

Specifications for the bias voltage control software API for the tagger microscope

Hovanes Egiyan, Richard Jones

August 13, 2012

Abstract

We specify the interface required to integrate the voltage control of the GlueX tagger microscope detector into the experimental control system of Hall D.

1 Introduction

The tagger microscope detector consists of five columns of 100 rows of scintillators that will be readout using silicon photodetectors (SiPMs). The bias voltage of each of the SiPMs needs to be individually adjusted with a precision of 0.1 Volts. The control points for the voltage supply to the SiPMs are divided into groups of 30 channels with a total of 167 groups. Each group is controlled using a custom designed board that can set and report values of the different parameters for each SiPM channel. The remote communication with the board can be done through the Ethernet port using custom protocol using Ethernet Transceiver. Each control board is identified by an address set using a jumper ???, and each channel is identified using ??? (needs UConn input).

The programming of the firmware on the control boards has been done by the University of Connecticut group, and in order to be able to integrate the bias voltage control and monitoring into the EPICS-based controls framework of Hall D an interface layer needs to be developed to be used by the EPICS IOCs running on a regular Linux based computer. Within the EPICS framework different parameters of the voltage channel will have their corresponding EPICS variables. The variables that write to the control

boards will be processed (and thus communicate with the board) whenever the desired value is changed by the user, while the variables that read their values from the boards will need to be updated at about 1 Hz frequency, or whenever the readings are changed. This document specifies the requirements for such a library assuming that the time for sending a single requests to a control board and receiving a response from it over a local network is under one millisecond during the operations with the full configuration.

2 List of parameters

Table 1 lists the parameters that will be controlled and monitored by EPICS for every voltage channel. A brief description for each parameter is given in the second column. The column “Access by EPICS” indicates whether the parameter is expected to be read, written, or both by EPICS. The **status** parameter should indicate the state the channel is in, such as *On*, *Off*, *Ramp-up*, *Ramp-down*, *Tripped*, *Over-temperature*, etc. The **enable** parameter can be written by EPICS to turn on and off the channel, and it also can be read by EPICS to determine what the most recent EPICS request for enabling or disabling that channel was. Note that it provides us with different information from what is reported in **status** parameter which reports the actual state of the voltage channel.

3 List of functions

In order to access and modify the parameters in Table 1 we need to develop a library of functions that will be used by a program running on a remote host and communicating with the control boards over a local network. This program will serve the EPICS variables corresponding to different parameters using ChannelAccess protocol. The most commonly used programming languages for developing an EPICS ChannelAccess servers are C and C++, therefore the set of functions needs to be easily usable by C and C++ codes.

The methods that are required in the API can be divided into two types. The first type of the functions will be accessing and modifying the parameters listed in Table 1 that are related to a particular bias voltage channel. These functions are given in Table 2. The second type of functions would change or report the status of the communication between the server and individual



Short Name	Brief Explanation	Units	Access by EPICS
v_sp	Voltage setpoint	V	Read/Write
v_rb	Voltage readback	V	Read
v_max	Maximum allowed voltage	V	Read/Write
i_rb	Current readback	μA	Read/Write
i_trip	Trip current	μA	Read/Write
t_trip	Time before tripping	s	Read/Write
temp_rb	Temperature reading	$^{\circ}\text{C}$	Read
ramp_up	Ramp-up rate	V/s	Read/Write
ramp_dn	Ramp-down rate	V/s	Read/Write
status	Channel status	N/A	Read
enable	Enable/Disable channel	N/A	Read/Write

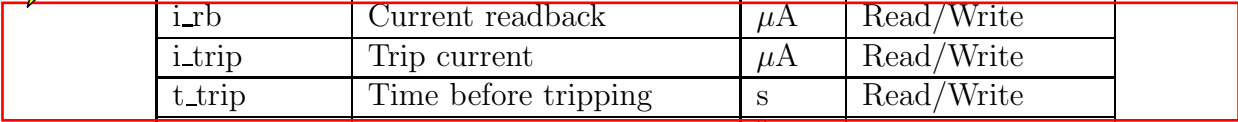


Table 1: List of the parameters than need to be controlled/monitored by EPICS for each microscope voltage channel.

