

DØ Run II Online Examine Framework Design and Requirements

J. Yu & J. Kowalkowski

DØ Note # 3578

We present a design of the DØ Run-II Examine framework. We propose the Examine framework design to be the baseline design of any user interactive framework. We also lay out various requirements for Run II DØ online Examine package. By no means does this document detail the specific needs of each of the different types of Examines programs. This document only attempts to define a baseline structure for the Examine executables and GUI.

1 Introduction

Online Examine is a program which is used by the shifters or each individual detector group personnel to monitor various sections of the detector and data integrity. Examine provides tools for monitoring the experiment, via histograms compared to the best known reference sets and event displays.

The current approach for Run-II Examine is to have two separate pieces, a GUI and an executable, working hand-in-hand. The Examine executable reconstructs events, calculates relevant physical quantities, and fills histograms. The GUI allows the user to interact with the given Examine executable and display histograms being filled or events being reconstructed by the executable. The communication between the two pieces is carried out by either the DØ Client/Server package or a CORBA compliant package depending on the choice of the GUI language.

Examine must provide enough freedom for the users to select the events. One must be able to select events directly from the DAQ by:

- One or more data stream names w/ prescale factors
- One or more Level 1 trigger names w/ prescale factors
- One or more Level 2 trigger names w/ prescale factors
- One or more Level 3 trigger names w/ prescale factors
- Any combinations of the above

The lists of the above selections should be configurable in a selected RCP file.

Each Examine executable will have its own event selection RCP file. In order to ensure this functionality during the event selection, the Data Distributor (DD) is designed to be aware of the associations between the selections and each individual Examine executable (see Ref. [1] for DD design requirements).

The user presentation portion (GUI) of the program must be fool-proof and automatic to minimize human manual intervention. For example, all histogram operations - display, reset, update, declare, etc - must be push button controlled.

2 Design Philosophy

The Examine framework must not only utilize as much of the existing interfaces/hooks and packages in the offline batch framework as possible but must also be designed with interactive offline framework in mind. Since in principle, the Examine framework is a generalized version of the interactive offline framework, we must try to keep the design flexible enough to easily add any unforeseen functionalities that may be required by the offline interactive framework.

The executables must be completely RCP driven the same way as the current batch offline framework is. The packages and interfaces listed in the framework RCP file are used in building the given Examine program. In addition, the Examine framework should be easily expandable by utilizing free standing interfaces as much as possible.

The specific packages that should be included in an Examine executable must be determined according to the responsibilities of the type of an Examine. The specific, detailed needs of individual Examines should be provided by the persons in charge of authoring the particular Examine. Thus, we will not include the specific discussions in this note, although they will have to be part of this note in the near future. Since the event data buffering and handling is discussed in detail in Ref. [1], we will only concentrate on the Examine framework design, executable and GUI representation requirement portion in this note.

3 Run - I Examines

In Run-I, there were five different types of Examine executables and were a minimum five Examine processes running at all times in the normal shift environment. Table 1 summarizes the Examine programs and their responsibilities. The responsibilities in table 1 are the determining factors of the

Table 1: Types and responsibilities of Examines in Run -I.

Examine	Responsibility
Detector Examines	
Calorimeter	Monitor CC, EC, and ICD
Muon	Monitor WAMUS and SAMUS systems
Tracker	Monitor CDC and FDC systems
Global Examine	Reconstruct express line events online, fill reconstructed quantities in histograms, and provide event displays for shifters.
Trigger Examine	Mostly provide software trigger performances and rates for each bit.

Examine work-horse CPU and memory capacities, as well as the packages needed to carry out the given responsibility.

All the different Examines were separate executables running on vax stations. The user interactive portion of the Examine histogram display was carried out via PAW, accessing histograms stored in the shared memory on

the same physical machine. The scheme was that one starts data processing of the given executable, followed by starting a PAW session to connect to the shared memory to display histograms while they are filled.

4 Run - II Examine Configurations

Since the detector has been upgraded from Run-I configuration, there are more parts of the detector to be monitored. This will increase the number of detector monitoring Examines. Table 2 summarizes the types of Examines and their responsibilities that are needed for Run-II.

Table 2: Types and responsibilities of Examines in Run-II.

