Integrating Controls and Event Data

Ron Fox (fox@nscl.msu.edu)

August 13, 2003

Abstract

This document describes the software that allows you to integrate controls and event data into a single event file. The mechanisms and theory of the software are described. A step by step recipe is given for taking existing readout software and fitting it into the necessary frameworks. Run time procedures are given that describe how to ensure that the data is available to the readout program, and sample scripts are shown that describe how to ensure that the controls data are logged to event files.

An online version of this manual can be found at the NSCL documentation website: http://docs.nscl.msu.edu

Refer to the table of contents below

Contents

to locate specific sections of the document.

1 How it Works

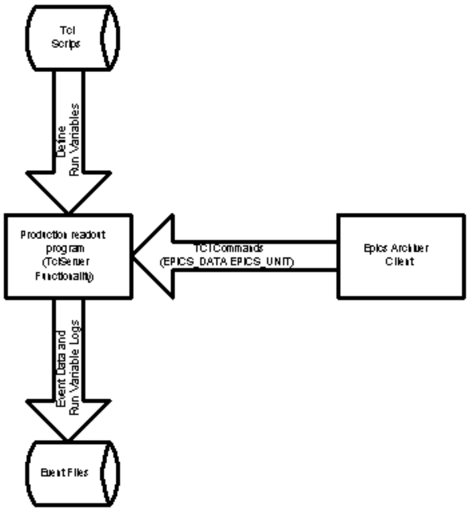
This section describes:

- The software components that cooperate to integrate controls data with the experiemnal event file.

- An overview of the step by step procedure to include controls information in experiment event files.

The figure below shows schematically how the system works:

Figure 1: Component Block Diagram



The key components are:

Production Readout A new production quality readout program. this program has many additional features however the two which are important to this document are:

- The program can function as a TCL server, allowing it to accept TCL commands from external programs (similar to the scaler display program).
- TCL Scripts can designate TCL variables in as *run variables*. Run Variables are periodically logged to the event file.

Epics Archiver A tcl server client that takes a list of control system channel names and maintains in its server (Production Readout) a pair of arrays:

- EPICS_DATA(*channel_name*) contains the value of a channel named *channel_name*.

- EPICS_UNIT(*channel_name*) contains the units of a channel named *channel_name*.

To use these components to integrate controls and event data you must:

- [ ] Port any existing readout software to the production readout software framework.
- [ ] Configure the EPICS archiver so that it will connect to the Readout program and maintain the desired set of channel information in the EPICS_DATA and EPICS_UNIT arrays.
- [ ] Write a script that allows the epics archiver to connect to the readout program.
- [ ] Write a script that designates elements of the EPICS_DATA and EPICS_UNIT arrays as run variables.
- [ ] Write a script to start up the readout program with the desired command line switches.

2 Porting Existing Readout Software to Production Readout

This section describes how to port existing readout software to the production readout system. For information about the production readout software, see the user and programming documentation in http://docs.nscl.msu.edu

For illustrative purposes, we will assume that you start working in a directory whose contents are a subdirector named *oldaq*, and that oldaq contains an existing working readout program. Porting this working readout program is a matter of:

- ☐ Obtaining the production readout skeleton in the 'empty' directory.
- ☐ Copying the appropriate files from your existing, working readout program to the 'empty' directory.
- ☐ Making slight modifications to the production readout skeleton and the existing, working software to adapt it to the production readout framework as a "traditional event segment".
- ☐ Build and test your new readout software.

2.1 Obtaining the production readout skeleton

The production readout skeleton software is located in /usr/opt/daq/pReadoutSkeleton on the data acquisition system boxes. To get a copy of this skeleton cd to the empty directory and issue the command:

```
cp /usr/opt/daq/pReadoutSkeleton/* .
```

This will copy the following files:

Table 1: Production Readout Skeleton Files

File	**Contents**
Skeleton.cpp	Program skeketon for user readout code
Makefile	Starting point for Readout's makefile
CTraditionalReadoutSegment.cpp	Bridge to old style readout program code
CTraditionalcalerReadout.cpp	Bridge to old style scaler readout code.

