Gretina user documentation

Ron Fox

Gretina user documentation

by Ron Fox

Revision History

Revision 1.0 May 9, 2012 Revised by: RF
Original Release

Table of Contents

List of Figures

Chapter 1. Introduction

This document describes how the S800 and GRETINA data acquisition systems are integrated at the NSCL. The intended audience of this document are experimental users that must configure and set up experiments to run with the S800 and GRETINA as coupled detectors.

The remainder of this document is organized as follows:

1. Background on GRETINA and the S800 data acquisition systems are provided. Local contacts for both systems are provided as well.
2. A recipe book is provided that describes how to hook the two systems together in a way that integrates them with existing software you might have.
3. The upper level event structure of final stage event builder is described.
4. Some reference material is provided for the components that marry the two systems systems together.

Chapter 2. Background on GRETINA and the S800 data acquisition systems

This chapter provides background on the GRETINA and S800 data acquisition systems. The emphasis is on the chain of processing in each system that results in event data exported to the outside world. How the systems are controlled will also be touched on as will the overarching strategy employed to marry these two systems together.

Please refer questions about the s800 system details to either Daniel Bazin (bazin@nscl.msu.edu (mailto:bazin@nscl.msu.edu)) or Scott Wiliams (williasc@nscl.msu.edu (mailto:williasc@nscl.msu.edu)). Please refer questions about the GRETINA system details to Dirk Weisshaar (weisshaa@nscl.msu.edu (mailto:weisshaa@nscl.msu.edu)).

2.1. The GRETINA Data acquisition system

The GRETINA data acquisition system is a complex digital data acquisition system that uses sophisticated waveform analysis algorithms to reconstruct hit positions and gamma ray tracks through an array of segmented HPGe detectors. From the standpoint of this document, however, we can view GRETINA as consisting of the following components:

- A distributed, synchronized time-keeping system that is able to label event fragments from each source in the system with a timestamp. This subsystem also provides hooks to synchronize timestamps systems external to GRETINA (auxilliary detectors in GRETINA terminology).
- A slow controls framework for setting up, starting and stopping runs.
- A Global Event Builder that accepts event fragments from data sources, possibly out of total time order, and produces an output stream of data that are totally ordered by their timestamps.
- An output tap that allows systems external to GRETINA to request sampled data from the global event builder.
- All of this is wrapped up in a large computing cluster (visualize 3 racks full of linux systems), that run on a 'local subnet' behind a firewall that is between the NSCL network and the GRETINA data acquisition/computing cluster network.

The Gretina time keeping system provides an *Imperative synch pulse* when it is ready to take data. That pulse is intended to tell al timestamp counters to reset to zero. A clock pulse train is passed around to the various components of GRETINA and auxiliary detectors.

EPICS (Experimental Physics and Industrial Control System) is used to control and setup the data taking conditions of the detector. EPICS control panels that run on parts of the EPICS computing cluster. These control panels allocate functions to processors in the cluster, turn on or off and parameterize procesing

stages. EPICS *Process Variables* (PVs) are also defined to control the run state, to determine the cluster state and to manage signals that coordinate with auxilliary detectors.

Gretina digitizer continually sample wave forms and look for pulses that may be interesting. An *Accept* pulse (which can come from outside GRETINA) initiates the readout of interesting pulses. These are transmitted to the Gretina Global event builder, they also are processed by the decomposition software which contributes hit data to the global event builder with the timestamnp preserved.

The global event builder (GEB) itself is a TCP/IP server with a well defined protocol and client application programming interface. This allows auxiliary detectors to contribute other event fragments to the GEB as long as they can produce a synchronized timestamp for the event and have access to the internal GRETINA subnetwork.

An important point is that each fragment sent to the GEB is identified by a timestamp and by a data type. For example Waveform data is type 2, Decomposition data type 1. Other data types have been defined for various auxiliary data fragments. The output if the GEB is an ordering of all fragments by timestamp regardless of data type.

