

[LVC Portal](#) [GEO](#) [LIGO](#) [LSC](#) [VIRGO](#) [Help](#)welcome: AlexanderUrban | [settings](#)[Refresh my LIGO group memberships](#)[Wiki\\_Home](#) > [GRBreview](#) > [GCNListener](#)

## GCN Listener Service

 [Brief overview presentation](#)

### Contents

1. [GCN Listener Service](#)
  1. [Personnel](#)
  2. [Description](#)
  3. [Software Location](#)
  4. [Software Dependencies](#)
  5. [Running the service](#)
  6. [Event Streams](#)
  7. [Parsing VOEvents and Alert Types](#)
  8. [Pseudocode](#)
  9. [Unit Tests](#)
  10. [Open Issues](#)
  11. [Other Important Links](#)

## Personnel

**Project Lead and Primary Contact:** Alex Urban**Core Contributors:** Alex Urban, Patrick Brady, Leo Singer, Roy Williams

## Description

Given the very rapid (~seconds to hours) timescale over which gamma-ray bursts (GRBs) and their X-ray and optical afterglows occur, good multimessenger astronomy requires collaboration between LIGO, Virgo and astronomer colleagues to happen as quickly as possible. Very low latency searches for gravitational waves (GWs) from compact binary mergers, for example, have a demonstrated ability to provide triggers within ~10 seconds of the coalescence time, while GRB notices are provided directly from satellite observatories over the Gamma-ray Coordination Network (GCN) within ~1 minute of a typical GRB. For the purpose of both the low-latency coincidence search with external triggers and the medium-latency triggered coherent searches, a pathway through the GCN over which such external

triggers may be supplied is mission-critical. Such a pathway exists in the form of the GCN Listener service, where users may subscribe for real-time alerts in the event of something like a GRB trigger. In LIGO's case, this service is used to upload electromagnetic events to GraceDB and trigger the aforementioned searches for associations with GW candidates. Information uploaded with these events includes observatory, event time, sky location, figures of merit (including an SNR based on gamma-ray counts), and position of the Sun, among other things.

Please see the [GRB search plan](#) for more information.

## Software Location

---

GCN listener code currently lives in the GRiNCH (Gravitational-wave Candidate Event Handler) git repository, which is hosted on LIGO-VCS at

```
ligo-vcs.phys.uwm.edu:/usr/local/git/grinch.git
```

and is visible on the LIGO-VCS [cggit](#). Code relevant for this piece of the GRB infrastructure lives under the `grinch` Python module, and the submodule that processes, filters, and handles events is called `gcnhandler.py`. A script, `pygcn_listen`, then connects to an external socket over the GCN and makes calls to this handler for every new notice that comes through. Finally, a second wrapper script, `gcn_listener`, submits and checks the status of `pygcn_listen` via Condor.

**Note:** At present, this service is operated under the GraceDB Processor umbrella; GraceDB Processor will eventually be the subject of its own code review.

## Software Dependencies

---

Outside the standard, cluster-wide system installations of LIGO-specific software such as LALSuite, the GCN Listener has the following dependency:

- PyGCN, an anonymous GCN/TAN client for receiving GCN notices in XML format (Version 0.1.4; <https://pypi.python.org/pypi/pygcn>)

An up-to-date version of `pygcn` is installed locally on the `gdb_processor` account at UWM, and could easily be migrated to another cluster.

**Note:** The service also depends rather strongly on VOEventLib, an Open Source XML-parsing Python module useful in parsing the VOEvent (Virtual Observatory Event) alerts containing GRB information, but this software is installed cluster-wide as part of the LDG software stack.

## Running the service

---

The listener service is run from the `gdb_processor` account on the Nemo cluster at UWM, specifically on `greatdane.cgca.uwm.edu`. (This run location is being changed to improve long-term stability. All production GraceDB processor services will be moved to a dedicated machine before ER7, probably on the CIT cluster, while `greatdane` at UWM will continue to operate as a full-scale test machine on which new code releases (such as the GraceDB project code) can be tested end-to-end, including submission to a test instance of GraceDB.