2.2 Copying the appropriate files from the working readout program

An old style working readout program will typically have:

- A Makefile

- A ReadoutMain.cpp application object entry point file.
- A file derived from skeleton.cpp that contains the event and scaler readout software.
- Additional source and header files that define additional classes and functions to assist in the readout process.

In the example we are describing we will assume that the original skeleton.cpp has been modified and is now called gasN4_daq.cpp. We will also assume there is a configuration library named param.cpp with an associated header named param.h

First, what *not* to copy:

- Do *not* copy your old Makefile.
- Do *not* copy your ReadouMain.cpp file.

Copy all files that are experiment dependent. If some files have names that conflict with files in the production readout skeleton, be sure to rename these files as you copy them. In our example case, we need to copy:

- gasN4_daq.cpp
- param.h
- param.cpp

This is done as follows:

```
cp oldaq/gasN4\_daq.cpp .
cp oldaq/param.h           .
cp oldaq/param.cpp         .
```

2.3 Modifying your code to port to the new framework

To modify your code to work with the new framework you must:

- [ ] Edit the Makefile to include your software in the build.
- [ ] Edit Skeleton.cpp to register a "traditional" event segment for readout.
- [ ] Resolve incompatibilities between the old and production frameworks.
- [ ] Build the new readout program.

2.3.1 Editing the Makefile

Since we are using the "traditional" readout scheme we must first ensure that the compatibility mode bridges will be compiled and linked. Locate the section of the makefile that reads:

```
#
#  Add any additional objects you need to build to this line.
#
Objects=Skeleton.o
#
#   If you are using a 'traditional readout skeleton uncomment the next line:
#Objects= Skeleton.o CTraditionalEventSegment.o  CTraditionalScalerReadout.o
```

Modify this section to read:

```
#
#  Add any additional objects you need to build to this line.
#
#Objects=Skeleton.o
#
#   If you are using a 'traditional readout skeleton uncomment the next line:
Objects= Skeleton.o CTraditionalEventSegment.o CTraditionalScalerReadout.o
```

Add your object modules as well to this line. In our example, this will make the Objects macro definition read:

```
#
#  Add any additional objects you need to build to this line.
#
#Objects=Skeleton.o
#
#   If you are using a 'traditional readout skeleton uncomment the next line:
Objects= Skeleton.o CTraditionalEventSegment.o CTraditionalScalerReadout.o   \
              param.o gasN4_daq.o
```

2.3.2 Modifying your program

You will be using the production readout system to read a "traditional" event segment and scaler bank. Traditional event segments and scaler banks allow you to use existing code for the old readout system.

You must register a traditional event segment and scaler readout by editing Skeleton.cpp:

Locate the function MyExperiment::SetupReadout. It should look like this:

```
void
CMyExperiment::SetupReadout(CExperiment& rExperiment)
```

```
{
   CReadoutMain::SetupReadout(rExperiment);

   // Insert your code below this comment.

   rExperiment.AddEventSegment(new MySegment);

}
```

Modify the last executable line of this function to read:

```
...
   rExperiment.AddEventSegment(new CTraditionalEventSegment);
```

Locate the function CMyExperiment::SetupScalers and modify it to read:

```
void
CMyExperiment::SetupScalers(CExperiment& rExperiment)
{
   CReadoutMain::SetupScalers(rExperiment);

   // Insert your code below this comment.

   rExperiment.AddScalerModule(new CTraditionalScalerReadout);

}
```

In the production readout framework, scalers are counted difrerently. Scalers come in banks, and each scaler bank is expected to supply a count of the number of scalers it reads. To accomodate this, the daq_SetScalerCount function has been removed, and the old skeleton code needs to define and implement a new function named numscl() that returns this information.

In our sample, this means that gasN4_daq.cpp must be edited as follows:

Remove the call to daq_SetScalerCount from iniscl():

```
 Old code:
    ...
   daq\_SetScalerCount(32);
   INIT4434(SCALER\_B,SCALER\_C,SCALER\_N);
   ...
New code:
   ...
   INIT4434(SCALER\_B,SCALER\_C,SCALER\_N);
   ...
```

Add the definition and implementation of numscl:

```
/**
   numscl - return the number of scalers to be read:
**/
unsigned int numscl()
{
  return 32;
}
```

Since the production readout system does not manage the CAMAC system the same way, be sure to do a branchinit() and crateinit() on every branch and crate you reference.