The GEB provides an output tap that allows clients that are not part of the GRETINA data flow to sample data from the detector. When sampling data from the GEB, clients specify a mask that defines the set of data types they want. Each client request asks for a maximum number of fragments and receives at most the requested number of fragments. There is no sampling within the set of requested fragments, however data that may come between requests is not buffered for tap clients and they will miss those fragments.

2.2. The S800 Data acquisition system.

The readout system for the S800 was recently (April/May 2012) redesigned and implemented to provide performance improvements required for some experiments that have been forecast to run with the S800 and GRETINA. This section describes this new system with a focus on how it interfaces with the outside world.

The new system features VM-USB and CC-USB controllers from Wiener/JTec. These are list processing controllers that can autonomously react to trigger by performing a list of operations that read out the portion of an event from the interface bus (CAMAC or VME) in which they are installed.

The system currently uses a single CAMAC crate containing some of the digitizers for the S800 focal plane detector package and a single VME crate which has the XLM's that read the CRDC FEE boards. The VME crate is also intended to read out the Image 2 detector package at a later point in time.

Triggers are assigned a timestamp via programmable logic units in the crates. These timestamps are synchronized at the start of the run and increment from a shared external clock signal.

Buffers of event fragments then are read by an event builder which matches up triggers that are in coincidence and reformats the data to retain a high degree of compatibility with the old S800 event format. The event builder includes a TCP/IP server. Clients of that server receive the fully assembled/formatted events the event builder creates.

2.3. How the S800 and Gretina systems are integrated

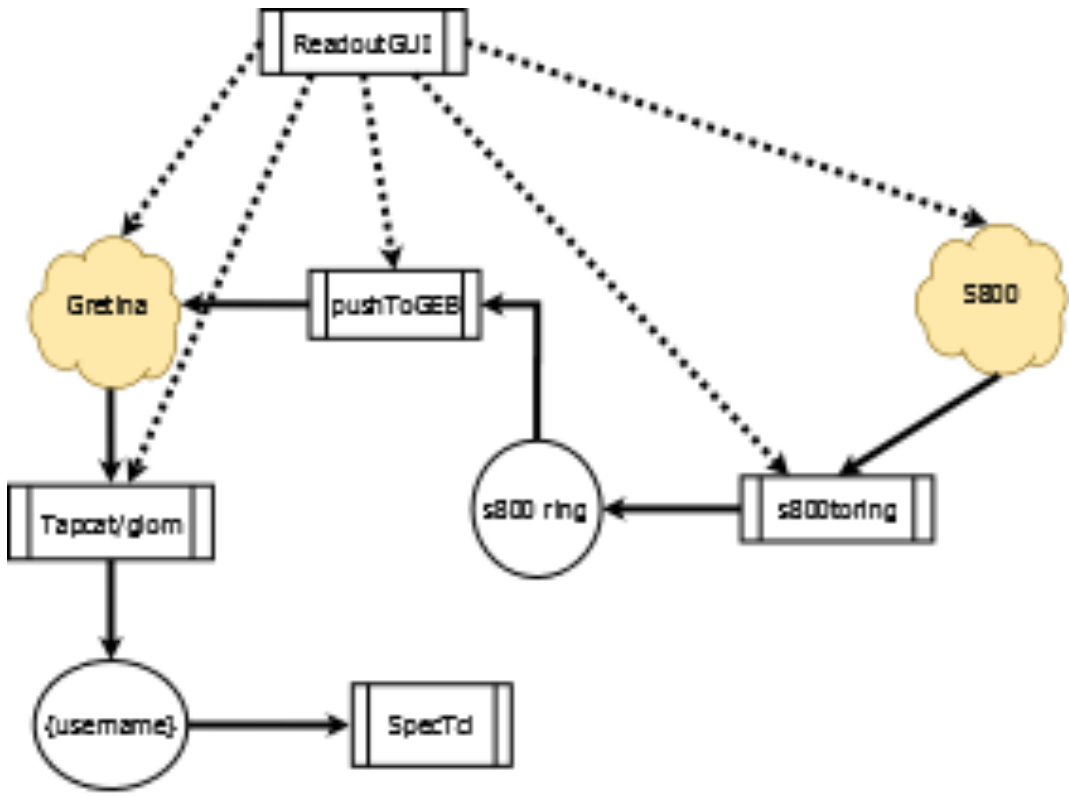
Integrating the S800 and Gretina systems means:

- Having a method to acquire data from the S800 event builder and push it in to Gretina in the form expected by the Gretina GEB.
- Having a method to catch data from Gretina and build events from a coincidence window so that SpecTcl code can do p-gamma coincidence analysis.
- Having a mechanism to start and stop the two systems in a roughly synchronized manner and to get Gretina to pulse its imperative synch at the appropriate time.

  This mechanims must also start all Gretina global event builder clients as the Gretina GEB drops all connections with clients at the end of a run.

The figure below shows a block diagram of the entire system. In the figure clouds represent complete data acqusition systems (Gretina, S800). Circles represent NSCLDAQ Ring buffers. Rectangles with represent schematic processes (in general a schematic process may actually be a unix pipeline of processes). Furthermore, solid lines represent data flow and dashed line represent control flow.

Figure 2-1. System Block diagram



Let's start by looking at the control flow as that's critical as well to getting some of the elements in the picture to run.

The ReadoutGUI's ReadoutCallouts mechanism is used to extend the readout gui to control both Gretina and the S800. `OnBegin` is what actually will start data taking in the S800 and in Gretina. Since the Gretina Global Event Builder drops connections to its clients at the end fo the run, the Gretina part of `OnBegin` starts the boxes labeled `pushToGEB` and `Tapcat/glom`.

Those items complete the dataflow elements which operate as follows:

- s800toring accepts data from the S800 local event builder and creates NSCLDAQ ring items which are, by default, pushed into the ring named `s800`.
- From there, pushToGEB takes events, extracts their timestamps and uses the Gretina GEB client API to push those events to the Gretina Global Event Builder.
- Tapcat/glom samples bursts of contiguous totally time ordered data from the Gretina Global Event builder and creates ring items in a ring that is named after the users's login. The user's SpecTcl can, in turn sample data from that ring for analysis.

`OnEnd` conversely ends the run in both Gretina and the S800. This causes pushToGEB and Tapcat/glom to exit. The s800toring process is persistent, however. It is started when the ReadoutGUI initializes.

Chapter 3. The Gretina/S800 cookbook

This chapter contains a cookbook of recipes for the merged S800/Gretina data acquisition system. Included are descriptions of

- How to add the gretina/s800 controls to the Run Control GUI.
- How to modify the behavior of components of the system.
- How to get S800/Gretina merged data into SpecTcl.
- How to monitor the S800 data from the event builder in SpecTcl.

3.1. Using ReadoutGUI with the s800/gretina daqs.

Recipe 1: ReadoutCallouts.tcl. Add the following to your experiment's `ReadoutCallouts.tcl` file to integrate it with Gretina and the S800 DAQ

```
package require mergedCallouts
mergedCallouts::Initialize
...
proc OnBegin runNumber {

    # Stuff to do before the start of the run.
        ...

    # Start the run:

    mergedCallouts::OnBegin

    # Stuff to do after the run.
        ...
}
proc OnEnd runNumber {
    # Stuff to do before the run ends.
        ...
    # Actually end the run:

    mergedCallouts::OnEnd

    # Stuff to do after the run ends:
    ...
}
```

Recipe 2: Starting the ReadoutShell. The following command lines should be used on spdaq44 to start the ReadoutShell (ReadoutGUI) for the merged system.

```
. /opt/lucid/daq/gretina/bin/gretinaSetup
/opt/lucid/daq/10.1/bin/ReadoutShell -host=spdaq48 \
                -path=/opt/lucid/daq/10.1/dummyrdo -nomonitor
```

The first line unsets EPICS environment variables that prevent EPICS requests from propagating onto the GRETINA network.

