Embedding control parameters in data buffers

Ron Fox (fox@nscl.msu.edu)

November 29, 2005

Abstract

This document describes how to embed EPICS control system parameters into data buffers produced by the NSCL production readout software. The document structure is as follows:

- A step by step procedure is shown with a sample set of channels and minimal explanation. This is intended for people who just want to get up quickly without understanding what they are doing.
- The structure and components of the system are described.
- The controlpush program is described.
- The production readout features used to set up the link are described.

The online and printable version of this document are avaialable from the NSCL web site: http:/docs.nscl.msu.edu

Refer to the table of contents below to locate specific sections of the document.

Contents

1 Quick Start Guide

This section provides a quick start to using the system. We assume that you:

- Have a Readout program from the production readout skeleton built (e.g. /usr/opt/daq/pReadoutSkeleton skeleton).
- This skeleton is in the *readout* directory below your home directory (`~/readout`).

We provide:

- A sample channel definition file
- Scripts to startup the readout program
- Scripts to source into the readout program that start controlpush and establish the set of variables that will be logged to disk.
- A step by step procedure for getting up and running.

All sample files are available for download from the nscl documentation server at http://docs.nscl.msu.edu/daq/LinuxPush/samples

These files are intended as a starting point from which you can tailor your own configuration.

1.1 A sample channel Definition file

Channel definition files describe to the system the set of channels you will be monitoring. The channels described in this file will be periodically read out from the EPICS control system and logged as run-variable buffers by the production readout software.

Channel definition files are lists of channels. Below is a sample channel definition file that shows many of the features of channel definition files:

```
#  This line is a comment
     # as is this one.

# fully whitespace or empty  lines like the one above are ignored.
D171QA
D173QB
D179DV
D181DH          This is a comment too, no # needed.
D183QA
   D185QB       Spaces, tabs  can lead the channel name.
D197DS
D197DS-F
E201QA
E203QB
E206DV
E208QA
```

```
E210QB
E214DS
E214DS-F
P218QA
P220QB
P222ER
P222ET
P224QA
P226QB
STRCHAN100
STRCHAN101
STRCHAN100
STRCHAN300
STRCHAN40
```

This example can be found
at http:/docs.nscl.msu.edu/daq/LinuxPush/samples/channel.txt

1.2 Scripts to start readout

Rather than passing the name of the Readout program directly to the ReadoutGUI, we will pass a script. The RunReadout script will start controlpush, which transfers control data from the control system to the readout program, and startup the readout program as well.

```
#!/bin/bash
TRIGGER=""          # If you have a CAMAC trigger -> "--camac-trigger"
PORT=2701           # This is the default port number.
INTERVAL=10         # Seconds between updates.
export PATH=$PATH:/usr/opt/daq/bin

cd ~/readout

killall -9 controlpush # Kill off old pushers.
killall -9 Readout # kill off old readout programs.

controlpush --port=${PORT} --interval=${INTERVAL} channel.txt & # start pushing channels...
Readout --port=${PORT} $TRIGGER # start the readout program.

killall -9 controlpush # kill off my pusher...
killall -9 Readout              # Kill off old readout programs.
```

This script is
at http://docs.nscl.msu.edu/daq/LinuxPush/samples/RunReadout

At the beginning of the file are some definitions: TRIGGER, PORT, and INTERVAL. If your trigger comes from a CES CBD8210 VME branch highway driver, you should modify this definitions so that the text `--camac-trigger` is between the quotes. The PORT defintion defines the TCP/IP port on which the Readout software will listen for TCL client connections. Unless there is some compelling reason, don't change this. The INTERVAL variable is set to the number of seconds between control system updates.

Note that the script kills off the control program on a normal exit, but ensures against lingering controlpush programs by trying to kill them off on startup.

1.3 Scripts to source into Readout

The production readout program will also need to source a script in order to define the set of TCL Variables to record to disk.

The script is shown below:

```
set ChannelFile channel.txt



#
#   Parse the channel file and create runvars as needed.
#
proc SetupChannels {} {
    global ChannelFile
    set f [open $ChannelFile r]
    set file [read -nonewline $f];       # Suck in the whole file.
    close $f
    set lines [split $file "\n"];       # Make a list of lines
    foreach line $lines {
        set channel [lindex $line 0];     # Channel or comment...
if {($channel != "") && ([string index $channel 0] != "\#")} {
    puts "Creating run variables for $channel"
    runvar EPICS_DATA($channel)
    runvar EPICS_UNITS($channel)
    runvar EPICS_UPDATED($channel)
}
    }
}

 SetupChannels;               # Process channel definition file
```