2.4 Build and test your readout program

To build the new readout program type:
`make`

Test the program by running it directly. Note that the production readout program uses a Tcl/Tk based command interpreter that supports most of the same commands as the "traditional" readout system (show all is not supported). Note that the production readout framework recognizes the following command switches:

Table 2: Production Command line switches

Short Form	**Long Form**	**Meaning**
-h	–help	Outputs a brief help message
-V	–version	Outputs the framework version number
-p¡portnum¿	–port=¡portnum¿	Enables tcl server functionality on port ¡portnum¿
-w	–window	Runs Tk interpreter (enabling readout to have GUI).
-c	–camac-trigger	Use CAMAC trigger (default is via CAEN V262).

Note that when you run Readout from the Readout GUI front end, you will need to write a startup script in order to use any of these switches.

Table 3: Variables maintained by EpicsArchiver

Variable	Contents
EPICS_DATA(*channel*)	Current value of *channel*
EPICS_UNIT(*channel*)	Units (e.g. Amps) of *channel*

3 The Epics Archiver Program

EpicsArchiver is a component that monitors a set of EPICS channels, communicates with a TclServer and maintains a set of TCL variables in that server that reflect the current value of these parameters. This section describes how to run the EpicsArchiver. Note that EpicsArchiver is not limited to working with the production readout program to log epics channels to the event data stream. It can also be used with an ordinary TclServer to produce monitoring and alarm displays of selected control system parameters.

EpicsArchiver maintains two arrays for its tclserver enabled process. Each array is indexed by channel name (Tcl arrays have textual indices).

EpicsArchiver runs on Windows2000 at present. To run it you will need to:

- ☐ Make a directory for its configuration files.
- ☐ Make a .ini file that specifies the overall program options.
- ☐ Make a channel list file that describes the set of channels to monitor.
- ☐ Create a dekstop shortcut to make running the program easy.

3.1 Creating a directory for Epicsarchiver configuration files

- ☐ Double click on the “My Computer” icon on the desktop.
- ☐ Double click on the Icon labelled “Local Disk (C:)” in the window that pops up.
- ☐ Click File-¿New-¿Folder A new folder will be created, change its name to the name of your experiment (e.g. gas001).

3.2 Creating the epicsarchiver.ini file

Use the notepad editor to create a file in this new folder named “epicsarchiver.ini”. A sample ini file is shown below:

```
FileIn=C:\gas001\channels.txt
FileLog=C:\gas001\log.txt
SleepTime=10
UpdateTime=10
```

Table 4: epicsarchiver.ini file parameters

Keyword	Meaning of value
FileIn	Path file that has list of channels
FileLog	Path to file that will have program log
SleepTime	Number of seconds between readouts.
UpdateTime	Number of seconds between reconnects to disconnected channels
PortNumber	TCP/IP port of tclserver connected to (2701 is the convention).
HostName	Name or IP of host running tclserver we connect to.

```
PortNumber=2701
HostName=u4pc2.nscl.msu.edu
```

The lines parameters that can be set in this file are:

3.3 Creating a channels file

The channels file contains the names of the channels to monitor, one per line. For our running example, this file will be named: c:
gas001
channels.txt and might contain:

```
P222ER
P222ET
```

To monitor the channels P222ER and P222ET.

3.4 Creating a shortcut for epics archiver

We suggest you make a shortcut on the desktop to run epics archiver. Since epics archiver exits when it is no longer able to talk with its tcl server, you should make the short cut a batch file that loops, continuously invoking epicsarchiver. Create a file with the following contents in your epicsarchiver directory (in this example C:
gas001

```
:Loop
   c:\"program files"\tclclients\epicsarchiver.exe
goto Loop
```

Save this file and create a shortcut to it on your desktop. You can now run your customized epics archiver by double–clicking this shortcut. The batch file will continue to run the program until it successfully connects to its tcl server. The program will run until it is no longer able to write to the Tclserver (e.g.

the readout program exits), or until the Escape key is pressed while input focus is in the epicsarchiver window. Note that once the program exits, the batch file will re-run it again.