`dummyrdo` is a dummy Readout program that just throws away commands it is sent. The actual readout programs are free standing. The `-nomonitor` option is new option that handles the run time clock a bit differently than before. Specifically scaler events are not monitored to correct the run time.

3.2. Modifying the behavior of the components.

This section describes environment variable settings that can alter the behavior of the components controlled by the ReadoutGUI.

Recipe 3: Selecting the Ring name that will get assembled data. By default Gretina/S800 merged data goes into a ringbuffer that whose name is the username of the account that is running the ReadoutGUI. Normally this is what you want. If, however you want the data to go to a different ring, prior to running the ReadoutGui execute the following command:

```
export RINGNAME=myring
```

The `RINGNAME` environment variable if set will specify the actual destination ring buffer name (`myring` in this case).

To remove this environment definition:

```
unset RINGNAME
```

Recipe 4: Setting the coincidence window. Gretina Global Event Buider data is a stream of totally time ordered event fragments. An element of the tapcat pipeline, called `glom` glues together event fragments that are within a specific time interval into single events.

The default coincidence interaval is 200 gretina timestamp ticks which corresponds to 2 microseconds. To set this interval to 3 microseconds, issue the following command prior to starting the ReadoutGUI:

```
export COINCIDENCE_TICKS=300
```

Recipe 5: Changing the S800 ring name. Data caught from the S800 event builder are put into a ringbuffer. Arbitrary clients can spy on the data in this ring, however its primary purpose is to distribute data to the pushGEB pipeline. To change the S800 output ring name you can type the following command prior to starting the Readout GUI:

```
export S800RING=mys800ring
```

This sets the S800 data ring to `mys800ring`

Recipe 6: Selecting the S800 event builder host. At present the S800 event builder lives on spdaq48. If that changes, you can tell the ReadoutGUI where to point the S800 catch software:

```
export S800_HOST=s800evb.nscl.msu.edu
```

Tells the Readout GUI to expect the S800 event builder to be running on the host `s800evb.nscl.msu.edu`

3.3. Reading merged data online in SpecTcl

Merged data is, by default sent by the tapcat pipeline to a ring whose name is the same as the username of the account that started the ReadoutGUI. These recipes describe different ways to connect SpecTcl to that ring.

Recipe 7: Connecting to merged data in native format. The new rewritten multicolored GUI has a mechanims for connecting to ring buffers. Click on the Online (spectrodaq)... menu item of the Data Source menu. In the resulting dialog box, type the hostname of the system in which ReadoutGUI (normally spdaq44) is running, check the `ring` format radio button and click the Ok button.

Recipe 8: Connecting to merged data in compatibility format. Compatibilty mode is a pipe data source that converts ring buffer items into old-style NSCL data buffers. See Appendix C of http://docs.nscl.msu.edu/daq/ringtutorial/index.html (http://docs.nscl.msu.edu/daq/ringtutorial/index.) for more information about the buffer compatibility utilities.

The following bit of Tcl connects SpecTcl to the merged data ring in compatibility form. The

```
attach -pipe /usr/opt/daq/10.1/bin/spectcldaq tcp://spdaq44/$::tcl_platform(user)
start
```

The global array `tcl_platform` contains several platform specifici elements. The `user` element is the user's login name. The `tcp:` URL like thing is the URI of the ring in spdaq48 that was named the same as the logged in user.

Recipe 9: Getting native mode merged data from a non-standard ring. This recipe assumes you have modified the ring name to `mygretina`. The following commands attach to that ring in spdaq48 and start data analysis.

```
attach -format ring -pipe /usr/opt/daq/10.1/bin/ringselector \
                    --source=tcp://spdaq44/mygretina         \
                    --sample=PHYSICS_EVENT
start
```

The `--sample` option ensures that SpecTcl is only fed a sample of the event data rather than allowing it to be the rate bottle neck.