Examine	Responsibility
Detector Examines	
Calorimeter	Monitor CC, EC, ICD, FPS, CPS systems
Muon	Monitor WAMUS and SAMUS systems
Tracker	Monitor SMT, CFT, Solenoid(?) systems
Beam	Monitor LØ and FPD systems
Global Examine	Reconstruct events in selected streams and fill in reconstructed quantities in histograms and at the same time provide event displays for shifters.
Trigger Examine	Mostly provide software trigger performances and rates for each bit.

The current conceptual design of the Examine work-horse machines is most likely a collection of Linux PC nodes for analysis, histogramming, and event reconstruction. On the other hand, the GUI portion of Examine would likely be run under Windows NT. However, both the executable and the GUI portion of a given Examine should be written such that the program does not depend on the choice of the platforms or operating systems.

5 Existing Offline Batch Framework

Since the offline framework is essentially a subset of that of the online Examine, it would make sense that the Examine framework would fully utilize the existing offline batch framework. In addition, the functionality required by user interactive offline framework shares many of its necessary features of the Examine framework. Thus it is useful to design the frameworks to share as much code as possible. In this section, we will discuss the existing offline framework and what interfaces or packages that need to be added to satisfy the Examine framework requirements.

The current offline framework exists only in non-interactive manner. The framework provides several interfaces which are essentially abstract classes that modules in the given package implement using inheritance. The user's package defines and implements various interfaces that get executed at each stage of the framework event processing flow (see Ref. [2]).

The offline framework interfaces consist of two large categories: interfaces that make up the main event processing flow and the interfaces that are free-standing and independent of the flow.

5.1 Interfaces in the Offline Framework Flow

There exists a set of interfaces in the event processing flow. A given package can have multiple interfaces (hooks) implemented depending on the needs of an analysis. Each interface specifies the stage of execution within the event processing loop.

The existing interfaces/hooks in the event processing flow are:

1. "Generator" + "Decide"
2. "Build"
3. "Filter"
4. "Process"
5. "Analyze"
6. "Dump"

The sequence listed above is a recommended framework flow where execution of implemented interfaces of a package occurs.

5.2 Free-standing Hooks

The free standing interfaces/hooks are the interfaces that can be executed at any time, independent of the main event processing flow. The implemented interfaces can be called at the specified stage of the program independent of the flow. A more detailed description of these interfaces can be found in Ref. [2].

Currently there are three free-standing interfaces available in the batch framework that a user can implement. The hooks are:

- “RunInit”
- “RunEnd”
- “JobSummary”

6 Design of the Examine Framework

As discussed in section 2, the Examine Framework can be viewed as a generalized interactive offline framework. In order to understand the requirements for the Examine better, it is necessary to provide a clear picture of the underlying architecture or the skeletons of the framework. Thus, we will describe the Examine framework data and message flow architecture, given the current design, in this section. Since the current design can be subdivided into two sections based on functionality, each section, for the most part, is independent. We will describe the design of these two different sections separately. The two sections are **Data Processing** and **Histogram Presentation**.

Even though the descriptions of the two functions are done separately in this section, the final look of the user interactive control panel does not necessarily have to be two independent windows. In fact, we currently would like to combine the two control panels into one GUI window as shown later in Fig. 3 in section 15.

6.1 Data Processing Control

Figure 1 shows the baseline design of the Examine **Data Processing** message flow. The functionality of the box at each stage is as follows:

Nov. 16, 1998, JBK & J. Yu

Examine Processing Data and Message Flow

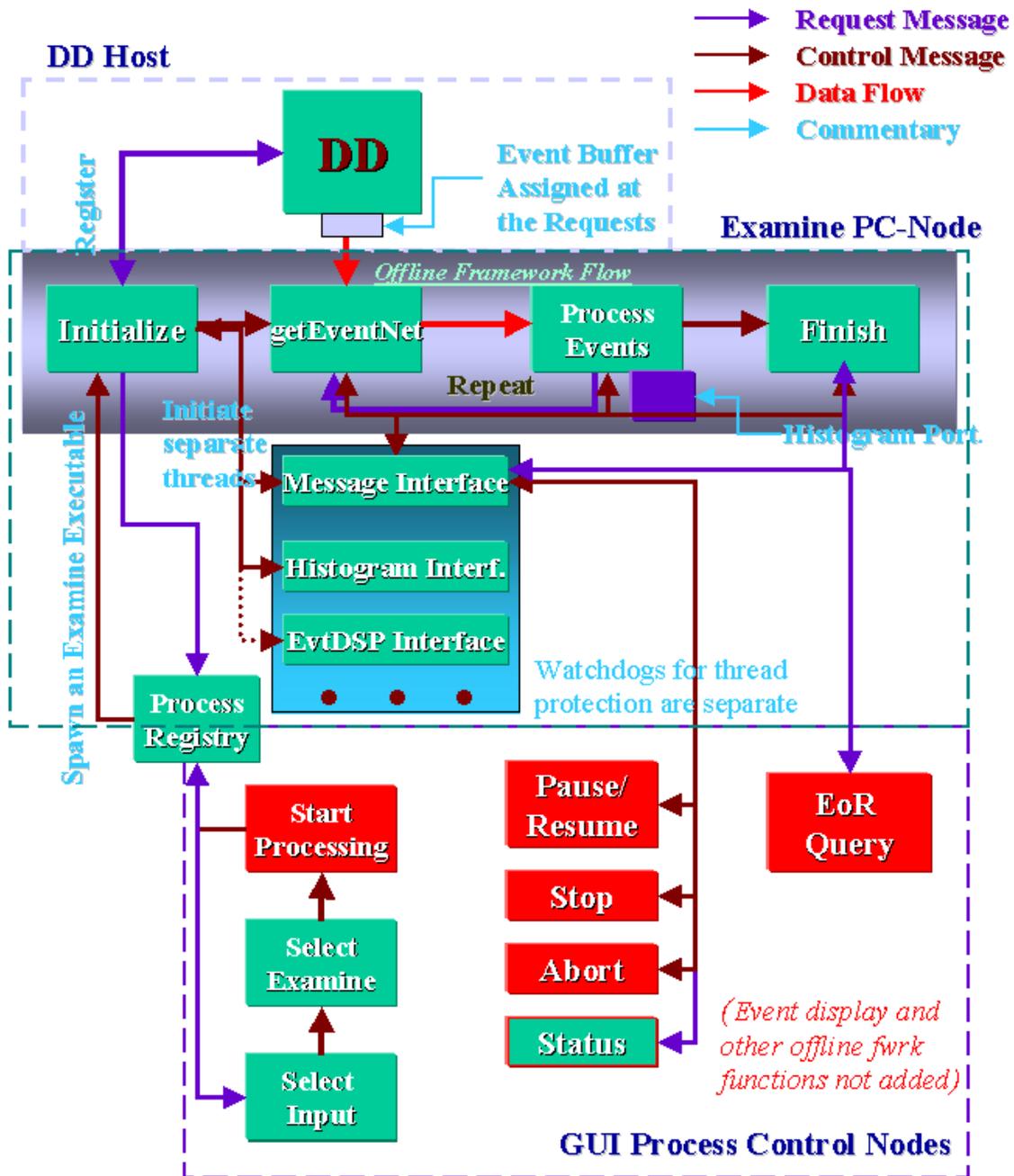


Figure 1: A message and data flow diagram for data processing portion of the Examine control panel.

1. **Select Input:** Present the choices of the following input sources
 - (a) DAQ
 - (b) File
 - (c) List of the running Examine Processes

If input 1a or 1b is chosen, one steps to the Examine type selection.

The “list of running Examine Processes” is obtained from the “Process Registry”. Selecting one of the running processes will allow the user interface (GUI) to connect to the given *histogram and/or event display ports* but not to the *message port* which governs the process control message flow.

2. **Select Examine:** This step selects the type of Examine executable to run. The different type of Examine will have different combinations of packages defined in the framework RCP. The list of Examine packages is kept in the *Process Registry*, being distinguished by their names and corresponding RCP files (histogram). The *Process Registry* is necessary to minimize duplication of the same type of Examines with the same list of histograms. The user will make the final decision whether to run one’s own Examine or not.
3. **Start Processing:** This stage sends a request for a specific Examine executable to be initiated to the *Process Registry*. The list of RCP files are passed to the *Process Registry* to tag the Examine process for the future queries by other users.
4. **Process Registry:** This program looks at the Examine PC array and selects a machine, and spawns the requested Examine executable. The spawned executable informs the **Registry** of its various communication ports. The **Registry** relays this information back the Graphical User Interface so that the GUI can attach to the running Examine process. Since the **Process Registry** plays an important role in the current design, we will have more detailed discussion in section 7.
5. **Initialization:** At this stage, the following should occur
 - (a) Start independent threads:

- Message Interface
 - Main Framework Watchdog Interface
 - Thread Watchdog Interface
- (b) Construct a “Generator” package that uses event selection to attach itself to the DD.
 - (c) Construct packages required for histogram display, event display, etc. These are the communication interfaces in Fig 1.
 - (d) Construct all the reconstruction and analysis packages needed by the Examine executable. Histogram and event display ports are created at this stage.
 - (e) Registers all the created communication ports to the *Process Registry*.
6. **getEventNet**: Use the “Generator” package to get an event from source. The I/O packages in the framework (need a new one called *ReadEventNet* for direct DAQ reception) get the events from the DD buffer and pass the event to the next stage. **ReadEvent** and **NewEvent** are probably the corresponding offline framework packages. There should be some methods to keep the statistics and the status of the package for the status report.
 7. **Process Event**: This stage reconstructs an event to the depth relevant to the given Examine type and fills histograms. This process also keeps the statistics and a status flag. The offline framework interfaces that could be involved in this step are:
 - “Build”
 - “Filter”
 - “Process”
 - “Analyze”
 - “Dump”
 8. The steps 6 and 7 repeat till one of the followings occur
 - (a) DD sends the end of run message in the buffer.

- (b) The *Message Interface* receives one of the control messages - stop, abort, pause.

The actions *Stop* and *Abort* cause the executable to step to the **Finish**.

9. **Finish:** When one of the above control conditions occurs, the program should gracefully go to the “Finish” stage. The process will, then, end doing the following cleanups.

- (a) Query the user for:

- Continuing to run, retaining all histogram entries.
- Continuing to run, writing out the current set of histograms in the memory to a disk file (“*Flush*”) and resetting histograms.
- Terminating the program, writing out histograms and other open files, and closing communication ports.
 - Destroy the “Generator” package. Act of doing so closes up the files (if the input was file) or detaches from the DD.
 - Destroy all the reconstruction and the analysis packages. Act of doing so dumps all the histograms into the relevant files and deletes the histogram port.
 - Report back to the user interface of ending the executable process.
 - Notify the *Process Registry* that the program is ending.
 - Terminate the *Message Interface* and various other communication interfaces.
 - Exit.

10. The user interface then either terminates itself, destroying the GUI, or prompt the user for continuation of the given process.

7 The Process Registry

There should be one program that runs on the Examine PC farm that knows all the processes currently running on the farm and the corresponding machines. The Examines are known to the *Process Registry* by their type names and RCP version numbers.

7.1 Exchanged Message to/from the Registry

The messages that should be exchanged between external processes and the registry are :

- “Query” - Examine Name, RCP versions \Rightarrow “Results(xxx,yyy,zzz)”
- “Start” - Examine Name, RCP names and versions \Rightarrow “ACK(xxx,yyy,zzz)” or “NACK(xxx,yyy,zzz)”
- “GetList” - Input source \Rightarrow “Results(xxx,yyy,zzz)”

7.2 Entries to the Process Registry

The Entries to the ProcessRegistry are :

- Process Name
- Location:
 - Host Name
 - Message Interface Port
 - Histogram Interface Port
 - Event Display Interface Port
 - Other Interface Port
- Examine Type
- RCP information and identification

It is probably a good idea to manage the ports as a list, where each entry in the list contains port names and the numbers. For example, we can keep the list of three port names and the corresponding port numbers as follows:

- (“Message”, 1117)
- (“Histogram”, 22223)
- (“Watchdog”, 13245)

7.3 Responsibilities of the Registry

The responsibilities of the process registry are:

1. Receive a request to start a new Examine from an Examine User interface.
2. Assign an Examine Farm machine for the job.
3. Spawn the requested Examine on the assigned machine.
4. Receives information from the Examine process regarding port names and numbers.
5. Send all port information back to the user interface.
6. Keep information of all Examine processes currently running, distinguishing them by their type names and associated RCP version numbers.
7. Provide the list of running Examine processes in the Examine Linux cluster, in response to queries from an Examine user interface.
8. Provide all the port information for the given Examine to the user interface upon request.
9. Keep the log of which machine ran which Examine with the corresponding time stamps.
10. Ping all the Examine processes periodically to check for the stale processes. Trigger process restarts.