Once all dependencies have been installed, running the listener is as simple as executing

```
[gdb_processor@pcdev2:~/] $ pygcn_listen
```

with no options or flags. This will connect to only the primary GCN node at NASA Goddard (see below) and print all logging to stderr. (Feature requests for the imminent future includes support for multiple alert nodes and custom-named rolling log files.) To interface with Condor, simply execute

```
[gdb_processor@pcdev2:~/] $ gcn_listener start
Submitting job(s).
1 job(s) submitted to cluster xxxxx.
```

Passing the `status` command will query condor for all jobs with your username:

```
[gdb_processor@pcdev2:~/] $ gcn_listener status

-- Submitter: gdb_processor@nemo.phys.uwm.edu : <192.168.5.2:54008> :
pcdev2.nemo.phys.uwm.edu
  ID      OWNER      SUBMITTED    RUN_TIME ST PRI SIZE CMD
xxxxx.0  gdb_processor  4/1  19:14    0+00:00:04 R  0   0.0  pygcn_listen
```

Finally, this (and all other!) jobs may be stopped by passing the `stop` command:

```
[gdb_processor@pcdev2:~/] $ gcn_listener stop
Are you sure?? This will kill all condor jobs for this user... [y/n] n
I thought as much!
```

```
[gdb_processor@pcdev2:~/] $ gcn_listener stop
Are you sure?? This will kill all condor jobs for this user... [y/n] n
Righty-ho, then. Exterminate!
All jobs for user gdb_processor marked for removal.
```

Of course, the `--help` flag explains all of this:

```
[gdb_processor@pcdev2:~/] $ gcn_listener --help
usage: gcn_listener [-h] {start,stop,status}

Management script for the real-time GCN listener service.

positional arguments:
  {start,stop,status}  Launch, kill or check the status of GCN listener
                        tasks

optional arguments:
  -h, --help            show this help message and exit
```

A file cache for all saved events, including files with information that does *not* get uploaded to GraceDB

(e.g. pointers to light curves), is stored on-disk at

UWM: `~gdb_processor/working/gcn_listener/cache/`.

These are further indexed by the `role` that appears in the corresponding VOEvent file; for all *bona fide* GRBs, this `role` variable has the value "observation."

In the interest of general use, in the near-term future (i.e., prior to ER7) this cache location will be made locally configurable. Finally, note all events that were saved prior to September 2014 are visible under

UWM: `~gdb_processor/Users/aurban/gracedb-voevent/comet/comet/cache`.

## Event Streams

---

The GCN listener connects to a hard-coded set of *streams*, each one corresponding to a specific observatory. At present, the streams we connect to are for Swift (where alerts come from the Swift Burst Alert Telescope instrument), Fermi (where alerts come from the Fermi Gamma-ray Burst Monitor instrument) and SNEWS (i.e., the Supernova Early Warning System, a network of neutrino detectors designed to provide rapid notices of galactic and nearby supernovae). The listener is currently plugged in to the following alert node:

1. `68.169.57.253:8099` (Primary GCN node at NASA GSFC)

For the past month of notices, listening to only this node has been sufficient. However, the earlier Comet/Twisted version of the code had support for multiple alert nodes, and we also tuned into

1. `voevent.phys.soton.ac.uk:8099` (Southampton University)
2. `voevent.transientskp.org:8099` (LOFAR)

In the near-term future we plan to restore this functionality in the PyGCN version.

In the case of Swift, bursts arrive already having been classified as a GRB (that is, with an official designation as "GRB 141117A" or the like). In the case of Fermi, however, assigning a designation can take a good deal more time, and at any rate a designation is not typically available when we ingest an alert. Fermi triggers are sub-classified by their probable source, which is captured within the code by a variable called `Fermi_Likely`.

The `Fermi_Likely` variable may take the following values:

0. An error has occurred
1. `UNRELIABLE_LOCATION`: Location not trusted
2. `PARTICLES`: Local particles, equal rates in opposite detectors
3. `BELOW_HORIZON`: Distant particles, assumed to come from the horizon
4. `GRB`: Burst with good localization
5. `GENERIC_SGR`: Soft Gamma Repeater (except 1806-20)
6. `GENERIC_TRANSIENT`: Astrophysical transient of unknown class
7. `DISTANT_PARTICLES`: Particles at a distance
8. `SOLAR_FLARE`: This is a Solar Flare event
9. `CYG_X1`: Thi: This trigger came from SGR 1806-20

10. GROJ\_0422\_32: This trigger came from GRO J0422-32
11. unrec\_value: Unrecognized value
12. TGF: Terrestrial Gamma Flash

Finally, in the case of SNEWS, all alerts are accepted but there is some question as to whether they're actually being *broadcast* from the observatories themselves.