3.4. Monitoring S800 event data in SpecTcl.

Since there is a ringbuffer between the process that catches data from the S800 event builder and the process that pushes event fragments to the Gretina Global Event Builder, it is possible to attache SpecTcl to a source of pure S800 data.

Recipe 10: Getting data from the S800 in native mode. Since the Readout GUI's Data Source button does not yet have a provision to supply the ring name, you need to provide explicit Tcl code to connect to the ring. Typically this code looks like:

```
attach -format ring -pipe /usr/opt/daq/10.1/bin/ringselector \
                    --source=tcp://spdaq44/s800         \
                    --sample=PHYSICS_EVENT
start
```

Recipe 11: Getting compatibility data from the S800. As in Recipe 8 we can use the compatibility tools for this:

```
attach -pipe /usr/opt/daq/10.1/bin/spectcldaq tcp://spdaq48/s800
start
```

Chapter 4. Structure of merged data

This chapter describes the format of data as it appears in native mode in the merged data ring. The data from this ring is in standard ring buffer format. See http://docs.nscl.msu.edu/daq/ringtutorial/index.html (http://docs.nscl.msu.edu/daq/ringtutorial/index.html) Appendix A for information about the general structure of ring items.

This chapter will focus on describing the format of the body part of `PHYSICS_EVNET` ring items.

The first uint32_t in each event is a self inclusive size of the entire event. Following that will be a number of items that look like NSCL event packets with event type 0x66eb (squint and the 6's look likg G's if you want to know why we chose that tag).

Each packet has the following outer general format

```
#include <stdint.h>
struct ggebPacket {
    uint16_t packetSize;
    uint16_t packetTag
    uint32_t gretinaType;
    uint32_t payloadLength;
    uint64_t timestamp
    uint16_t payload[];
};
```

`uint16_t packetSize;`

The packet size in uint16_t units.

`uint16_t packetTag;`

The packet id that indicates this is a Gretina Global Event Builder. This value will always be `0x66eb`

`uint32_t gretinaType;`

Type of gretina data. These are defined in `/opt/lucid/daq/gretina/include/GEBLink.h` For the most part you will get `GEB_TYPE_DECOMP`, output of the decomposition system and `GEB_TYPE_S800` S800 focal plane data.

`uint32_t payloadLength;`

The amount of data in bytes (uint8_t) in the payload of the event.

`uint64_t timestamp;`

The timestamp of the event fragment.

The `payload` field is a placeholder for the body of the event fragment. The format of the `payload` data depends on the type of data. For Gretina data types, see the Gretina documentation. For S800 data see https://groups.nscl.msu.edu/opdevtech/wiki/index.php/S800_VERSION_0x0005 (https://groups.nscl.msu.edu/opdevtech/wiki/index.php/S800_VERSION_0x0005).

Chapter 5. Reference material

s800toring

Name

`s800toring` — Put data from S800 event builder in a ringbuffer

Synopsis

/opt/lucid/daq/10.1/bin/s800toring ***host port ring-name***

DESCRIPTION

Starts a pipeline of processes which take data from the S800 event builder and push them into a ring.

`host` is the host on which the event builder is running. *`port`* is the server port on which the event builder is listening for client connections. *`ring-name`* is the name of a local ring into which the ring data will be put.

This command is implemented as a shell script which starts the following unix pipeline in order from start to finish:

`netcat.tcl`

This is a Tcl script that works like the unix utility `netcat` however its buffering is much more suited to the buffer sizes the s800 event builder uses.

`BufferToRing`

This is a program that accepts NSCL Buffered data on stdin and emits ring items on stdout.

`stdintoring`

This is a program that accepts ring items on stdin and inserts them into a target ring.

EXAMPLES

The example below is the typical invocation of the program:

Example 5-1. Standard way to run s800toring

```
/opt/lucid/daq/10.1/bin/s800toring spdaq48 9002 s800
```

pushToGeb

Name

`pushToGeb` — Push event fragments to Gretina Global Event Builder.