8 Responsibilities of the *Messaging Interface*

1. Wait for messages from the user interface (GUI).
2. Respond to *status report requests*.
3. Respond to *Control Requests* in a timely fashion.

4. Respond to the “ping” requests from the *Process Registry* for stale process check.
5. Deliver framework *Dump* interface information.

9 Responsibilities of the *Histogram Interface*

1. Record histogram port information.
2. Wait for messages from the *Histogram Control* portion of the user interface.
3. Provide available histogram categories and individual histogram names to the GUI.
4. Send the appropriate histograms to the GUI when requested.
5. When an update is requested, continuously send the given set of histograms to the GUI.
6. Send control messages related to histogram saving and resetting to the relevant package.

10 Framework Watchdog

There should be a `MainThreadWatchDog` that intercepts any possible problem with the main framework program and restarts the executable automatically. When the program restarts, the old histograms that were being filled should be there in the memory so that to the user everything should be as transparent as possible. Jim thinks that it is possible to use the system “Memory Map” facility. The possible procedure is to have the program open a file, using the virtual memory in the system, and update the memory whenever the histograms are updated. If for some reason the executable dies, the watchdog can still restart the process, read in the memory map, and start back where it left out before the crash. Jim thinks that this facility is fast and is relatively easy to implement into the framework as one of the separate threads.

The main thread should have protection built in so that it knows how many processes are connected to itself. It should know when the last time it had been queried, and depending on the condition, throw an alarm for a possible *Run-away* process.

11 Responsibilities of the *Thread Watchdog*

The child ThreadWatchDog is also a separate thread that wakes up periodically in the middle of the process and checks the response of the child threads (message, histogram, event display, and watchdog interfaces). If the threads do not register for a certain period of time, the *Watchdog* stops the thread and restarts.

1. Wake up periodically for a very short time to check the staleness of all child threads.
2. If there is a staled child thread, it removes the thread and restart it using the same port.
3. It must know all the port numbers used by the child threads and the association between the thread and the port numbers.
4. Needs to send log messages of its actions to the central alarm/log server.

12 Histogram Presentation Control

One of the independent threads that is initiated at the beginning of an Examine process is the “*Histogram Interface*”. This histogram interface handles the communication between the histogram control portion of the Examine GUI and the executable. Since the histogram control has an intimate relationship with the choice of the physics analysis tool (PAT), the online group has concentrated on choosing a PAT. After a few sessions of demonstrations of HistoScope and ROOT, we have made a decision to choose the ROOT as our ultimate PAT platform.

The histogram control system’s functionality, however, is independent of the choice of PAT. Figure 2 shows the message flow on the histogram control portion of the GUI-executable interaction. The requests from the GUI for