## Parsing VOEvents and Alert Types

---

The following links are maintained by NASA and the GCN system overlords, and are organized by observatory:

- [Swift](#)
- [Fermi](#)
- [SNEWS](#)

These are the documentation pages that detail the various alert types, as well as what kind of information is contained in the VOEvent file that gets received by our listener. In brief, we are interested in six types of event notice:

Instrument	Notice Type	Latency	On-sky Precision	Description	VOEvent Variables Used	Reliability*
Swift	61 (BAT_GRB_Pos)	7-30 seconds	~arcminutes	A significant trigger from the <i>Swift</i> Burst Alert Telescope, whose position has been determined with on-board software	IVORN, role, designation, trigger time	Almost always a real GRB
Fermi	110 (GBM_Alert)	~5 sec	(No Localization)	A significant trigger from the <i>Fermi</i> Gamma-ray Burst Monitor, and its <b>timestamp</b> has been identified	IVORN, role, trigger time	Sometimes found to be spurious
Fermi	111 (GBM_Flt_Pos)	~10 sec	~15 deg.	A localization has been calculated with in-flight software	IVORN, role, Fermi_Likely, trigger time	Sometimes found to be spurious; no account of systematic sky position uncertainty
Fermi	112	20-300	1-10 deg. (+3 deg.	A better localization has been calculated	IVORN, role, Fermi_Likely,	Almost always a

	(GBM_Gnd_Pos)	sec	systematic)	with automated on-the-ground software	trigger time	real GRB
Fermi	115 (GBM_Fin_Pos)	~2 hr	1-3 deg. (+3 deg. systematic)	The best available localization has been calculated with a human-in-the-loop	IVORN, role, trigger time	Almost always a real GRB
SNEWS	N/A	N/A	N/A	A significant trigger from the Supernova Early Warning System (SNEWS) network of neutrino detectors	IVORN, role, trigger time	N/A

\* This information comes from speaking with Valerie Connaughton of the Fermi science team.

In terms of information content in these VOEvent packets, our event filter utilizes the following pieces of information (for more information, see Pseudocode, below):

- The IVORN
- The event `role`
- The official designation (e.g. GRB 150304), if there is one
- The trigger time, in ISO 8601 format (which is measured to millisecond precision)

Note the IVORN and role are used to identify an event portfolio on-disk, the designation is used to select a more useful file name, and the trigger time is used to identify other bursts in GraceDB at the same time. Fermi notices of type 111 and 112 also use the `Fermi_Likely` variable.

## Pseudocode

---

Here is an outline of the algorithm used to filter alerts that come in over the GCN.

To begin with, what is fed to the event catcher script (`eventcatch.py`) is a GCN Notice: a real-time alert announcing some recent event. Notices can be either wholly new events, or updates to Notices from existing events with new information (e.g. light curves or better sky localization). The Notice is ingested as an XML-parsable string in the VOEvent (Virtual Observatory Event) format, and passed through `eventcatch.py`.

When the event string is ingested, it is processed by the following algorithm:

1. Parse the VOEvent string.
2. Grab this event's IVORN (International Virtual Observatory Resource Name), which acts as a key to the registry that stores actual information and behaves like a standard URL. For example, the IVORN `ivo://nasa.gsfc.gcn/SWIFT#BAT_GRB_Pos_618673-699` would resolve to something like an FTP server (`nasa.gsfc.gcn/SWIFT`) where `/BAT_GRB_Pos_618673-699` is a physical location.
  - A. IF the IVORN is of an illegal form OR the IVORN is from a stream other than Swift, Fermi or SNEWS, THEN reject this event immediately.
3. Grab this event's `role`, which classifies its science status (possible values are `observation`,

utility and test).