To quit you must click on the “X” window decoration on the epicsarchiver window.

4 Putting it all Together

This section describes how to integrate the new features of the production readout software with the EPICS archiver so that you can log control data to event files. Briefly you will:

- Startup the Readout program.
- Source a script into the readout program to allow the epics archiver to connect as a TclServer client.
- Start the epics archiver.
- Source a script into the readout program that will make all of the epics variables runvar's which in turn will ensure they are periodically logged to the event file.

4.1 Starting the new Readout Program

In order to start the production readout program as a TclServer you will need to create a startup script. The startup script is then run by the ReadoutGui just as the old readout software was. To start a Readout program that will be a TclServer listening for connections on port 2701 (the convention for controls data) and triggering off the CES CBD8210 INT2 input (camac trigger), create a file (e.g. runreadout) that contains the text:

```
#!/bin/bash
cd ReadoutDir
Readout --camac-trigger --port=2701
```

Where *ReadoutDir* is the directory your readout program is stored in, and *Readout* is assumed to be the name of the executable Reaout program.

Once this script has been created, make it executable:

```
chmod a+x runreadout
```

Specify to the ReadoutGui that the program you want to run is runreadout.

4.2 Authorizing remote tcl client connections

Since the ability to execute tcl commands over a remote connection is a security hazard (consider what you can do with the tcl **exec** command), tcl servers initially only allow connections from the localhost. To accept connections from other hosts, you must execute the **serverauth** command within the Readout program. To do this, source in a script (e.g. authorize.tcl) that contains a line like:

```
serverauth add u4pc1.nscl.msu.edu
```

where you must substitute the host name in the example above with the hostname of the system on which you will run the epics archiver (usually u$n$pc1 where n is the data u in which you run.

To source this script in, on the ReadoutGUI click File-¿Source choose the file you want to source using the file chooser dialog window that pops up.

4.3 Starting the epics archiver

To start the epics archiver, double click the short cut you created on the windows PC's desktop.

4.4 Making Epics variables into runvar's

The epicsarchiver program will create two arrays, EPICS_DATA, containing channel values, and EPICS_UNIT containing the units of the channels. In order to log this data to the event file it is necessary to specify the elements of these arrays as run variables. To do this, create and source a script like this:

```
foreach channel [array names EPICS_DATA] {
   runvar EPICS_DATA($channel)
   runvar EPICS_UNIT($channel)
}
```

This script iterates through the set of indices of EPICS_DATA and tells the readout program to make the corresponding elements of EPICS_DATA and EPICS_UNIT into run variables. Run variables are logged to the event stream each scaler readout interval. The format of these documentation buffers is described in the production readout program user guide at http://docs.nscl.msu.edu

5 Cool advanced stuff

You can use the fact that the Readout program can run as a wish interpreter to visually monitor the control variables you are logging (you can do this in a separate TclServer too of course). This section presents a set of shell and Tcl scripts that show how to do this.

To start the readout program as wish based tcl server you must set the display environment variable in the runreadout script and specify the window switch to readout. A sample script that does this is shown below:

```
#!/bin/bash
cd ~/controlfix/testing
export DISPLAY=compgrp5:0.0
Readout --camac-trigger --port=2701  --window
```

The script below determines if the tk interpreter is active and loaded. If so, it presents a very simple display of all the epics data updated by the epicsarchiver.

```
foreach channel [array names EPICS_DATA] {
    runvar EPICS_DATA($channel)
    runvar EPICS_UNIT($channel)
}


if {[info var tk*] != "" } {
    if {$tkloaded} {
foreach channel [array names EPICS_DATA] {
    destroy .f$channel
    frame .f$channel
    label .f$channel.label -text $channel
    label .f$channel.value -textvariable EPICS_DATA($channel)
    label .f$channel.units -textvariable EPICS_UNIT($channel)
    pack .f$channel.label .f$channel.value .f$channel.units -side left
    pack .f$channel -side top
}
    }
}
```

Note how this script checks for the existence of tk variables and the state of tkloaded before creating the gui. This script can be run either in window or nonwindowed modes of readout.