus the number of distinct values which may be set). The number of the first and/or term of this type is given by the attribute `first_term`. The attributes `refset1` and `refset2` give the full names (class plus number) of the associated reference sets.

11.2.12 Element `l1ct_jetcount`

ATTRIBUTES

Name	Type	Default
<code>refsetno</code>	CDATA	#REQUIRED
<code>count</code>	CDATA	#REQUIRED
<code>number</code>	CDATA	#REQUIRED
<code>region</code>	(c n s)	#REQUIRED

PARENTS

level1

CONTENTS

EMPTY

DESCRIPTION

A set of level-1 calorimeter trigger total count threshold trigger terms, with a region restriction. These terms count the number of trigger towers which are above programmable thresholds set in associated *reference sets* within a given region of the detector; a term fires if this count is above **count**.

The attribute **refsetno** gives the number of the associated reference set (of class **l1jet_refset**).

The attribute **number** gives the and/or term number.

The attribute **region** is the region of the detector to which this term is sensitive: 'c' (central), 'n' (north), or 's' (south).

11.2.13 Element **l1ct_jetquad**

ATTRIBUTES

Name	Type	Default
count	CDATA	#REQUIRED
number	CDATA	#REQUIRED
quadrant	(1 2 3 4)	#REQUIRED

PARENTS

level1

CONTENTS

EMPTY

DESCRIPTION

A set of level-1 calorimeter trigger total count threshold trigger terms, with a quadrant restriction. These terms count the number of trigger towers which are above a fixed threshold (burned into PROMS) within a given quadrant of the detector; a term fires if this count is above **count**.

The attribute **number** gives the and/or term number.

The attribute **quadrant** is the quadrant of the detector to which this term is sensitive, 1–4.

11.2.14 Element `l1ct_l1tcount`

ATTRIBUTES

Name	Type	Default
<code>refsetno</code>	CDATA	#REQUIRED
<code>count</code>	CDATA	#REQUIRED
<code>number</code>	CDATA	#REQUIRED

PARENTS

`level1`

CONTENTS

EMPTY

DESCRIPTION

A level-1 calorimeter trigger large tile term. These terms count the number of large tiles which are above programmable thresholds set in associated *reference sets*; this term fires if the count is above `count`.

The attribute `refsetno` gives the number of the associated reference sets (of class `l1t_refset`).

The attribute `number` gives the and/or term number.

11.2.15 Element `l1ct_refset`

ATTRIBUTES

Name	Type	Default
<code>class</code>	(<code>l1em_refset l1hadveto_refset l1jet_refset l1t_refset</code>)	#REQUIRED
<code>n_sets</code>	CDATA	#REQUIRED

PARENTS

`level1`

CONTENTS

EMPTY

DESCRIPTION

This element defines a set of level-1 calorimeter trigger reference sets.

The type of the reference set is specified by the `class` attribute:

<code>l1em_refset</code>	Used for EM E_T count thresholds.
<code>l1hadveto_refset</code>	Used for EM E_T count thresholds.
<code>l1jet_refset</code>	Used for total E_T count thresholds.
<code>l1lt_refset</code>	Large tile reference sets.

The attribute `n_sets` gives the number of available reference sets of this type.

11.2.16 Element `l1ct_thresh`

ATTRIBUTES

Name	Type	Default
<code>class</code>	(<code>l1emetsum l1hdetsum l1totetsum l1misspt</code>)	#REQUIRED
<code>first_term</code>	CDATA	#REQUIRED
<code>n_terms</code>	CDATA	#REQUIRED

PARENTS

`level1`

CONTENTS

EMPTY

DESCRIPTION

A set of level-1 calorimeter trigger value threshold trigger terms. These terms sum over the entire calorimeter in various ways and compare the result with a specified threshold.

The type of sum is specified by the `class` attribute:

<code>11emetsum</code>	Scalar E_T sum over EM calorimeter.
<code>11hdetsum</code>	Scalar E_T sum over hadronic calorimeter.
<code>11totetsum</code>	Scalar E_T sum over entire calorimeter.
<code>11misspt</code>	Vector E_T sum over entire calorimeter.

The attribute `n_terms` gives the number of available and/or terms of this type (and thus the number of distinct values which may be set). The number of the first and/or term of this type is given by the attribute `first_term`.

11.2.17 Element `11qual`

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	#REQUIRED
<code>number</code>	CDATA	#REQUIRED

PARENTS

`level1`

CONTENTS

EMPTY

DESCRIPTION

This element assigns a name to a level-1 qualifier. Here, `number` is the qualifier bit number, starting from 0.

11.2.18 Element `l2global`

ATTRIBUTES

Name	Type	Default
<code>crate</code>	CDATA	#REQUIRED
<code>exe_name</code>	CDATA	#REQUIRED
<code>l2_name</code>	CDATA	#REQUIRED
<code>fail_tool</code>	CDATA	#REQUIRED

PARENTS

`level2`

CONTENTS

(`l2parm*`)

DESCRIPTION

This element provides some global information about the Level 2 system. The `crate` attribute gives the name of the Level 2 global processor crate; it should match the name given in one of the `l2ppcrate` elements. Further, for that `l2ppcrate` element, all the `l2input` crate lists must have exactly one crate, which must be a Level 2 preprocessor crate (named in a `l2ppcrate` element). The `exe_name` and `l2_name` attributes identify the executable and object names to Level 2 (these names are upcased when they are sent to Level 2). The attribute `fail_tool` should be the name of a Level 2 tool that is guaranteed to always fail (regardless of any mark/pass settings, etc). That tool must be defined below, and must have no required parameters.

A `l2global` element may contain `l2parm` elements. These define global parameters of the Level 2 system that must be set. When a trigger configuration is read, requested values are set in the trigger configuration's `l2global` element.

11.2.19 Element `l2input`

ATTRIBUTES

Name	Type	Default
<code>number</code>	CDATA	#REQUIRED
<code>crates</code>	CDATA	#REQUIRED

PARENTS

`l2ppcrate`

CONTENTS

EMPTY

DESCRIPTION

This element describes an input to a preprocessor. The `number` attribute should be an integer, giving the number of the input. The `crates` attribute should be a space-separated list of crate names. All the crates in the list must have been allocated in order to activate the input.

11.2.20 Element `l2mbt`

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

`l2ppcrate`

CONTENTS

(#PCDATA)

DESCRIPTION

This element gives a template string for enabling and disabling MBT inputs. Whenever the inputs are changed, this string will be sent to Level 2, transformed in the following ways:

- A “`l2script` ” will be added to the beginning.
- A construction like “`%N?onstr:offstr%`” (where *N* is a string of decimal digits) will be replaced with *onstr* if input *N* is enabled and with *offstr* otherwise. The notation “`%N`” (with no following ‘%’) is equivalent to “`%N?N:-1%`”.

11.2.21 Element `l2parm`

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	#REQUIRED
<code>type</code>	CDATA	#REQUIRED
<code>default</code>	CDATA	"#REQUIRED"

PARENTS

`l2global`, `l2pp`, `l2tool`

CONTENTS

EMPTY

DESCRIPTION

This element describes a parameter to a Level 2 object (a preprocessor or tool, or a global parameter). The `name` attribute gives the parameter name (this name is upcased before being sent to Level 2). The `type` attribute gives the type of the parameter. This should be one of the following strings:

- “`int`” — An integer.
- “`float`” — A floating-point value.