Nov. 16, 1998, J. Yu

Examine Histogram Control Flow

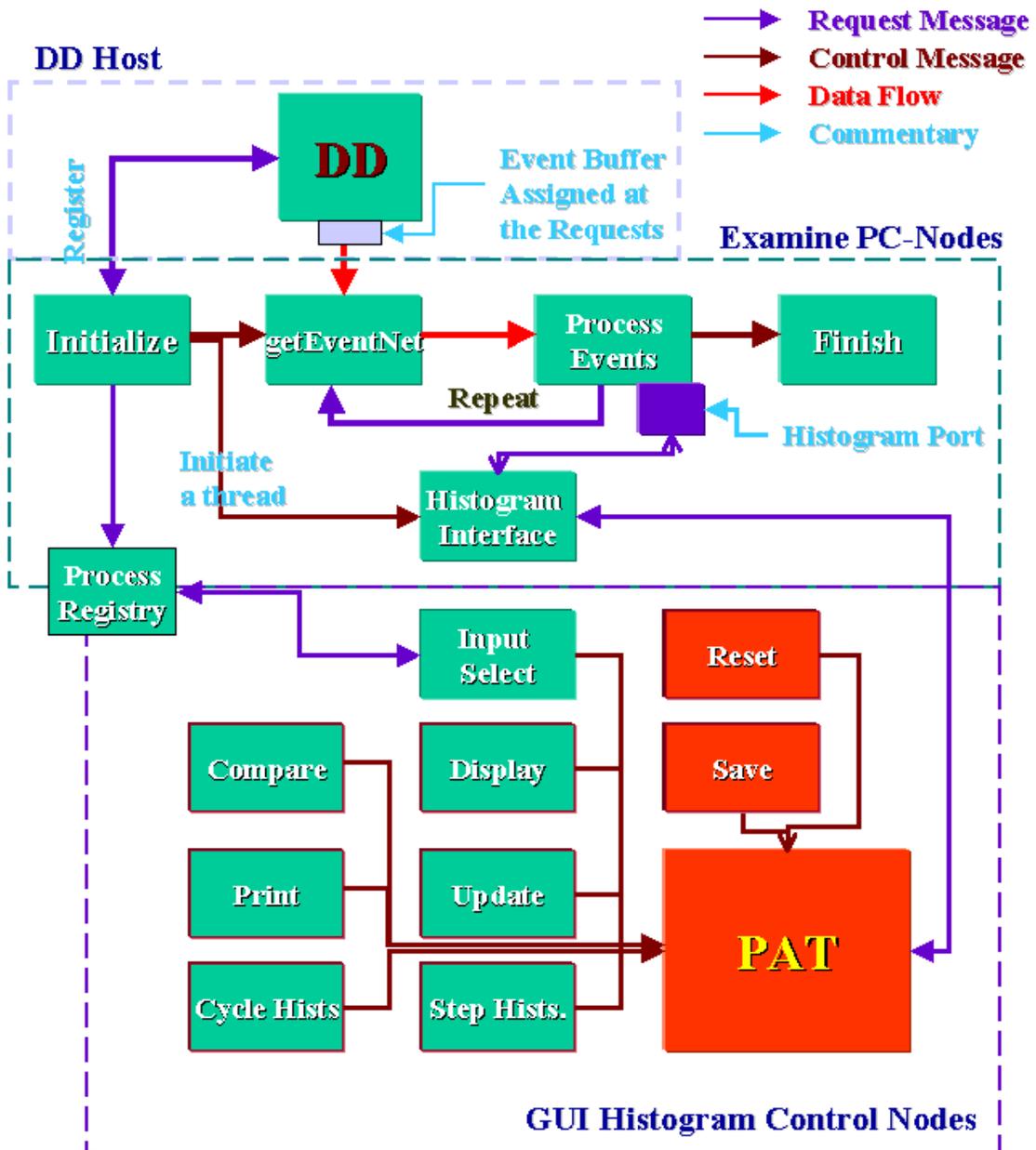


Figure 2: A message and data flow diagram for histogram display portion of the Examine control panel.

histogram must be mapped to corresponding commands in the given PAT. The histograms that are accumulated in the memory port must be transported within the given PAT framework to the display, via the “*Histogram Interface*”. Both the HistoScope and ROOT browsers have abilities for displaying histograms transported over the network.

13 Online Addition to Offline Framework

Based on the discussions in sections 6 through 12, we can identify some changes to the existing offline batch framework. The changes are to the methods of the package base classes.

13.1 Subdivision of the Process Space

Table 3 shows subdivisions of the process space while an Examine executable is running. It should be noted that the process space split into two main parts; 1) the main thread which does reconstruction and histogramming and 2) independent threads which provide a means for external processes to interact asynchronously.

Table 3: Subdivision of the Examine process space. The mainthread column lists public methods of the framework base class.

Child Threads	Main Thread
Message Interface	Framework Control
Histogram Interface	<i>findPackage(Name)</i>
ThreadWatchDog	<i>resume()</i>
MainThreadWatchDog	<i>stop()</i>
EventDisplay Interface	<i>stepEvent()</i>
etc...	<i>stepPackage()</i>
	<i>pause()</i>
	<i>etc....</i>

13.2 Framework Class

The framework class must support the following methods.

- *findPackage(Name)*: Given a name of the package, return a pointer to the package.
- *pause()*: Stop the event processing as soon as possible, before the next framework package interface processing starts.
- *resume()*: Resume processing from the current point.
- *stop()*: Cause the framework to jump to **Finish** stage that is explained in section 6.1, as if it has depleted the event source.
- *stepEvent()*: Step through the framework process one full event and stop before starting the next event.
- *stepPackage()*: Step through one framework package interface and stop.
- *Flush()*: Cause the framework to write out data and histograms to open files.
-

These methods will be executed by the *Message Interface Thread*. By no means are the above methods meant to be the complete set of the public methods. The design of the framework is such that it should be easy to implement methods as needed.

