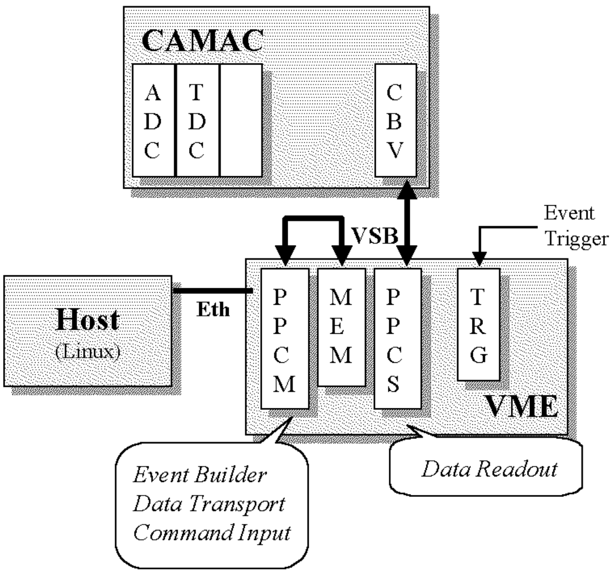
MAR$_\mathrm{a}$BOU – A MBS and ROOT Based Online Offline Utility

C. Broude [a], R. Lutter, O. Schaile, K. Schöffel, K. Steinberger, and P. G. Thirolf
[a] Weizman Institute of Science, Rehovot, Israel

Status

The new data acquisition program MAR$_\mathrm{a}$BOU [1] went into full operation at the beginning of this year. All of the existing experiments were ported to the new system without major difficulties. As compared to the old MLLE system MAR$_\mathrm{a}$BOU users get the benefit of ROOT's extended histogramming and analyzing capabilities.

Currently two stations are available both equipped with a pair of PowerPCs (200 MHz CES RIO2-8062) and a VME-to-CAMAC interface. The hardware configuration is shown in fig.1



PPCM PowerPC **Master**, 200 MHz, LynxOs
PPCS PowerPC **Slave**, 200 MHz, LynxOs
MEM Dual-ported Memory
VSB VME Subsystem Bus
CBV VME-CAMAC Interface
TRG Trigger Module

Fig. 1: MAR$_\mathrm{a}$BOU hardware configuration

Getting started

The user has to describe the event structure of his experiment by writing a few C++ statements to a so-called ROOT macro (see fig.2). These declarations are then processed by ROOT's C++ interpreter CINT [2] and generate code for the MBS front-end as well as for the analyzing program in the Linux workstation. The MBS part has to be compiled under LynxOs and to be linked to the MBS readout task. It performs CAMAC readout and event formatting according to definitions in the configuration macro.

The analyzing part consists of a set of C++ class definitions and methods providing means to process incoming event data. Compiling this code and linking it to ROOT and MAR$_\mathrm{a}$BOU libraries results in a module called M_analyze which acts as a client to the MBS front-end.

How to process MBS data

M_analyze is responsible for

- input of MBS data via TCP/IP
- conversion of MBS events to ROOT objects
- histogramming and monitoring
- event filtering
- storage of event data in ROOT format

Although most of these tasks are auto-generated by the CINT macro without any additional line of user-written code a user normally wants to add his specific event analysis. Within the ROOT framework he has access to a variety of filtering and fitting methods simplifying his task.

As M_analyze is able to re-read ROOT-formatted event data from file instead of taking data directly from MBS it may be used without modifications to analyze events in an offline session.

Histograms, filter conditions, and fitting functions are stored in a memory-mapped file and thus may be accessed online by other programs. At the end of each run this file is converted to standard ROOT format for later use in an offline session.

M_analyze is controlled by a graphical user interface called C_analyze: it allows to choose the mode of operation (input from TCP or file), to define file names and run ids, to control the data flow (start, pause, stop), and to save the memory-mapped data. Fig.3 shows the interaction between C_analyze/M_analyze as a data producer and different consumers.