- "string" — An uninterpreted string value.
- "tool" — A reference to a Level 2 tool. The value should be the name of the tool.

The `default` attribute gives a default value for the parameter. If it is omitted, then the parameter must always be provided.

11.2.22 Element `l2pp`

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	#REQUIRED
<code>crate</code>	CDATA	#REQUIRED
<code>exe_name</code>	CDATA	#REQUIRED
<code>l2_name</code>	CDATA	#REQUIRED
<code>l1qual</code>	CDATA	#REQUIRED

PARENTS

`level2`

CONTENTS

(`l2parm*`)

DESCRIPTION

This element describes a Level 2 preprocessor, called `name`. The `crate` attribute gives the name of the crate in which this preprocessor is located; it should match the name given in one of the `l2ppcrate` elements. The `exe_name` and `l2_name` attributes identify the executable and object names to Level 2 (these names are upcased when they are sent to Level 2). The `l1qual` attribute gives the name of the Level 1 qualifier used to request that this preprocessor run. It must have been defined earlier in a `l1qual` element.

This element may contain `l2parm` elements, to define parameters to be downloaded to the preprocessor.

11.2.23 Element l2ppcrate

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
crate	CDATA	#REQUIRED
l2cratename	CDATA	""

PARENTS

level2

CONTENTS

(l2input*, l2mbt?)

DESCRIPTION

This element defines information common to all preprocessors in a crate. The `crate` attribute gives the name of the crate; it should match the name of one of the readout crates defined earlier. The `l2cratename` gives the name by which this crate is known to Level 2. If omitted, it defaults to an upcased version of the crate name.

This element may contain `l2input` elements to define the inputs to the preprocessor. A `l2mbt` element gives the template for enabling the preprocessor inputs.

11.2.24 Element l2tool

ATTRIBUTES

Name	Type	Default
name	CDATA	#REQUIRED
l2name	CDATA	""
pp	CDATA	""
maxinstances	CDATA	"-1"

PARENTS

level2

CONTENTS

(l2parm*)

DESCRIPTION

This element defines a Level 2 tool type. The `name` attribute is the name of the tool type. It is strongly suggested that this name start with 'l2'. The `l2name` attribute is the by which this tool is known to Level 2. If it is omitted, it will default to the value of `name`. This can be used if the Level 2 name does not start with 'l2'. (The Level 2 name is upcased before being sent to Level 2). The attribute `l2pp` is a space-separated list of required preprocessors. (These names must have been defined in an earlier `l2pp` element.) The `maxinstances` attribute specifies the maximum number of allowed instances of this tool. The default of -1 indicates that no limit checking is done.

This element may contain `l2parm` elements, to define the tool parameters.

11.2.25 Element level1

ATTRIBUTES

Name	Type	Default
<code>n_expogroups</code>	CDATA	#REQUIRED
<code>n_bits</code>	CDATA	#REQUIRED

PARENTS

resources

CONTENTS

((term|trigmgr|l1ct_thresh|l1ct_refset|l1ct_emcount|l1ct_jetcount|l1ct_ltcountr|l1ct_emcountr|l1ct_jetcountr|l1ct_emquad|l1ct_jetquad|l1qual)*)

DESCRIPTION

This element defines the available level-1 resources. The attribute `n_expogroups` gives the number of available exposure groups, while the `n_bits` attribute give the number of available level-1 trigger bits.

Further level-1 resources are defined by the elements contained in this one.

11.2.26 Element `level2`

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

`resources`

CONTENTS

(`l2ppcrate*`, `l2global`, `l2pp*`, `l2tool*`)

DESCRIPTION

This element holds the definitions of the available level-2 resources. It starts with a list of `l2ppcrate` elements, followed by a (required) `l2global` element. It then contains `l2pp` and `l2tool` elements, in that order.

11.2.27 Element `resources`

ATTRIBUTES

Name	Type	Default
(None.)		

PARENTS

None.

CONTENTS

(devtype*, devices?, crates?, level1?, level2?)

DESCRIPTION

This is the top-level element for resource definitions. It contains sections to define downloadable devices, readout crates, and level-1 resources.

11.2.28 Element subdevice

ATTRIBUTES

Name	Type	Default
comics_name	CDATA	#REQUIRED
attribs	CDATA	#REQUIRED

PARENTS

crate, device

CONTENTS

EMPTY

DESCRIPTION

Sometimes it is convenient to be able to refer to several EPICS devices by a single name. That can be done using this element. In essence, one defines a “logical” device to COOR that is the union of several EPICS devices.

The “logical” device you define to COOR should contain the attributes for all the EPICS devices associated with that logical device. (Note that it is a restriction that those EPICS devices must have different attribute names.) Then, in the `device` or `crate` element in the resource file, include a `subdevice` element for each of the additional EPICS devices, giving the attributes that that device handles. Note that one can also use this mechanism to be

able to use a name on COOR different from the EPICS name, simply by having all the device's attributes handled by subdevices.

The `comics_name` attribute is the name of the subdevice as it should be sent to COMICS. The `attribs` attribute is a space-separated list of attribute names handled by this subdevice.

11.2.29 Element term

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	#REQUIRED
<code>number</code>	CDATA	#REQUIRED

PARENTS

`level1`, `trigmgr`

CONTENTS

EMPTY

DESCRIPTION

This element can occur in two contexts. In a `level1` element, it defines a fixed and/or term to the level-1 framework. These are terms which don't require any downloading. The term name is `name`, and `term` (an integer) is the term number. In a `trigmgr` element, it assigns a name to one of the trigger manager's input terms.

11.2.30 Element tieto

ATTRIBUTES

Name	Type	Default
<code>name</code>	CDATA	#REQUIRED

PARENTS

`crate`, `device`, `calpulser`

CONTENTS

EMPTY

DESCRIPTION

Sometimes, devices interact in such a way that if you allocate one of them, you should allocate the other as well. This is called “tying.” For example, the calorimeter pulsers should be tied to the crates that they affect. That way, if someone loads a configuration that uses those crates but not the pulsers, others will be prevented from controlled the pulsers affecting those crates.

Ties may be defined for crates, devices, and calorimeter pulsers. They are specified by a group of `tieto` elements, where the `name` attribute names either a device or a crate.

After processing the trigger configuration, COOR looks for objects with ties. If the targets of the ties are not allocated by this client, COOR attempts to allocate them. (The ownership mode as the original object is used. The `inhibit` attribute is also propagated.) For targets that are already owned, their ownership mode must be at least as restrictive as that of the original object.

If a crate is tied to another crate, then when the first crate is used in the readout list of an exposure group, the second crate is automatically added as well. This is useful for modeling, for example, the dependence of tracking crates on sequencers.

Note that the devices named in a `tieto` must have defaults specified for all their attributes.