This file is available for download as:
ReadoutStartup.tcl from http://docs.nscl.msu.edu/daq/LinuxPush/samples/ReadoutStartup.tcl

1.4 Putting it all together

To review. To get a set of EPICS channels periodically logged in buffers generated by the production readout system (and therefore stored on disk):

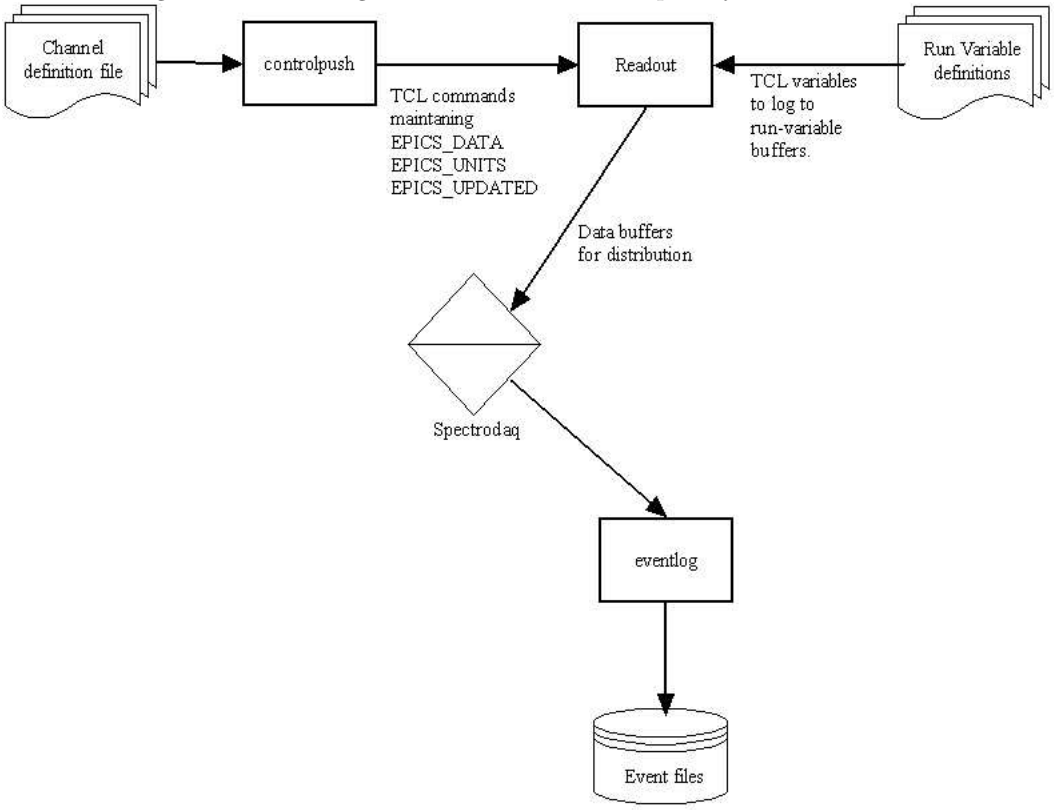
1. Download the channel file from
   http://docs.nscl.msu.edu/LinuxPush/samples/channel.txt. Store channel.txt in `~/readout` along with your production readout software. Modify it to reflect the set of channels you want to monitor.

2. Download the Readout startup script from
   http://docs.nscl.msu.edu/LinuxPush/samples/RunReadout store it in `~/bin` and "chmod a+rx `~/bin/RunReadout`". Edit this script if necessary to modify the options passed to Readout and chanlog the channel pushing program. Arrange for the readout gui to run this script as the "Readout program".

3. Download the Readout initialization script from
   http://docs.nscl.msu.edu/daq/LinuxPush/samples/ReadoutStartup.tcl save this script in `~/readout`.

4. Download the ReadoutCallouts.tcl script from
   http://docs.nscl.msu.edu/daq/LinuxPush/samples/ReadoutCallouts.tcl. Modify this file as directed by the comments in the file itself. If you already have a ReadoutCallouts.tcl script for your experiment (this will either be in your home directory or the directory from which you run the ReadoutGui), integrate this script with it, otherwise save this script in one of those two places. The script will run whenever Readout is started and will force the ReadoutCallouts.tcl file to be executed by the productino readout program.

Figure 1: Block diagram of the control to daq subsystem



2 How it all works

Figure 1 is a block diagram of the system. A toolkit approach was used to assemble the system out of simple well defined components.