13.3 Package Base Class

Each individual Examine package must inherit from this *Package* base class. The base class should require the following methods to be written in subclasses:

- *statusReport(Ostream, WhereToReport)*
- *getRCP()*
- *reInitialize(RCP)*

- *Activate()*
- *deActivate()*
- *Flush()*
- *etc...*

13.4 Additional Framework Hook

There may be an additional hook, *OutputStage*, necessary to insure full reconstruction of an event before delivery to the Event Display. To be more specific, an event display I/O package would deliver an event to the Event Display GUI during the call to the *OutputStage* interface.

14 Examine Executable Requirements

In this section, we will discuss the requirements which are related to Examine executable responsibilities.

- The executable should be platform independent, although the most likely platform for the executable to run on is linux.
- The GUI and the executable should be able to start up either on the same machine or on two physically different machines under two different operating systems.
- The executable must be connected to the central alarm server.
- The executable should utilize the offline framework interfaces and packages.
- Choice of a subsystem Examine must be possible within the given Examine executable. For instance, one must be able to select only the SMT system within the detector Examine without having to run through all the other detector systems.
- The selection of the desired subsystems in the Examine must be RCP controlled. In other words, all the control functions of the executables must be RCP driven, following the offline framework scheme.

- The packages involved in reconstruction and analyses must follow offline framework guideline, which is based on RCP files.
- The packages must provide public methods, listed in section 13.2, for free-standing interfaces to communicate with.

The requests from client/GUI to :

- start/stop/finish processing
- Pause/Resume processing
- Terminate/Abort processing

must be provided in the packages as public methods that can be accessed via the free-standing “*Message Interface*”.

- Communicate with the it Process Registry and respond to various inquiries.
- Book standard and additional user selected sets of histograms, defined in the RCP files at the start of processing,
- Fill standard sets of histograms.
- Perform appropriate level of reconstruction to fill histograms. The depth of reconstruction depends strongly on the responsibility of the given Examine. For example, the global Examine will require a full reconstruction while the calorimeter Examine would require the digitized pulse heights only with some reconstructed information at times.
- Must be able to save histograms into a file, when requested.
- Read in Run Control Parameters (RCP) that include the name of the event selection file, discussed in the introduction. *The RCP must have, at the minimum:*
 - *Reference histogram filename*
 - *Filename for event selection conditions*
 - *Choice of event selection (stream/trigger Names)*
 - *etc.*

- Pass the requested selection conditions to the Data Distributor for buffer registration and allocation.
- Attach itself to the assigned buffer or queue for event reception from the DD.
- Reconnect automatically to the DD if the connection is lost. Retry connecting to DD for a given period and prompt the user of the connection problem.
- Receive messages on run transitions from the DD and pass the messages to GUI (ie., to the user for the transition, via the “Message Interface”) and take appropriate action.
- Allow many connections for histogram access, but keep the shifter GUI as the only controller of the histogram reset, delete, and declare manipulation.
- Should be able to accept a data filename as an input source and perform analyses using the file.

15 Examine GUI Requirements

In this section, we will list the requirements for Examine GUI representations. Figure 3 shows a conceptual design of an Examine GUI window.

- Must run on any platform.
- Must know how to connect to the “*Process Registry*”.
- Must have an input selection drop-down menu with **DAQ** or **File** as the data source entries.
- Must provide the list of currently available Examine processes for Histogram access.
- Must provide the list of possible Examine executables.
- Must be able to request a start of an Examine executable to the “*Process Registry*”.