11.2.31 Element `trigmgr`

ATTRIBUTES

Name	Type	Default
<code>class</code>	<code>(11ctt 11fpd 11fps 11lum 11muo)</code>	<code>#REQUIRED</code>
<code>first_term</code>	CDATA	<code>#REQUIRED</code>
<code>n_terms</code>	CDATA	<code>"16"</code>
<code>n_terms</code>	CDATA	<code>"16"</code>
<code>term16</code>	CDATA	<code>""</code>
<code>device</code>	CDATA	<code>#REQUIRED</code>

PARENTS

`level1`

CONTENTS

`(term*)`

DESCRIPTION

This element defines a trigger manager card, which has `n_terms` and/or terms as output. The and/or terms are assumed to be in two blocks. The first 16 terms are in contiguous block starting at number `first_term`, and the remaining terms are also in a contiguous block starting at number `term16`. (If `term16` is left blank, it defaults to `first_term+16`). The `device` attribute gives the COMICS device name of the card, to be used for downloading. If this is blank, then no downloading will be done. The `class` attribute identifies the subsystem of which this trigger manager is a part.

The `term` elements contained in this element assign names to the manager's input terms.

12 TAKER

12.1 Run Transition Dialogs

Before a run transition (start, end, pause, resume), TAKER can prompt the user for information about the transition, such as a comment. This takes the form of a set of keyword-value pairs. This information then ends up in several places:

- In the `brun` and `erun` files, to be inserted in the database. (See Sec. 7.)
- In the logbook entries generated by COOR.
- In the run transition messages sent by COOR to the significant event system. (See Sec. 8.4.)

The set of questions that TAKER asks is specified by the configuration parameter `'brun_dialog_descs'`. This variable should be a list of tuples, each of which has one of the following forms:

`(pred, 'entry', label, kwd[, validate])`

or

`(pred, 'radio', label, kwd, butlist[, ncol])`

Each field is either an entry or a radio button. In the above, *pred* is a predicate which determines when this field will be requested (see below), *label* is the string that gets displayed to the user, and *kwd* is the tag used in output. For entry fields, *validate* is an optional entry validator function; see `Pmw.EntryField` for details. For button fields, *ncol* is the number of columns to use for the button layout. If it is omitted, they will be put in a single horizontal row.

The predicate *pred* should be a function. It takes a dictionary as an argument, and should return true if this particular field is to be displayed. The entries in the dictionary consist of the configuration information returned by COOR in response to the 'load' command (see Sec. 8.1). In addition, TAKER adds the key 'transition', with its value being one of 'begin', 'end', 'pause', or 'resume'.

A simple example:

```

def begin_p (x):    return x.get ('transition') == 'begin'
def end_p (x):     return x.get ('transition') == 'end'
def always (x):    return 1
brun_dialog_descs = [(begin_p, 'entry', 'Shifter', 'Shifter'),

                    (end_p,   'radio',
                      'Evaluation', 'Evaluation',
                      ['Good', 'Uncertain', 'Useless']),

                    (always,  'entry', 'Comment', 'Comment'),

                    (lambda x:
                     begin_p(x) and x.get('physics'):
                      'entry',
                      'Initial luminosity',
                      'Initlum')]

```

If a run is stopped by a `force_stop` (or paused via `force_pause`), then the user will not have a chance to answer the end-run dialogs. In that case, COOR will look in the configuration variable `'brun_fallbacks'` to determine what to output. This variable should be a dictionary. The keys that are currently examined are `'end_run'` and `'pause_run'`. The value of each key should be a list of strings in the form `'key: value'` (as it would be output to the brun file). Note that COOR will automatically add a `'Comment'` line.

12.2 TAKER plugins

For some applications, one would like to extend the user interface provided by TAKER. This can be done without changing the TAKER code itself by using *plugins*. A TAKER plugin is a module of Python code, obeying certain conventions. All known plugins are gathered into a TAKER menu. When one of these menu items is chosen, the corresponding plugin module is imported and activated by calling a special entry point in the module. TAKER presents an interface to the plugin, through which it may send commands to COOR.

12.2.1 Finding plugins

TAKER has a directory search path in which to look for plugins. This is normally set through the `TAKER_PLUGINS` environment variable (implemented

through the `taker_plugins` variable in `coor.params`). `TAKER` searches this path for files with names ending in `'_plugin.py'`. This is the set of available plugins. They are made available in the 'Plugins' menu.

12.2.2 Activating plugins

Plugins may be activated only after a trigger configuration has been loaded.

When the user selects a plugin from the menu, `TAKER` imports its source, then calls the function `'init_plugin (parent, t_proxy)'` in the plugin module. In this call `parent` is the main Tk window for the application, and `t_proxy` is an object through which the plugin can request services of `TAKER`.

12.2.3 Taker proxy methods

The proxy object `t_proxy` passed to the plugin's `init_plugin` method supplies the following methods:

- `display_message (self, message)`

Display message in the `TAKER` text window.

- `info_command (self, cmd, done, filter = None)`

Send an information request to `COOR` (`A_INFO` `TAKER` transition). `cmd` is the text of the command to send. It will have `'info '` automatically prepended.

`done` is a function object to call when the request completes. It is called with two arguments: the return status from `COOR`, and `filter`. Note: The `COOR` return status tells only whether the requested state transition actually occurred, not whether the request itself succeeded. For most cases, this will always be true. This may change in the future.

By default, any `TEXT` messages sent back from `COOR` are displayed in the text window. This can be changed by supplying an object instance for the `filter` argument. In that case, for each `TEXT` message sent by `COOR` during processing of this request, the `'filter'` method of the `filter` object will be called with the `COOR` message as an argument (and the message will not be displayed).

`info_command()` returns true if the request was successfully sent to COOR, false otherwise (because TAKER was in a state that did not allow A_INFO transitions).

- `change_command (self, cmd, done, filter = None)`

This is like `info_command()`, except that 'info ' is not prepended to `cmd`, and an A_MODIFY transition is made instead of A_INFO.

- `add_free_cb (self, cb)`

Request that `cb` be called when the current trigger configuration is freed. `cb` is called with no arguments.

- `del_free_cb (self, cb)`

Remove `cb` from the list of configuration free callbacks. `cb` must be exactly the same object that was passed to `add_free_cb()`.

12.2.4 Plugin interactions

For plugins that have user interfaces, there are two ways they can be designed.

- As a modal dialog. In this case, the function that starts the dialog does not return until it has been dismissed. The rest of the TAKER application is nonresponsive during this time.

See `coor/src/py/plugins/modal_plugin.py` for an example of this style.

- As a non-modal dialog. In this case, the function that starts the dialog returns, leaving the dialog posted. Both the plugin user interface and the TAKER user interface are available.

If this style is used, the plugin should be set up to delete itself if the trigger configuration is freed. The `add_free_cb()` method should be used for this purpose.

See `coor/src/py/plugins/nonmodal_plugin.py` for an example of this style.

13 The Name/Value Service

13.1 Introduction

COOR also provides a simple name/value service, mapping arbitrary names to string values. Names are hierarchical, using a period as a separator. Some examples of names are `‘.coor.store_number’` or `‘.cal.ccne.calibdate’`. (These are examples only; these particular names may not actually exist.) Any given name can either have a simple string value, or it can be a directory of other names. A name may also have a collection of named properties associated with it.

Names starting with `‘.coor’` are reserved to COOR.

Names that have the `‘brun’` property set to something (other than `‘0’` or the null string) will be written to the brun file (see Sec. 7). Names that have the `‘erun’` property set will be written to the erun file.

The name database is persistent. It is saved in the file pointed to by the environment variable `COOR_NAME_SERVER_DB`.

The protocol used for communicating with the name server is described in section Sec. 8.5.