Synopsis

/opt/lucid/daq/gretina/bin/pushToGeb ***?options?***

DESCRIPTION

This program accepts data from an NSCL ring buffer, extracts timestamps from each physics event and sends those events to the Gretina Global Event Builder (GGEB). For non physics events, a timestamp of 0 is provided as agreed upon with the Gretina collaboration. Those get assigned an arbitrary timestamp by the GGEB

The software that extracts the timestamp is a dynamically loaded shared object. This allows the same program to be used with a variety of event structures by simply writing a timestamp extraction function, building as a shared library and telling **pushToGeb** to use it. See TIMESTAMP EXTRACTION below.

See OPTIONS below for more information about how to use this program.

OPTIONS

Each of the options below has a 'short option' version as well. use the `--help` option to get a brief help from the progrm which will list the short options as well as long options. Since the long options are more readable it is strongly recommended that you use those however.

`--help`

Outputs a brief help text to `stdout`. Note that if `--help` is specified, no further options are processed and the program will exit after printing the help text.

`--version`

Ouptuts the program name and version to `stdout`. If `--version` is used, no further options are processed and the program will exit after printing the version number.

`--usercode=`*`timestamp-extractor`*

Provides the path to a shared library that has the timestamp extraction code for the events that are going to be pulled from the ring. *`timestamp-extractor`* must be a path to a shared object library that contains that code. See TIMESTAMP EXTRACTION below for more information about how to write that information.;

`--ctltype=`*`GGEB-type`*

All event fragments submitted to the GGEB have a type. This is a small integer that identifies what that fragment's payload represents. The types that are registered are listed in `/opt/lucid/daq/gretina/bin/GEBLink.h`.

This option allows you to override the default GGEB type we have chosen to use for non-physics data (type 6). Non physics data includes run state changes and scaler events.

`--phystype=`*`GGEB-type`*

Sets the GGEB event type to use to tag physics events. By default this is `5` which is what we agreed to use. Be absolutely sure you know what you are doing if you change this.

`--gebhost=`*`hostname`*

Specifies the GGEB hostname. Normally this comes fromt he output of the **gebnode** command which outputs the value of the EPICS process variable that contains the name of the host in which the GGEB is runnning.

`--gebport=`*`port`*

Specifies the TCP/IP port on which the GGEB is listening for connection. Normally this is the output of the **gebport** commande which outputs the value of the EPICS process variable that contains the GGEB port number.

`--source=`*`ringURL`*

Specifies the URL of the NSCLDAQ ring from which events will be pushed.

TIMESTAMP EXTRACTION

pushGEB requires code to extract the timestamp from each event. In order to maintain event format independence, this operation is delegated to user code. The user code must be built as a shared object library which will be specified `--usercode` option value.

The user code must be a C (not C++) function named `extractTimestamp`. The argument signature is as follows:

```
long long
extractTimestamp(void* pEvent);
```

The *`pEvent`* parameter is a pointerr to the event. The return value of the function is the timestamp that will be used by the GGEB to order this event fragment with the other event fragments it is emitting.

You can build a shared library by specifying the `-shared` option on the **gcc** command line used to build your code.

EXAMPLES

The following is the typical way **gebPush** id started e.g. by ReadoutGUI's ReadoutCallouts.

Example 5-1. Typical way to start gebPush

```
/opt/lucid/daq/gretina/bin/pushToGeb \
    --usercode=/opt/lucid/daq/gretina/lib/libs800.so \
    --gebhost=`/opt/lucid/daq/gretina/bin/gebnode`   \
    --gebport=`/opt/lucid/daq/gretina/bin/gebport`   \
    --source=tcp://localhost/`whoami`
```

If you are not that familiar with bash and its related shells. ‘command‘ runs a command and substitutes what it wrote to `stdout` on the command line where the command had been.

tapcat

Name

`tapcat` — Copy Gretina output TAP data to stdout.