- The controlpush program reads a channel configuration file. This file contains a list of channels that will be recorded to disk along with the physics event data.
- The controlpush program periodically updates the values, units and last update times of the channels in the configuration file.
- The controlpush program maintains a connection with a TclServer compatible program. A TclServer is a program that accepts TCP/IP connections and process Tcl/Tk commands that are sent across these connections. In this case, The TclServer is part of the production readout software.
- The controlpush program pushes a bunch of Tcl *set* commands across its

TclServer connection and maintains a set of variables in that server:

EPICS_DATA is an array indexed by channel name. It contains the last known value of the control parameter.

EPICS_UNITS is an array indexed by the channel name. For channels that have units, the values of this array will contain the units string (e.g. mm for milimeters).

EPICS_UPDATED is an array indexed by the channel name. It contains a string that encodes the date and time at which the channel could last be successfully updated. The form of this string is YYYY-MM-DD HH:MM:SS. For example; a channel last updated March 5, 2004 at 5:36pm and 12 seconds will be encoded: 2004-03-05 17:36:12.

- The production readout program uses the ReadoutStartup.tcl script to describe the set of channels you want recorded to disk. This script gets sourced into the production Readout program by the ReadoutCallouts.tcl script which is calls That file's OnStart proc when it starts the readout program. These variables are called run variables. Readout periodically creates buffers that contain tcl script fragments that can be executed to re-create the values of all of the run variables.

- The production readout program submits these run variable buffers along with all of the other data buffers produced during a run to the SpectroDaq data distribution server.

- If data recording is enabled, the eventlog program is actively accepting buffers from SpectroDaq and recording them to the run's event file. since eventlog is not selective about the buffers it records, the run variable buffers willl be recorded along with all the other run data.

3 The controlpush program

The controlpush program is responsible for all communication with EPICS. It is also responsible for connecting to a TclServer (while for the sake of this document the TclServer is the Readout program, this is not necessary, controlpush can be used to e.g. build TclServer based monitoring displays).

This section describes:

- How to run the controlpush program, and the command line arguments it accepts.
- The detailed format of the controlpush channel configuration file.
- The specification to which the controlpush program was written.

3.1 controlpush command line options

Below is the help text from the controlpush command:

```
Purpose:
  Transfer control system data to Readout/tcl servers

Usage: controlpush [OPTIONS]... [FILES]...
   -h         --help          Print help and exit
   -V         --version       Print version and exit
   -pINT      --port=INT      Port to use when connecting to tcl server.
   -iINT      --interval=INT  Control parameter update interval
   -nSTRING   --node=STRING   Host on which tcl server is running
```

Each command option has a short and a long form. The command options are:

-h, –help Print the short help that is shown above and then exit without doing anything else.

-V, –version Print the program version and exit without doing anything else.

-pPort –port=Port *Port* is the TCP/IP port to which controlpush will try to connect. if omitted, this defaults to 2701. The program will retry periodically if not able to connect but will, eventually, give up.

-nNode –node=Node The TCP/IP host to which controlpush will attempt to connect. If omitted, this defaults to localhost. The host parameter can either be the dns name of the host or a dotted IP (e.g. 35.8.32.111). Note that in general, if the node is other than localhost, the TclServer must have authorized connections from that host. This is done via the "*serverauth add* **node**" command.

-iInterval –interval=Interval The interval between EPICS update attempts in seconds. If omitted, this defaults to 10 seconds.

The program accepts a single file argument. This argument is the name of the channel definition file or the string "-" to read channel definitions from standard input.

The channelpush program is stored in the bin directory of the daq distribution (/usr/opt/daq/bin at the NSCL).

3.2 controlpush channel configuration file format

The controlpush channel configuration file tells controlpush which channels to monitor. This file can contain comments, blank lines and other spacing information as desired to make it legible:

- Fully blank lines are ignored (this includes empty lines as well).
- Lines whose first non-whitespace character is a pound symbol "#" are treated as comments and ignored.
- The first "word" of all other lines is treated as a channel name. A word is defined to be a contiguous series of non-whitespace characters. All characters following the first such word are ignored, and can therefore contain commenting information.