13.2 The Python Name Service API

A simple Python class is provided to access the name service. Create an instance of the `NV_Service` class like this:

```
from coor.NV_Service import NV_Service
nv = NV_Service()
```

The `NV_Service` constructor takes a few optional arguments:

- `addr = None`: The address to which to connect. This should be a string in the form `‘HOST:PORT’`. If it is defaulted, the class first checks the environment variable `COOR_NAME_SERVER_ADDR`. If that is not set, then the address is taken from `d0online_names`.
- `timeout = 5`: The timeout for name service transactions.
- `cn_class = itc.SOCK_Auto_Connector`: The class to use for the underlying connection. This may be replaced with a different class that provides the same interface as `itc.SOCK_Auto_Connector`.

Errors are reported by raising `NV_Service.Error`. The first element of the argument tuple is an error code, the second element is the message sent to the server, and the third element is what was received from the server. The error code can be any of the itc error codes, or one of these symbols defined in the `NV_Service` class:

- `BAD_TYPE`: The message returned by the server was of an incorrect itc type.
- `BAD_MSGSEQ`: The message received by the server had an incorrect itc message sequence number.
- `BAD_REPLY`: The server did not reply 'ok'.

Methods available in the `NV_Service` class include:

- `checkpoint` — Request that the name database be written to disk immediately.
- `get name` — Get the value of *name* from the server. If *name* is an ordinary variable, the result is returned as a string. If *name* is a directory, the result is a list of the directory contents. Each element of the list is a name, ending in a trailing period if that name is itself a directory.
- `getall name` — Get the value of *name* and (if it's a directory) everything underneath it. Returns a dictionary. The key `'_value'` in that dictionary gives the value of the name (this will itself be a dictionary if this name is a directory). Each property of the name is also present as a key in the dictionary. Returns `None` if there was a problem.
- `getprop name property` — Get the value of the *property* property of *name*.
- `set name value` — Set the value of *name* to *value*.
- `setprop name property value` — Set the value of the *property* property of *name* to *value*.

13.3 The C++ Name Service API

A simple C++ class is also provided to access the name service. The header is in `coor/NV_Service.hpp`; the code is in `-lcoor`.

This defines the class `coor::NV_Service`. Create an instance of the `coor::NV_Service` class like this:

```
#include "coor/NV_Service.hpp"
...
coor::NV_Service nv;
```

The `coor::NV_Service` constructor takes a few optional arguments:

- `host = ""`: The host to which to connect.
- `port = 0`: The port to which to connect. If either the host or the port are defaulted, they are filled in by looking in the following places:
 - The `COOR_NAME_SERVER_ADDR` environment variable.
 - The `d0online_names.py` file.
 - Compiled-in defaults.
- `timeout = 5`: The timeout for name service transactions.

There is also an alternate constructor that can be used to replace the default `itc` connection class with a different class. See the header.

Errors are reported by raising the exception `coor::NV_Service_Error`, which is a `ZMexception`.

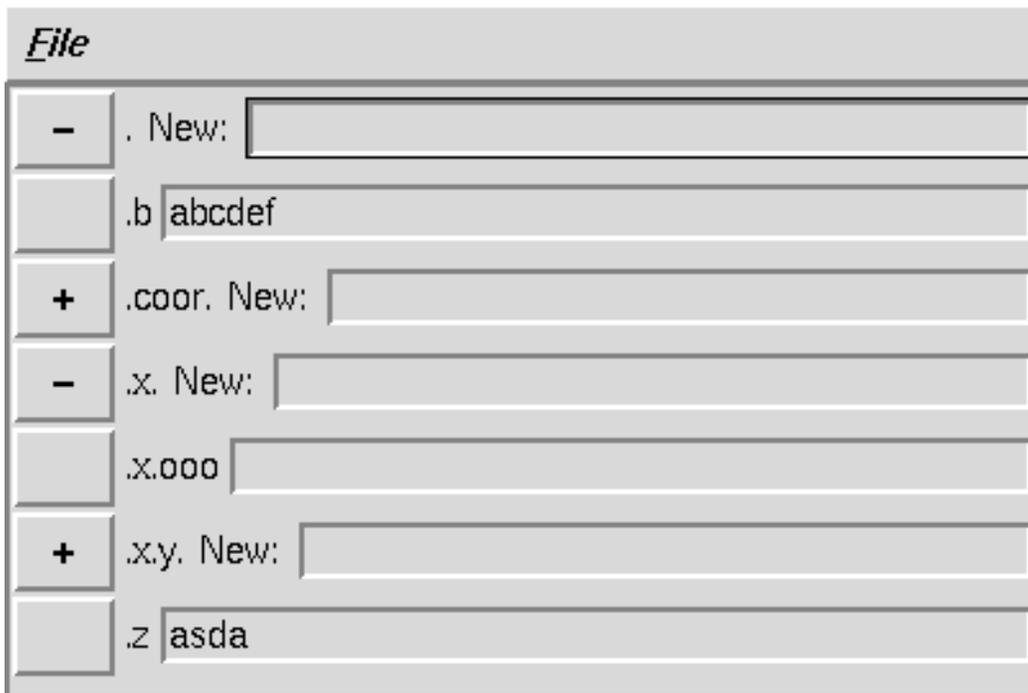
Methods available in the `NV_Service` class include:

- `checkpoint` — Request that the name database be written to disk immediately.
- `get_var name` — Get the value of `name` from the server. It is an error if `name` is not an ordinary variable.
- `get_dir name` — Get the contents of directory `name` from the server. It is an error if `name` is not a directory. The result is a vector of strings. Each element of the vector is a name, ending in a trailing period if that name is itself a directory.

- `getprop name property` — Get the value of the *property* property of *name*.
- `set name value` — Set the value of *name* to *value*.
- `setprop name property value` — Set the value of the *property* property of *name* to *value*.

13.4 The Name Service Editor

A simple editor program is available for manually making changes to the database. Start it with the command `'nv_editor'`. Here's what it looks like:



Names which are ordinary variables have a blank button in front of them. To change the value of a name, move to the entry after that name, type the new value, and press enter. Once you start making changes to an entry, the text will turn red until the changes have been sent to the server.

Names which are directories have a button in front of them with a '+' if the contents of the directory are not being shown, or '-' if they are. Click

on the button or use the ‘Insert’ key to toggle between these states. To create a new name in a directory, type the name of the new variable in the ‘New:’ entry following the directory’s name and press Enter. You may create multiple directory levels by giving a name containing a period.

You can delete a name by selecting its entry (so that the entry has the input focus) and selecting ‘Delete’ from the menu (or use ‘Alt-d’). You will be asked to confirm a deletion. Deleting a directory deletes that entire subtree.

To resynchronize the display with the current contents of the name server, choose ‘Update’ from the menu (or use ‘Alt-u’).

You can edit the properties of a name by choosing ‘Properties’ from the menu (or use ‘Alt-p’). You can do this only for an ordinary name, not a directory. In the property editor, you can change the value of an existing property, or create a new, empty, property by typing its name in the ‘New property’ entry. (Note: the property isn’t actually sent to the server until you assign a non-null string to it.)

To quit the editor, choose ‘Quit’ from the menu, or use ‘Alt-q’.

A Document Type Definitions

This section lists the DTDs for the input files which COOR reads.

