KLOE Memo n. 95
June 1997

Guidelines for the KLOE Run Control program and its User Interface

A. Doria, M. L. Ferrer, E. Pasqualucci

1 Runctl and Operator

The control of the Data Acquisition System is made through two complementary programs, both running on the run control node. One is the actual Run Control program (*Runctl*), that implements the transitions of a finite state machine between the run states and sends for each transition the opportune commands to all programs taking part to the DAQ. The other is the graphical interface the person on shift directly interact with; this *Operator* program implements buttons and menus to activate commands, displays relevant parameters, displays of messages, warnings and alarms from the system and checks some configurations.

Only one couple *Operator* - *Runctl* can be activated at a time in a normal run. Both programs are written according to the KLOE process template, so they are capable of sending and accepting commands and of accessing all process variables in the standard way.

The minimal set of parameters the interface displays comprises: number and state of the run, event number, DAQ rate, start time and end time; modifiable parametes will also appear in the interface such as storage destination, maximum number of events per run, run mode. In particular, 4 run modes are forseen: *normal*, *partitioned*, *calibration* and *test* run. The run number will be marked with different flags (let's say P, C and T) in case the run is not in normal mode. The default mode is *normal* if storage devices are enabled, otherwise it is *test*.

The *Operator* program increments and saves in a file the run number at the beginning of each new run; at the startup of the program the run number is loaded from this file. Another special file *mapfile.dat* holds a static map of the network, needed at startup time to the Operator, to reset all nodes and to start *cmdsrv* on each node.

The Run Control is in charge of building the actual map of the DAQ network at startup and every time this is needed (e.g. at each initialization) and to activate at startup the processes on each node (through a command to the cmdsrvs). The *Operator* gets the map from *Runctl*, displays it and, for each node of the map, gets by the command server the list of processes running on the node. A check that all needed process are active is performed from time to time.

2 Run states and Run Control commands

The following four states are possible for the run; the arrows indicate which transitions are allowed and the commands that cause the corresponding transition:

1) UNDEFINED_STATE —–`INIT`—→ INIT_STATE

2) INIT_STATE —–`INIT,START`—→ INIT_STATE, RUNNING_STATE

3) RUNNING_STATE —–`PAUSE,END`—→ PAUSE_STATE, INIT_STATE

4) PAUSE_STATE —–`RESUME,END`—→ RUNNING_STATE, INIT_STATE

The UNDEFINED_STATE is the one the processes stay before the initialization or when an operation fails.

A number of buttons of the *Operator* Interface allow to send run control commands.

- `INIT`: *Operator* updates the displayed parameters, asks for a confirmation of the run mode, sends to *Runctl* the `INIT` command. A Self Test can be implemented at `INIT`. Receiving `INIT`, *Runctl* makes the network map, sets configuration parameters to all processes as needed (e.g. calibration files), and forwards `INIT` to all the nodes. Commands sent to a node are addressed to the command server, that locally dispatches the command to the processes.
- `START`: *Operator* updates the run number and writes it in a special file, asks for a confirmation of the run parameters, requires a comment for the run and sends `START` to *Runctl*. *Runctl* forwards `START` to all DAQ processes, in a defined sequence, from the bottom to the top of the chains. When the start is acknowledged by all processes, the trigger can be activated, through a command to the the process managing the Trigger Supervisor (T.S.).
- `PAUSE`: is sent to *Runctl*, that communicates to the T.S. to suspend the trigger.
- `RESUME`: is sent to *Runctl*, that communicates to the T.S. to restart the trigger.
- `END`: is sent to *Runctl*, that gets by the T.S. the last trigger number and communicates this process to stop the trigger. Then *Runctl* sends `END` to the collectors,

with the last trigger number as a parameter. The collectors, according with the last trigger number, have to flush the FIFO and to put an "end-of-run" event in the chain, causing the stop of the following processes.

A button of `START COLD` will by implemented, to perform `INIT+START` functions. Another button of the interface allows the startup of the DAQ processes, on all the nodes listed in a special file, while a button can be supplied to check that all needed processes are running. The *Operator* needs a sort of "console" to display messages arriving from the Run Control and eventually from other processes. A `WRITE` command will be implemented for the *Operator*, to output the messages on this console. The *Operator* also allows to monitor the DAQ rate, for example getting this parameter by the processes choosen from a menu.

The only source of errors we take into account at the moment is a Busy/Halt (longer then a timeout) in the trigger, signalling problems in the chain; the T.S process polls on it, sending an ERROR message to *Runctl.* that requires the stop of the run. Another possibility is to implement in the *Runctl* a function of "get-trigger-status" to monitor with the minimum delay an ERROR detected by the T.S.. We will evaluate which choice is more convenient. In any case some funtionalities have to be implemented to locate the origin of the error.