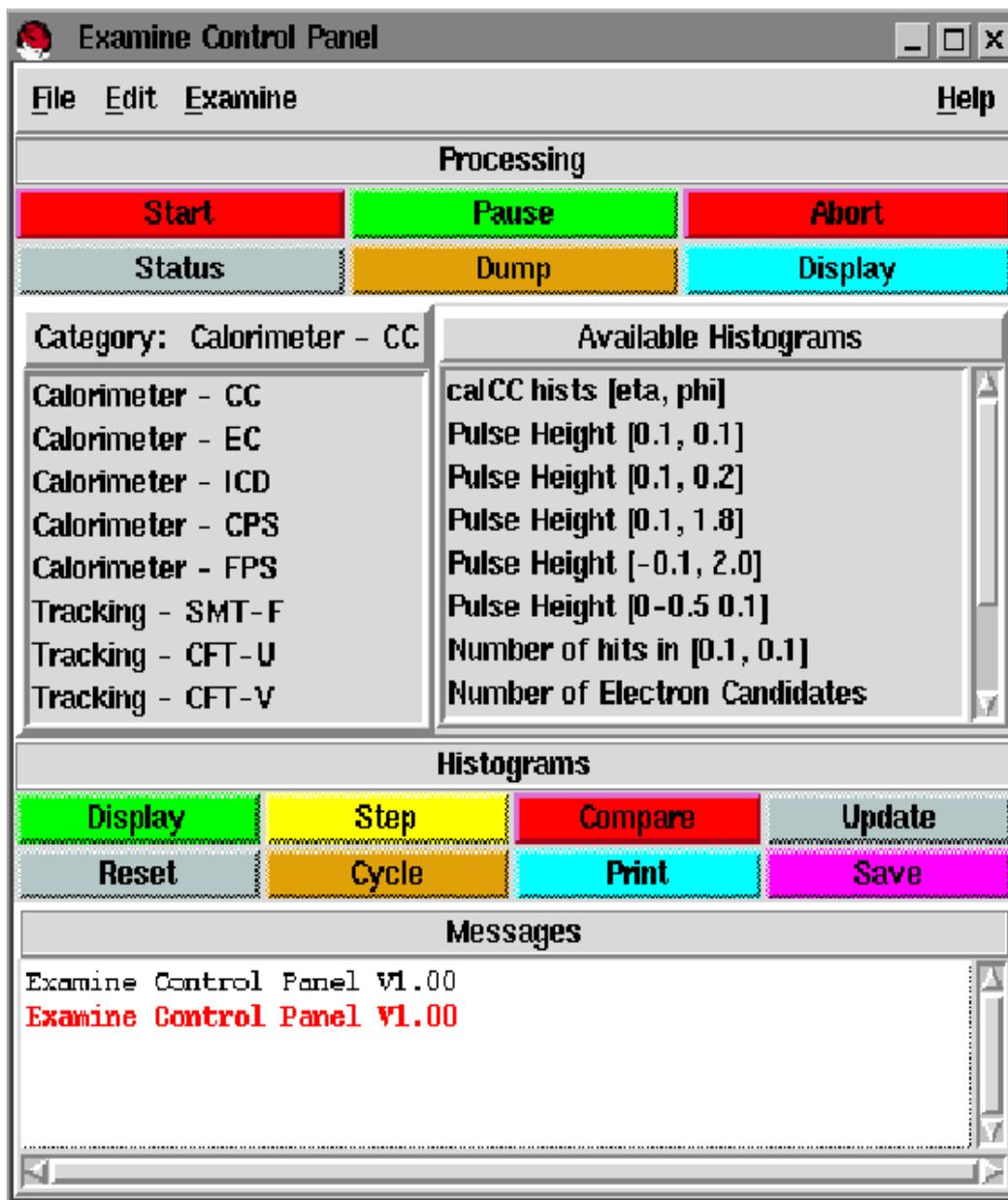


Figure 3: A prototype Examine GUI window written in Python.

- Must provide the list of available histograms via the “*Histogram Interface*”.
- Must provide sufficient online help for the operation of GUI and Examines.
- Must know how to communicate with the Examine executable via the “*Message Interface*” for process control.
- Must know to communicate to the PAT for histogram manipulations.
- Be aware of the Examine executable connection status. Be able to reconnect itself if the connect is lost.

References

- [1] S.Fuess *et. al*, Online Group, “DØ Online Data Distributor Requirements,” DØ Note #3540 (1998)
- [2] J. Kowalkowski *et al.*, “Framework User’s Guide,”
http://www-d0/run2_offline_software/framework/framework.html
- [3] H.Greenlee, J.Hobbs, S.Snyder, and V.White, “DØOM User Guide”, Section 3, 23 (1998)