A.1 Trigger Configuration

A.1.1 trigger_config.dtd

```
1 <!--
2   - $Id: trigger_config.dtd,v 1.45 2002/10/04 03:21:35 snyder Exp $
3   -
4   - File: dtd/trigger_config.dtd
5   - Purpose: DTD for coor trigger configuration.
6   - Created: Dec, 1999, sss
7   -
8   - This file is part of the DTD for coor's trigger configurations.
9   - The top-level element is 'configuration'.
10  -->
11
12
13 <!-- =====
14     Read in the internal scraps.
15  -->
16
17 <!ENTITY % readout_entity_definitions
18     SYSTEM "internal:readout_entity_definitions">
19 %readout_entity_definitions;
20
21
22 <!-- =====
23     Define some abbreviations for attribute definitions.
24  -->
25
26 <!-- A name. -->
27 <!ENTITY % name.att "name CDATA #REQUIRED">
28
29 <!-- A name that is also an ID. -->
30 <!ENTITY % nameid.att "name ID #REQUIRED">
```

```

31
32 <!-- A name for a crate, or crate list. -->
33 <!ENTITY % cratename.att "%name.att;">
34
35 <!-- A reference to a previously defined name, used to specify
36 crates to read out. -->
37 <!ENTITY % readout.att "readout CDATA #REQUIRED">
38
39 <!-- Ownership mode: either exclusive or shared. -->
40 <!ENTITY % ownmode.att "ownmode (exclusive|shared) 'shared'">
41
42 <!-- Level1 term require state : either require or veto. -->
43 <!ENTITY % require.att "require (require|veto) 'require'">
44
45 <!-- The enumeration part of a boolean attribute. -->
46 <!ENTITY % bool.att_enum "(yes|no)">
47
48 <!-- A boolean, defaulting to 'yes'. -->
49 <!ENTITY % bool_yes.att "%bool.att_enum; 'yes'">
50
51 <!-- A boolean, defaulting to 'no'. -->
52 <!ENTITY % bool_no.att "%bool.att_enum; 'no'">
53
54 <!-- Used to specify which one of a set of generic objects to allocate.
55 If defaulted, coor will pick one. -->
56 <!ENTITY % number.att "number CDATA ''">
57
58 <!-- The inhibit-download attribute for devices and crates. -->
59 <!ENTITY % inhibit.att "inhibit %bool_no.att;">
60 <!ENTITY % inhibit.att_yes "inhibit %bool_yes.att;">
61
62
63 <!-- =====
64 Includes.
65 -->
66
67 <!-- Load device type elements. -->
68 <!ENTITY % devtype.attrs "%cratename.att; %ownmode.att; %inhibit.att;">

```

```

69 <!ENTITY % devtype.attribs_inhibited "%cratename.att; %ownmode.att; %inhibit.att;
70 <!ENTITY % devtypes SYSTEM "internal:devtypes_dtd">
71 %devtypes;
72
73
74 <!-- Load l1term elements. -->
75 <!ENTITY % l1term.attribs "%require.att; %ownmode.att;">
76 <!ENTITY % l1terms SYSTEM "l1terms.dtd">
77 %l1terms;
78
79
80 <!-- =====
81     Top-level elements for the state dumps written by coord.
82     -->
83
84 <!ELEMENT clients (client*)>
85
86 <!ELEMENT client (configuration*)>
87 <!ATTLIST client %name.att;
88             run_number CDATA #REQUIRED
89             recording %bool.att_enum; #REQUIRED>
90
91
92 <!-- =====
93     Top-level element for normal trigger configurations.
94     -->
95
96 <!ELEMENT configuration ((download|crate_list)*,
97                         l1refsets?,
98                         (level2?,trigdef+)?,
99                         expogroup*,
100                        sdaq?,
101                        calibration?,
102                        stream*)>
103 <!ATTLIST configuration %name.att;
104                        version CDATA "0"
105                        autopause %bool_no.att;
106                        physics %bool_no.att;

```

```

107         type CDATA "test"
108         comics_runtype CDATA "data"
109         stream_scheme CDATA "">
110
111
112 <!-- =====
113     EPICS download section.
114 -->
115
116 <!ELEMENT download ((%download_content;|Calpulser)*)>
117 <!ATTLIST download name CDATA "">
118 <!-- download_content defined in devtypes.dtd. -->
119
120 <!ELEMENT Calpulser EMPTY>
121 <!ATTLIST Calpulser %name.att;
122                 %ownmode.att;
123                 %inhibit.att;
124                 pattern CDATA 'off'>
125
126
127 <!-- =====
128     Crate list definition section.
129 -->
130
131 <!ELEMENT crate_list (crateref*)>
132 <!ATTLIST crate_list %cratename.att;>
133
134 <!ELEMENT crateref EMPTY>
135 <!ATTLIST crateref ref CDATA #REQUIRED>
136
137
138 <!-- =====
139     Primary DAQ section.
140 -->
141
142 <!-- List of reference sets. -->
143 <!ENTITY % l1refsets "l1em_refset|l1hadveto_refset|l1jet_refset|l1lt_refset">
144

```

```

145 <!ELEMENT trigdef (expogroup*,triglist?)>
146 <!ATTLIST trigdef  l3type    CDATA "REGULAR"
147                   num_nodes CDATA "0">
148
149 <!ELEMENT l1refsets ((%l1refsets;)*)>
150
151
152 <!-- Reference sets. -->
153 <!ENTITY % l1ct_refset.attribs "%ownmode.att; %nameid.att; %number.att;">
154 <!ELEMENT l1em_refset (#PCDATA)>
155 <!ATTLIST l1em_refset %l1ct_refset.attribs;>
156
157 <!ELEMENT l1hadveto_refset (#PCDATA)>
158 <!ATTLIST l1hadveto_refset %l1ct_refset.attribs;>
159
160 <!ELEMENT l1jet_refset (#PCDATA)>
161 <!ATTLIST l1jet_refset %l1ct_refset.attribs;>
162
163 <!ELEMENT l1lt_refset (#PCDATA)>
164 <!ATTLIST l1lt_refset %l1ct_refset.attribs;>
165
166
167 <!-- Level-2 preprocessor/tool/filter declarations. -->
168 <!ENTITY % l2pp.attribs "%ownmode.att;">
169 <!ENTITY % l2pps SYSTEM "internal:l2pp_dtd">
170 %l2pps;
171
172 <!ENTITY % l2tool.attribs "%name.att;">
173 <!ENTITY % l2tool_types SYSTEM "internal:l2tool_dtd">
174 %l2tool_types;
175
176 <!ENTITY % l2global_defs SYSTEM "internal:l2global_dtd">
177 %l2global_defs;
178
179 <!ELEMENT level2 (l2global?,(%l2pp_content;%l2tool_content;l2notused?))>
180 <!ELEMENT l2notused EMPTY> <!-- Clean this up sometime -->
181
182 <!ELEMENT l2script (#PCDATA|l2filter)*>