HistPresent - a histogram presenter

The program HistPresent provides tools typically needed in an online monitoring environment of nuclear physics experiments. Nonetheless it turned out to be also useful for offline data analysis and presentation.

The program gives simple access to ROOT objects stored in memory-mapped areas or normal ROOT files. The objects handled are 1- 2- or 3-dimensional histograms, 1- or 2-dimensional windows, and functions. It allows operations on histograms like adding, subtracting, multiplying, dividing, scaling. Histograms can be re-binned, zoomed, overlayed and projected. All the graphics options provided by ROOT like legoplots, surfaces and contourplots are available.

Windows used as cuts in 1- or 2-dimensional histograms are important in the analysis. HistPresent allows definition of windows either graphically with the help of the pointer or by typing the values on the keyboard or by a combination of both. They can be stored on files e.g. to be used in an online monitoring program. It might be worth to mention that all ROOT objects on a file carry unique names/versions. This largely simplifies the access of objects and readable programming.

ROOT contains the minimization and error analysis program MINUIT. HistPresent provides a simple interface to the functions of MINUIT. Fitting of Gaussians (optionally with tails) and polynomials is possible just with the help of the mouse. More complicated formula functions may be provided by the user in form of a ROOT macro (essentially C++ code). A built-in template macro makes this an easy task even for a non-experienced user. Functions may be stored to and retrieved from a ROOT file like windows. This is typically used to apply corrections to raw data in an analysis program.

A rudimentary peak finder useful for gamma ray experiments is provided.

All graphical objects can be stored as PostScript or gif files or directly sent to a printer.

Many parameters used by the program like canvas positions and sizes, default file names, printer names etc. may be customized in a resource file (.rootrc). Some parameters like expansion factors, lin-log scale are remembered in and between sessions for each histogram individually.

HistPresent has been developed along with its use in experiments. It therefore mainly reflects the needs of the current users. Thanks to the modularity of ROOT and its built-in C++ interpreter CINT [2], implementation of a new functionality is a simple task.

```
{
gSystem->Load("/usr/local/marabou_$CERN_LEVEL/lib/libTMrbUtils.so");    // libraries containing class defs
gSystem->Load("/usr/local/marabou_$CERN_LEVEL/lib/libTMrbConfig.so");   // and methods
exa = new TMrbConfig("exa", "Example configuration");     // create a C++ object
                                                          // to store config data
exa->Trigger(1, "readout", "readout of camac data");      // we use trigger 1 for readout
data = new TMrbSubevent_10_11("data", "camac data");      // define a subevent type [10,11]:
                                                          // data stored in a zero-padded list
adc1 = new TMrbSilena_4418V("adc1", "C1.N3");             // define camac modules
adc2 = new TMrbSilena_4418V("adc2", "C1.N5");             // (module type & position)
data->Use("adc1", "etot de1 de2 de3 de4 veto");           // assign event parameters
data->Use("adc2", "pos1r pos1l pos2r pos2l");             // to camac modules
data->OnTrigger("readout");                               // connect subevent to trigger
exa->WriteTimeStamp();                                    // include a time stamp to raw data
exa->DefineVariables("I", 1000, "v1[3] v2[4] v3[5]");     // define variables and windows
exa->DefineVariables("F", 1.2345, "v4[2] v5 v6");         // (to be modified interactively)
exa->DefineWindows("I", 0, 1023, "w1 w3 w2");
sca1 = new TMrbUct_8904("sca1", "C1.N14");                // some scalers
sca2 = new TMrbUct_8904("sca2", "C1.N15");
exa->MakeReadoutCode("", "Overwrite");       // generate C readout code for MBS
exa->MakeAnalyzeCode();                      // generate C++ class defs & methods for data analysis within ROOT
exa->WriteToFile("", "Overwrite:Verbose");   // save configuration to root file
gSystem->Exit(0);
}
```