3.3 controlpush full specifications

```
Linux based control push program.


1  Accepts a file, update-interval, host and port on the command line.
  1.1 host defaults to localhost.
  1.2 port defaults to 2701
  1.3 File is required but - is understood to mean stdin
        to support piping and redirection.
  1.4 See File format below for the format of the input file.
  1.5 Update interval defaults to 10 seconds.
2 Each channel described in the file is looked up on EPICS.
  2.1  Lookup failures result in an error message, and
         the program continues ignoring the channel.
  2.2  For each channel that is succesfully found,
       it's units channel is also searched.
     2.2.1 Units channel connection timeouts are
           silently ignored (Assumption: not all
           channels have units).
3 Program attempts to form a TCP/IP to the requested host/port
  3.1 Requested host is assumed to be a tcl server.
  3.2 If the connection could not be made, the software
      will retry every 10 seconds at most 1000 times
      until a connection is formed.
  3.3 If the connection is lost, the software will
```

```
      retry every 10 seconds at most 1000 times
      until a connection is formed.
  3.4 During connection retries as described in 3.2,3.3
      no EPICS operations will be performed.
  3.5 Upon (re)connection the state of the channels
      are considered to be unknown:
      3.5.1 All defined channels are looked up again.
      3.5.2 All channel values are set to "-not-updated-".
            (see 4.1).
      3.5.3 -dead- channels as defined by (4.2) are added
            back to the list of channels that are queried.
4 When connected, the program will query the value and units
  of each channel every update-interval
  4.1 Prior to the first successful query, all units channel
      values are initialized to "-not-updated-".
  4.2 If a channel fails its update 2 consecutive times,
      its value is set to -dead- and it is permanently
      removed from the list of channels queried.
  4.3 Channel data for a single update are 'as synchronized
      as EPICS permits"
  4.4 Each channel will have associated with it a most recent
      update timestamp.
     4.4.1  This timestamp is only updated when a value is
            successfully received from the EPICS system.
     4.4.2  The timestamp will be the absolute time as
            known to the computer that this program is
            running in.
  4.5 Each channel value and its associated units are transmitted
      across the connection.
     4.5.1 Channel data (value, units and timestamp) are transmitted
           in the form of legal TCL commands
          4.5.1.1 Channel values are transmitted as legal variable setting
                 commands of the form:
                  set EPICS_DATA(channel-name)  value
          4.5.1.2 Channel unit values are transmitted as legal variable
                  setting commands of the form:
                    set EPICS_UNITS(channel-name) value
          4.5.1.3  Channel timestamps are transmitted as legal variable
                  setting commands of the form.
                    set EPICS_UPDATED(channel-name) time
                 4.5.1.3.1  The format of time is:
                             yyyy-mm-dd hh:mm:ss
                     This format allows times to be compared by comparring
                     their string representations (> < too) lexcially.
                    4.5.1.3.1.1 yyyy is a four digit year.
y                    4.5.1.3.1.2 mm   is a 2 digit month with leading zero if
```

```
                                needed.
                        4.5.1.3.1.3 dd   is a 2 digit day of the month with leading
                                         zero if needed.
                        4.5.1.3.1.4 hh   Is a 2 digit hour (00-23).
                        4.5.1.3.1.5 mm   Is a 2 digit minute (00-59).
                        4.5.1.3.1.6 ss   Is a 2 digit second (00-59).
      4.5.2 When a channel has not updated its last known value, units and
            last timestamp will still be transmitted.
            4.5.2.1  The initial timestamp value for will be "0000-00-00 00:00:00"
                     This indicates that a channel was located but a value could
                     never be retrieved for it.
      4.5.3 When a channel is marked -dead- (see 4.2) The value -dead- is
            transmitted.
             4.5.3.1 -dead- is only transmitted in the ccyle the channel gets marked
                     dead.  Once dead no data is transmitted again for the channel.
5 File format:
  5.1 One channel name per line.
  5.2 Blank lines are ignored.
  5.3 Only the first 'word' on a line is important.
    5.3.1 Word is defined as the first contiguous sequence of characters
          without any whitespace characters.
    5.3.2 As implied by 5.3.1 above, leading whitespace is not siginficant.
6 Command invocation:
    controlpush [--port=port] [--node=host] [--interval=interval] {file | - }
```

4 Readout features used

Two features of the production readout system are used to make all of this stuff work:

TclServer capability The production readout software can enable an internal TclServer. This allows it to be a target of the controlpush program. To enable TclServer functionality in the production readout software, include the –port=nPort switch on the invocation command line. *nPort* is the number of the TCP/IP port on which the internal tclserver will listen. Note that by default, the only clients in localhost can connect, however the internal TclServer supports the full *serverauth* command which allows you to enable access from other hosts. For example, the command "*serverauth add u3pc2*" allows the TclServer to accept connections from u3pc2.

runvariables The production readout software supports marking Tcl variables as *runvariables*. A run variable, is a quantity of interest that may vary through the course of a run. Runvariables are written as output buffers at each scaler readout interval. Control system variables are marked as run variables. This is how they get written out, once controlpush sets their values through the internal tcl server.