```

```

183
184 <!ELEMENT l2filter EMPTY>
185 <!ATTLIST l2filter %name.att;
186             count CDATA '1'>
187
188 <!ELEMENT expogroup (l1termlist?,l1trigger*)>
189 <!ATTLIST expogroup %name.att;
190             %number.att;
191             %readout.att;
192             other_gs CDATA "">
193
194 <!ELEMENT l1termlist ((%l1term_content;)*)>
195 <!-- l1term_content defined in l1terms.dtd. -->
196
197 <!-- Attributes for l1trigger. Also used by sdaq_l1trigger. -->
198 <!ENTITY % l1trigger.attrs "%name.att;
199             %number.att;
200             prescale CDATA '1'
201             l2_unbiased_ratio CDATA '16777216'
202             obey_feb %bool_yes.att;
203             auto_disabled %bool_no.att;
204             l1_qualifiers CDATA '0' ">
205
206 <!ELEMENT l1trigger (l1termlist,l2trigger*)>
207 <!ATTLIST l1trigger %l1trigger.attrs;>
208
209 <!ELEMENT l2trigger (l2script?,l3trigger*)>
210 <!ATTLIST l2trigger %name.att;
211             %number.att;>
212
213 <!ELEMENT l3trigger EMPTY>
214 <!ATTLIST l3trigger %name.att;
215             %number.att;>
216
217 <!ELEMENT triglist (#PCDATA)>
218
219
220 <!-- =====

```

```

221     SDAQ section.
222     -->
223
224 <!ELEMENT sdaq ((sdaq_l1trigger?))>
225 <!ATTLIST sdaq type CDATA #REQUIRED
226             %readout.att;
227             parasitic %bool_no.att;
228             only_streams CDATA "">
229
230 <!ELEMENT sdaq_l1trigger (l1termlist?)>
231 <!ATTLIST sdaq_l1trigger %l1trigger.attrs;
232             expogroup_number CDATA "">
233
234
235 <!-- =====
236     Calibration.
237     -->
238
239 <!ELEMENT calibration EMPTY>
240 <!ATTLIST calibration type          CDATA #REQUIRED
241             reference      CDATA 'None'
242             only_streams   CDATA ""
243             %readout.att;>
244
245
246 <!-- =====
247     Data logger section.
248     -->
249
250 <!ELEMENT stream EMPTY>
251 <!ATTLIST stream %name.att;
252             %number.att;
253             family   CDATA "default"
254             relrate  CDATA "1.0">
255
256

```

A.1.2 l1terms.dtd

```
1 <!--  
2   - $Id: l1terms.dtd,v 1.11 2001/04/05 23:49:52 snyder Exp $  
3   -  
4   - File: dtd/l1terms.dtd  
5   - Purpose: DTD for l1 trigger terms.  
6   - Created: Dec, 1999, sss  
7   -  
8   - This file is part of the DTD used when reading a trigger configuration.  
9   - It should define an element for each type of level1 trigger term,  
10  - specifying as attributes the attributes that that term takes.  
11  -  
12  - Each attribute list should include the PE '%l1term.attribs;'; this  
13  - defines the common attributes 'require' and 'ownmode'.  
14  -  
15  - This file should also define the PE 'l1term_content', which is a  
16  - content pattern listing all of the allowed level1 terms.  
17  -->  
18  
19  
20 <!ENTITY %termname.att "name CDATA #REQUIRED">  
21  
22  
23 <!-- A named direct-in term. -->  
24 <!ELEMENT l1specterm EMPTY>  
25 <!ATTLIST l1specterm %l1term.attribs; %termname.att; >  
26  
27 <!-- Trigger manager terms. -->  
28 <!ENTITY %trigmgr.attribs "%l1term.attribs; %termname.att; %number.att;">  
29  
30 <!ELEMENT l1ctt EMPTY>  
31 <!ATTLIST l1ctt %trigmgr.attribs;>  
32  
33 <!ELEMENT l1fpd EMPTY>  
34 <!ATTLIST l1fpd %trigmgr.attribs;>  
35
```

```

36 <!ELEMENT l1fps EMPTY>
37 <!ATTLIST l1fps %trigmgr.attribs;>
38
39 <!ELEMENT l1lum EMPTY>
40 <!ATTLIST l1lum %trigmgr.attribs;>
41
42 <!ELEMENT l1muo EMPTY>
43 <!ATTLIST l1muo %trigmgr.attribs;>
44
45 <!ENTITY % trigmgr_terms "l1ctt|l1fpd|l1fps|l1lum|l1muo">
46
47
48 <!-- ===== Calorimeter resources. ===== -->
49
50 <!-- Definitions -->
51 <!ENTITY % value.att "value CDATA #REQUIRED">
52 <!ENTITY % count.att "count CDATA #REQUIRED">
53 <!ENTITY % l1ct_thresh.attribs "%l1term.attribs; %value.att; %number.att;">
54 <!ENTITY % l1ct_count.attribs "%l1term.attribs; %count.att; %number.att;">
55 <!ENTITY % region.att "region (c|n|s) #REQUIRED">
56 <!ENTITY % quadrant.att "quadrant (1|2|3|4) #REQUIRED">
57
58
59 <!-- Thresholds. -->
60 <!ELEMENT l1emetsum EMPTY>
61 <!ATTLIST l1emetsum %l1ct_thresh.attribs;>
62
63 <!ELEMENT l1hdetsum EMPTY>
64 <!ATTLIST l1hdetsum %l1ct_thresh.attribs;>
65
66 <!ELEMENT l1totetsum EMPTY>
67 <!ATTLIST l1totetsum %l1ct_thresh.attribs;>
68
69 <!ELEMENT l1misspt EMPTY>
70 <!ATTLIST l1misspt %l1ct_thresh.attribs;>
71
72
73 <!-- Counts. -->

```

```

74 <!ELEMENT l1emcount EMPTY>
75 <!ATTLIST l1emcount %l1ct_count.attribs;
76 em_refset IDREF #REQUIRED
77 hadveto_refset IDREF #REQUIRED>
78
79 <!ELEMENT l1jetcount EMPTY>
80 <!ATTLIST l1jetcount %l1ct_count.attribs;
81 jet_refset IDREF #REQUIRED>
82
83 <!ELEMENT l1ltcount EMPTY>
84 <!ATTLIST l1ltcount %l1ct_count.attribs;
85 lt_refset IDREF #REQUIRED>
86
87 <!ELEMENT l1emcountr EMPTY>
88 <!ATTLIST l1emcountr %l1ct_count.attribs;
89 %region.att;
90 em_refset IDREF #REQUIRED
91 hadveto_refset IDREF #REQUIRED>
92
93 <!ELEMENT l1jetcountr EMPTY>
94 <!ATTLIST l1jetcountr %l1ct_count.attribs;
95 %region.att;
96 jet_refset IDREF #REQUIRED>
97
98 <!ELEMENT l1emquad EMPTY>
99 <!ATTLIST l1emquad %l1ct_count.attribs;
100 %quadrant.att;>
101
102 <!ELEMENT l1jetquad EMPTY>
103 <!ATTLIST l1jetquad %l1ct_count.attribs;
104 %quadrant.att;>
105
106
107
108
109 <!-- All caltrig elements. -->
110 <!ENTITY % caltrig_terms "l1emetsum|l1hdetsum|l1totetsum|l1misspt|
111 l1emcount|l1jetcount|l1ltcount|

```

```

112         l1emcountr|l1jetcountr|l1emquad|l1jetquad">
113
114
115 <!-- ===== -->
116 <!-- List all allowed level1 terms. -->
117 <!ENTITY % l1term_content "l1specterm|%trigmgr_terms;|%caltrig_terms;">