Synopsis

/opt/lucid/daq/gretina/bin/tapcat *`options...`*

DESCRIPTION

Outputs data from the Gretina output tap to `stdout`. This is intended to be the first stage in a pipeline that does something with Gretina output tap data. The normal use case is to have subsequent stages build events from the ordered stream of event fragments and send those events into NSCLDAQ's data distribution system (a ringbuffer).

For information about the program options see OPTIONS below. To see how this program is normally run' see EXAMPLES.

OPTIONS

`--help`

Outputs a brief help message to `stdout` that describes program usage. Once this option is seen, no later options are processed and the program exits immediately after outputting the help message.

`---version`

Outputs the program name and current version to `stdout`. If this option is present on the command line, no later options are processed and the program exits after the information is output.

`--host=`*`taphost`*

Specifies the host on which the tap server is running. At this time the output tap is part of the Gretina Global Event Builder (GGEB).

Normally the output of **gebnode** is used.

`--type=`*`mask`*

Supplies a bitmask of the data types that are desired. If this is `0` all GGEB types will be gotten. Typically this will be 0xfd which requests all types except the raw waverforms.

`--grouping=`*`fragments`*

The GGEB output tap provides clumps of contiguous events fragments. In between these clumps the output tap client may miss event fragments. This parameter indicates the maximum number of event fragments that will be put in a clump.

This currently defaults to `10` however it is likely this will be revised upwards to `100` as that seems to produce better results. If the value is too small, it may be difficult to assemble complete events due to skipped fragments between clumps.

`--timeout=`*`seconds`*

The maximum number of seconds the tap client library will wait for a group of fragments to come in before returning what it got already to the caller.

glom

Name

`glom` — Glom event fragments together to create events.

Synopsis

/usr/opt/daq/gretina/bin/glom ***`options...`***

DESCRIPTION

The Gretina Global Event Builder (GGEB) produces a stream of totally time ordered event fragments. **glom** Creates NSCL old-style event buffers whose events consist of event fragments that satisfy a coincidence time interval. This allows Gretina data to be analyzed as something other than singles data.

glom also understands the existence of non-event data and builds appropriate non-event buffers from them.

OPTIONS

`--help`

Outputs a brief help message to `stdout` that describes program usage. Once this option is seen, no later options are processed and the program exits immediately after outputting the help message.

`---version`

Outputs the program name and current version to `stdout`. If this option is present on the command line, no later options are processed and the program exits after the information is output.

`--dt=`*`ticks`*

Defines the number of timestamp ticks that make up a coincidence window.

`--controltype=`*`GGEBtype`*

Defines the Gretina Global Event Buidler (GGEB) data type that has been assigne to non physics data. All other data are assumed to be event data. This option is mandatory.

`--tag=`*`NSCL-packet-tag`*

The output event format includes a set of NSCL packets, one for each fragment that has been built into an event. This defines the id that will be used for the packet. The default is `0x66eb`.

BufferToRing

Name

`BufferToRing` — Turn NSCL buffered data into ring items.

Synopsis

/opt/lucid/daq/10.0/bin/BufferToRing [buffer-size-in-bytes]

DESCRIPTION

Converts a stream of NSCL old-style buffers on `stdin` into a stream of ring items that can be piped to **stdintoring** for insertion into an NSCLDAQ ring buffer.

The *`buffer-size-in-bytes`* overrides the default size of the buffers on the input stream (8192).

gebinfo

Name

`gebinfo` — Global event builder information.

Synopsis

gebnode

gebport

DESCRIPTION

Gretina maintains information about the Gretina Global Event Builder in EPICS process variables. Specifically the host on which the GGEB runs and the port on which it is listening for clients that contribute fragments. The values of these can vary from run to run.

gebnode outputs the host on which the GGEB is running. **gebport** outputs the port number. A typical use of these functions is in back-tick bash command substitutions.