boards (or ports). The list of functions operating on an individual control board are listed in Table 3. All the details of the communication protocol will be hidden from EPICS support allowing EPICS to open communication ports for each control board, to determine the list of channels available on each port, and to send requests for individual channel and receive responses using function calls. EPICS will call these functions asynchronously, that is the requests for each board will be queued, and when it is time for the request to be processed a callback function is processed in a separate thread. In general, the calls to the communication ports for the control boards will be made from multiple threads at the EPICS support level, therefore the API should have its own mutual exclusion scheme such that there are no collisions between the calls from multiple threads and that the dead-time of the communication ports due to locking of the shared resources on the server side is not prohibitively high. At the EPICS support level there will not be an attempt to prevent accesses to the communication ports from multiple threads.





Definition	Brief Explanation
<i>double GetVoltageSetpoint()</i>	Get Voltage setpoint as the return value.
<i>int SetVoltageSetpoint(double v)</i>	Set voltage setpoint to the value <i>v</i> .
<i>double GetVoltage()</i>	Get the measured voltage as the return value.
<i>double GetMaxVoltage()</i>	Get the maximum value for the setpoint as the return value.
<i>double SetMaxVoltage(double v)</i>	Set the maximum value for the setpoint to the value <i>v</i> .
<i>double GetCurrent()</i>	Get the measured current as the return value.
<i>double GetTripCurrent()</i>	Get the trip current as the return value.
<i>int SetTripCurrent(double i)</i>	Set the trip current to the value <i>i</i> .
<i>double GetTripTimeDelay()</i>	Get the trip delay time as the return value.
<i>int SetTripTimeDelay(double t)</i>	Set the trip delay time to the value <i>t</i> .
<i>double GetTemperature()</i>	Return temperature as the return value.
<i>double GetRampupRate()</i>	Get the ramp-up rate as the return value.
<i>int SetRampupRate(double r)</i>	Set the ramp-up rate to the value <i>r</i> .
<i>int GetRampdownRate()</i>	Get the ramp-down rate as the return value.
<i>int SetRampdownRate(double r)</i>	Set the ramp-down rate to the value <i>r</i> .
<i>long GetStatus()</i>	Get the encoded status word for the channel as the return value.
<i>int Enable()</i>	Enable voltage output for the channel.
<i>int Disable()</i>	Disable voltage output for the channel.
<i>bool IsEnabled()</i>	Returns <i>true</i> if channel is enabled, <i>false</i> otherwise.

Table 2: List of the methods related to a particular voltage channel needed for developing the EPICS support. The parameters specifying the channel are omitted here assuming the these are class methods. If C-interface is developed, the arguments for identifying the voltage channel will need to be added to the argument lists.



Definition	Brief Explanation
<i>void* Open(const char* addr)</i>	Open a communication port for the board with <i>addr</i> and return the pointer.
<i>int Close()</i>	Close the communication port for the board.
<i>int Reset()</i>	Close the communication port for the board.
<i>int IsConnected()</i>	Check the communication status and return it.
<i>vector GetChannelList()</i>	Return the list of channel identifiers on that board.

Table 3: List of the methods related to a particular control board needed for developing the EPICS support. The parameters specifying the board are omitted here assuming the these are class methods. If C-interface is developed, the arguments for identifying the control board will need to be added to the argument lists.



4 Summary

In this document we specified the requirements for the API that needs to be developed to integrate the bias voltage control into the Hall D experimental control system. This paper defines a list of parameters that should be implemented in the firmware as well as a set of functions that are needed for EPICS device support (see Table 2 and Table 3). The software library will need to be compiled and run on a Linux-based system, so it would be nice to require only external libraries that are part of the major Linux distributions.