```

A.2 Resources

A.2.1 resources.dtd

```

1 <!--
2   - $Id: resources.dtd,v 1.32 2002/08/21 03:09:12 snyder Exp $
3   -
4   - File: dtd/resources.dtd
5   - Purpose: DTD for coor resource definitions.
6   - Created: Jan, 2000, sss
7   -
8   - This file is the DTD for coor's resource definitions.
9   - The top-level element is 'resources'.
10  -->
11
12
13 <!-- =====
14   Define some abbreviations for attribute definitions.
15  -->
16
17 <!-- The enumeration part of a boolean attribute. -->
18 <!ENTITY % bool.att_enum "(yes|no)">
19
20 <!-- A boolean, defaulting to 'no'. -->
21 <!ENTITY % bool_no.att "%bool.att_enum; 'no'">
22
23 <!-- A name. -->
24 <!ENTITY % name.att "name CDATA #REQUIRED">
25
26

```

```

27 <!-- A device type. -->
28 <!ENTITY % type.att "type CDATA #REQUIRED">
29
30
31 <!-- Common device attributes. -->
32 <!ENTITY % device.attribs "%name.att;
33                               %type.att;
34                               downloader CDATA ' '">
35
36
37 <!-- For specifying a range of trigger terms. -->
38 <!ENTITY % first_term.attrib "first_term CDATA #REQUIRED">
39 <!ENTITY % n_terms.attribi "n_terms CDATA">
40
41 <!ENTITY % termrange.attribs "%first_term.attrib;
42                               %n_terms.attribi; #REQUIRED">
43 <!ENTITY % termrange16.attribs "%first_term.attrib;
44                               %n_terms.attribi; '16'">
45
46
47 <!-- =====
48     Top-level element.
49     -->
50
51 <!ELEMENT resources (devtype*,devices?,crates?,level1?,level2?)>
52
53
54 <!-- =====
55     The devices and crates sections.
56     -->
57
58 <!ELEMENT devtype (attribute*)>
59 <!ATTLIST devtype %name.att;
60               comics_prefix    CDATA ""
61               default_inhibit  %bool_no.att;>
62
63 <!ELEMENT attribute EMPTY>
64 <!ATTLIST attribute %name.att;

```

```

65             xmltype  CDATA  "CDATA"
66             default  CDATA  "#REQUIRED"
67             onfree   CDATA  "">
68
69 <!ELEMENT subdevice EMPTY>
70 <!ATTLIST subdevice comics_name CDATA #REQUIRED
71             attribs      CDATA #REQUIRED>
72
73 <!ELEMENT tieto EMPTY>
74 <!ATTLIST tieto %name.att;>
75
76 <!ELEMENT devices ((device|calpulser)*)>
77
78 <!ELEMENT device ((subdevice|tieto)*)>
79 <!ATTLIST device %device.attribs;>
80
81 <!ELEMENT calpulser (tieto*)>
82 <!ATTLIST calpulser %name.att;
83             comics_name CDATA ''
84             downloader  CDATA ''>
85
86
87 <!ELEMENT crates (crate*)>
88
89 <!ELEMENT crate ((subdevice|tieto)*)>
90 <!ATTLIST crate %device.attribs;
91             geosect CDATA #REQUIRED
92             novbd   %bool_no.att;>
93
94
95 <!-- =====
96     The level1 section.
97 -->
98
99 <!ELEMENT level1 ((term|trigmgr|l1ct_thresh|l1ct_refset|l1ct_emcount|
100             l1ct_jetcount|l1ct_ltcountr|l1qual|
101             l1ct_emcountr|l1ct_jetcountr|
102             l1ct_emquad|l1ct_jetquad)*)>

```

```

103 <!ATTLIST level1 n_expogroups CDATA #REQUIRED
104                 n_bits      CDATA #REQUIRED>
105
106
107 <!ELEMENT trigmgr (term*)>
108 <!ATTLIST trigmgr class      (l1ctt|l1fpd|l1fps|l1lum|l1muo) #REQUIRED
109                 %termrange16.attrs;
110                 term16      CDATA ""
111                 device      CDATA #REQUIRED>
112
113 <!ELEMENT term EMPTY>
114 <!ATTLIST term %name.att;
115                 number CDATA #REQUIRED>
116
117 <!ELEMENT l1ct_thresh EMPTY>
118 <!ATTLIST l1ct_thresh class (l1emetsum|l1hdetsum|l1totetsum|l1misspt) #REQUIRED
119                 %termrange.attrs;>
120
121 <!ELEMENT l1ct_refset EMPTY>
122 <!ATTLIST l1ct_refset n_sets CDATA #REQUIRED
123                 class (l1em_refset|l1hadveto_refset|
124                 l1jet_refset|l1lt_refset)
125                 #REQUIRED>
126
127 <!ELEMENT l1ct_emcount EMPTY>
128 <!ATTLIST l1ct_emcount refsetno CDATA #REQUIRED
129                 %termrange.attrs;>
130
131 <!ELEMENT l1ct_jetcount EMPTY>
132 <!ATTLIST l1ct_jetcount refsetno CDATA #REQUIRED
133                 %termrange.attrs;>
134
135 <!ELEMENT l1ct_ltcount EMPTY>
136 <!ATTLIST l1ct_ltcount refsetno CDATA #REQUIRED
137                 count      CDATA #REQUIRED
138                 number     CDATA #REQUIRED>
139
140 <!ELEMENT l1ct_emcountr EMPTY>

```

```

141 <!ATTLIST l1ct_emcountr refsetno CDATA #REQUIRED
142         count CDATA #REQUIRED
143         number CDATA #REQUIRED
144         region (n|c|s) #REQUIRED>
145
146 <!ELEMENT l1ct_jetcountr EMPTY>
147 <!ATTLIST l1ct_jetcountr refsetno CDATA #REQUIRED
148         count CDATA #REQUIRED
149         number CDATA #REQUIRED
150         region (n|c|s) #REQUIRED>
151
152 <!ELEMENT l1ct_emquad EMPTY>
153 <!ATTLIST l1ct_emquad count CDATA #REQUIRED
154         number CDATA #REQUIRED
155         quadrant (1|2|3|4) #REQUIRED>
156
157 <!ELEMENT l1ct_jetquad EMPTY>
158 <!ATTLIST l1ct_jetquad count CDATA #REQUIRED
159         number CDATA #REQUIRED
160         quadrant (1|2|3|4) #REQUIRED>
161
162 <!ELEMENT l1qual EMPTY>
163 <!ATTLIST l1qual %name.att;
164         number CDATA #REQUIRED>
165
166
167 <!-- =====
168         The level2 section.
169     -->
170
171 <!ELEMENT level2 (l2ppcrate*,l2global,l2pp*,l2tool*)>
172
173 <!ELEMENT l2global (l2parm*)>
174 <!ATTLIST l2global crate CDATA #REQUIRED
175         exe_name CDATA #REQUIRED
176         l2_name CDATA #REQUIRED
177         fail_tooltype CDATA #REQUIRED>
178

```

```

179 <!ELEMENT l2ppcrate (l2input*,l2mbt?)>
180 <!ATTLIST l2ppcrate crate          CDATA #REQUIRED
181                   l2cratename CDATA  ''>
182
183 <!ELEMENT l2mbt (#PCDATA)>
184
185 <!ELEMENT l2pp (l2parm*)>
186 <!ATTLIST l2pp %name.att;
187           crate          CDATA #REQUIRED
188           exe_name       CDATA #REQUIRED
189           l2_name        CDATA #REQUIRED
190           l1qual         CDATA #REQUIRED>
191
192 <!ELEMENT l2parm EMPTY>
193 <!ATTLIST l2parm %name.att;
194           type          CDATA #REQUIRED
195           default       CDATA "#REQUIRED">
196
197 <!ELEMENT l2input EMPTY>
198 <!ATTLIST l2input number CDATA #REQUIRED
199           crates CDATA #REQUIRED>
200
201 <!ELEMENT l2tool (l2parm*)>
202 <!ATTLIST l2tool %name.att;
203           l2name        CDATA ""
204           pp            CDATA ""
205           maxinstances  CDATA "-1">

```

B Run Mode Examples

This section gives example trigger configurations for the run modes enumerated in Sec. 10.4.