Fig. 2: Example of a configuration macro

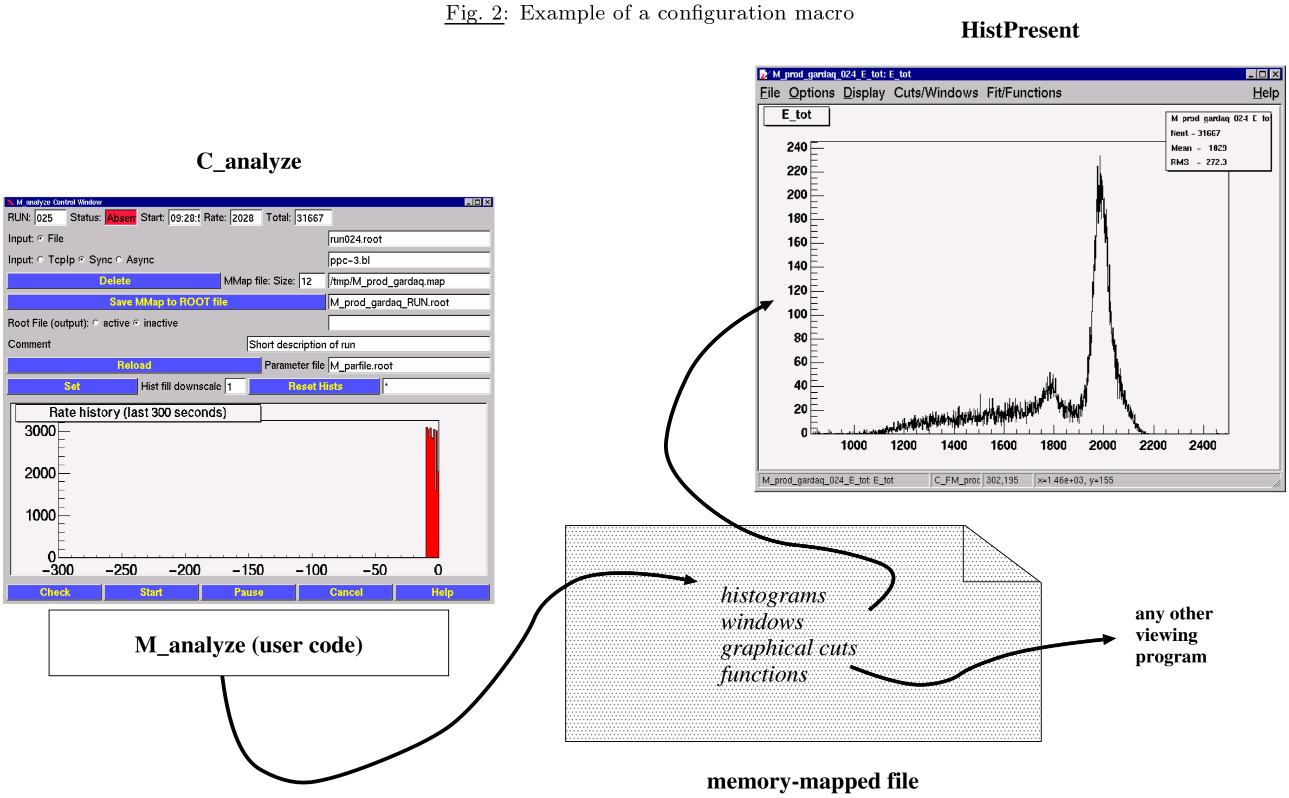


Fig. 3: Data exchange via memory-mapped file

References

[1] see also:
H. Essel, N. Kurz *et al.*: MBS – The GSI Multi Branch System, http://www-gsi-vms.gsi.de/daq/home.html
R. Brun, F. Rademakers *et al.*: The ROOT Object Oriented Framework, http://root.cern.ch

[2] M. Goto, The CINT C/C++ Interpreter, http://root.cern.ch/root/Cint.html