B.1 External

```
1 <?xml version="1.0"?>
2 <!DOCTYPE configuration SYSTEM "trigger_config.dtd">
3
4 <!-- Test the various running modes.
5     'external': no readout from the detector; data fed into the C/R
6     from an external source, with a hardwired stream ID.
7
8     PDAQ: None
9     SDAQ: None
10    Calib: None
11
12    -->
13
14 <configuration name="mode-external" version="1.0" autopause="no">
15
16     <download name="allcrates">
17         <Cal_ADC_Crate      name="ecnse"/>
18     </download>
19
20     <stream name="daq_test" relrate="1.0" number="9999"/>
21 </configuration>
```

B.2 FW-Only

```
1 <?xml version="1.0"?>
2 <!DOCTYPE configuration SYSTEM "trigger_config.dtd">
3
4 <!-- Test the various running modes.
5     'fw-only': Generate L1 accepts, but force L2 rejects.
```

```

6             Crate testing, etc.
7
8             PDAQ: L1 only
9             SDAQ: None
10            Calib: None
11
12            -->
13
14            <configuration name="mode-fw-only" version="1.0">
15
16                <download>
17                    <Cal_ADC_Crate      name="ecnse"/>
18                </download>
19
20                <expogroup name="eg" readout="ecnse">
21                    <l1trigger name="l1bit1">
22                        <l1termlist>
23                            <l1specterm name="fastz"/>
24                        </l1termlist>
25                    </l1trigger>
26                </expogroup>
27
28                <!-- No stream, since this configuration sends no data to the host. -->
29            </configuration>

```

B.3 PDAQ

```

1 <?xml version="1.0"?>
2 <!DOCTYPE configuration SYSTEM "trigger_config.dtd">
3
4 <!-- Test the various running modes.
5     'pdaq': Use L1, L2, L3.
6             Normal running with primary DAQ system.
7
8             PDAQ: Full
9             SDAQ: None

```

```

10         Calib: None
11
12     -->
13
14 <configuration name="mode-pdaq" version="1.0">
15
16     <download>
17         <Cal_ADC_Crate      name="ecnse"/>
18     </download>
19
20     <trigdef>
21         <expogroup name="eg" readout="ecnse">
22             <l1trigger name="l1bit1">
23                 <l1termlist>
24                     <l1specterm name="fastz"/>
25                 </l1termlist>
26             <l2trigger name="l2bit1">
27                 <l3trigger name="l3bit1"/>
28             </l2trigger>
29         </l1trigger>
30     </expogroup>
31     <triglist>
32         Triglist text.
33     </triglist>
34 </trigdef>
35
36     <stream name="daq_test"/>
37 </configuration>

```

B.4 Parasitic-SDAQ

```

1 <?xml version="1.0"?>
2 <!DOCTYPE configuration SYSTEM "trigger_config.dtd">
3
4 <!-- Test the various running modes.
5     'parasitic-sdaq': Read out using sdaq, but with triggers provided

```

```

6             by another run.
7             E.g., read out monitoring data from another run.
8
9             PDAQ: None
10            SDAQ: Parasitic
11            Calib: None
12
13            -->
14
15            <configuration name="mode-parasitic-sdaq" version="1.0">
16
17                <download>
18                    <Cal_ADC_Crate          name="ecnse"/>
19                </download>
20
21                <sdaq type="sdaqtype" readout="ecnse" parasitic="yes"/>
22
23                <stream name="daq_test"/>
24            </configuration>

```

B.5 FW-SDAQ

```

1 <?xml version="1.0"?>
2 <!DOCTYPE configuration SYSTEM "trigger_config.dtd">
3
4 <!-- Test the various running modes.
5      'fw-sdaq': Read out sdaq, triggered by the TFW.
6                E.g., tracking calibration.
7
8             PDAQ: L1 only
9             SDAQ: Parasitic
10            Calib: None
11
12            -->
13
14            <configuration name="mode-fw-sdaq" version="1.0">

```

```

15
16 <download>
17   <Cal_ADC_Crate      name="ecnse"/>
18 </download>
19
20 <expogroup name="eg" readout="ecnse">
21   <l1trigger name="l1bit1">
22     <l1termmlist>
23       <l1specterm name="fastz"/>
24     </l1termmlist>
25   </l1trigger>
26 </expogroup>
27
28 <sdaq type="sdaqtype" readout="ecnse"/>
29
30 <stream name="daq_test"/>
31 </configuration>

```

B.6 PDAQ-SDAQ

```

1 <?xml version="1.0"?>
2 <!DOCTYPE configuration SYSTEM "trigger_config.dtd">
3
4 <!-- Test the various running modes.
5     'pdaq-sdaq': Run both pdaq and sdaq.
6         E.g., normal running, reading out monitoring info via sdaq.
7
8     PDAQ: Full.
9     SDAQ: Parasitic
10    Calib: None
11
12 -->
13
14 <configuration name="mode-pdaq-sdaq" version="1.0">
15
16 <download>

```

```

17     <Cal_ADC_Crate      name="ecnse"/>
18 </download>
19
20 <trigdef>
21     <expogroup name="eg" readout="ecnse">
22         <l1trigger name="l1bit1">
23             <l1termmlist>
24                 <l1specterm name="fastz"/>
25             </l1termmlist>
26             <l2trigger name="l2bit1">
27                 <l3trigger name="l3bit1"/>
28             </l2trigger>
29         </l1trigger>
30     </expogroup>
31     <triglist>
32         Triglist text.
33     </triglist>
34 </trigdef>
35
36 <sdaq type="sdaqtype" readout="ecnse" only_streams="sdaq_stream"/>
37
38 <stream name="daq_test"/>
39 <stream name="sdaq_stream"/>
40 </configuration>

```

B.7 SDAQ

```

1 <?xml version="1.0"?>
2 <!DOCTYPE configuration SYSTEM "trigger_config.dtd">
3
4 <!-- Test the various running modes.
5     'sdaq': Trigger and read out using sdaq.
6
7     PDAQ: None
8     SDAQ: Full
9     Calib: None

```

```
10
11  -->
12
13 <configuration name="mode-sdaq" version="1.0">
14
15   <download>
16     <SMT_Crate      name="smt0_0"/>
17   </download>
18
19   <sdaq type="sdaqtype" readout="smt0_0"/>
20
21   <stream name="daq_test"/>
22 </configuration>
```

C State Diagrams