- A. IF the role is utility OR (the role is test and not from SNEWS), THEN reject this event immediately.
4. Grab this event's packet type:
  - A. IF the stream is Swift THEN:
    - i. IF packet type is 61, THEN it is a `BAT_alert`. Flag for saving/sending to GraceDB!
    - ii. ELSE: This alert is not interesting to us or is a test of some kind. Reject this event.
  - B. IF the stream is Fermi THEN:
    - i. IF packet type is 110, THEN it is a `GBM_Alert`. Flag for saving/sending to GraceDB!
    - ii. IF packet type is 111, THEN it is a `GBM_Flt_Pos`. Flag for saving/sending to GraceDB! AND print `Fermi_Likely`.
    - iii. IF packet type is 112, THEN it is a `GBM_Gnd_Pos`. Flag for saving/sending to GraceDB! AND print `Fermi_Likely`.
    - iv. IF packet type is 115, THEN it is a `GBM_Fin_Pos`. Flag for saving/sending to GraceDB!
    - v. ELSE: This alert is not interesting to us or is a test of some kind. Reject this event.
  - C. IF the stream is SNEWS THEN:
    - i. It could be a test, but flag for sending to GraceDB!
5. Now, find all previous VOEvent portfolios cited by this event. Does it cite one we've already captured?
  - A. IF yes THEN flag it for saving on disk and find the portfolio it cites.
  - B. ELSE: Start a new event portfolio in the file cache!
6. Does this event already have an official designation (e.g. GRB 141117A)?
  - A. IF yes THEN make that designation the filename of this Notice on disk (e.g. `GRB141117A.xml`).
  - B. ELSE: Well, that's unfortunate! Because now the filename will be the full event descriptor, which is rather long (e.g. `nasa.gsfc.gcn_Fermi#GBM_Flt_Pos_2014-11-15T06:12:00.03_437724723_58-041.xml`).
7. IF this event has been flagged as an interesting one that ought to be stored on disk, THEN save its VOEvent file (in the appropriate portfolio within the file cache and with the appropriate filename).
8. IF this event has been flagged for sending to GraceDB THEN:
  - A. IF it coincident in time within 1 second of an event that's already been uploaded, THEN send it as an update to that event.
  - B. ELSE: Send it as a new event.

## Unit Tests

For unit testing, see the `grinch` test directory, `~grinch/test/gcn`. A tarball of the directory is at [tar.gz](#). A simple script in this directory called `test_gcn_end2end.py` with the following contents was used to call an instance of the `grinch.gcnhandler.archive()` class:

Toggle line numbers

```
1 import os, sys
2 import logging
```

```

3 from grinch.gcnhandler import archive # Function that filters GCN
notices
4
5 # Pass a VOEvent file as a positional command line argument
6 args = sys.argv[1:]
7 filename = args[0]
8
9 # Set up logger
10 logging.basicConfig(level=logging.INFO, stream=sys.stdout)
11
12 # Get root directory
13 root = os.getcwd()
14
15 # Store the textual content of the VOEvent
16 with open (filename, "r") as f:
17     payload = f.read()
18
19 # Call the event filter
20 archive( payload, root, test=True )

```

This script is then run from the command line as `python test_gcn_end2end.py [voevent.xml]`, provided `grinch`, `VOEventLib`, and `lalsuite` are installed on the local machine.

A simulated VOEvent file (emulating the behavior of real alerts) was then prepared to test each of the alert types LIGO is interested in. In particular, the files

```

[gdb_processor@pcdev2:~/src/grinch/test/gcn] $ ls Swift/*.xml
Swift/Swift_BAT_Alert.xml

```

```

[gdb_processor@pcdev2:~/src/grinch/test/gcn] $ ls -1 Fermi/*.xml
Fermi/Fermi_Fin_Pos.xml
Fermi/Fermi_Flt_Pos.xml
Fermi/Fermi_GBM_Alert.xml

```

test each alert type in turn, while the associated log files

```

[gdb_processor@pcdev2:~/src/grinch/test/gcn] $ ls Swift/*.xml
Swift/Swift_BAT_Alert.out

```

```

[gdb_processor@pcdev2:~/src/grinch/test/gcn] $ ls -1 Fermi/*.xml
Fermi/Fermi_Fin_Pos.out
Fermi/Fermi_Flt_Pos.out
Fermi/Fermi_GBM_Alert.out

```

contain the results of running the above script on each one, organized by gamma-ray observatory. For evidence that the GCN listener passes the unit tests suggested by the review team, please consider the





output of these log files, as well as some purposefully "wrong" VOEvent files (to come).


## Open Issues

---


(  means the action is done;  means it has not yet been met )


 Action #1: get a gracedb.processor account set up with multi-user login the same as gdb\_processor. Migrate services cleanly to this new account for ER7 and future runs.


 Action #2: Update the gcn\_listener pages with the voevent types, info used, etc in a tabular format. Partly done, but a table of variables used inside the code is still needed.


 Action #3: Transition to pygcn, remove the "gcnlisten" script from grinch and remove the "twistd" script from grinch. This will simplify the code dependencies substantially and improve the maintainability of the listener.

 Action #4: (not GRB) add a password file capability to lvalert .... talk to Branson. 

 Action #5: [Ongoing] Provide regular cross-validation of triggers sent by GCN, received by the listener and uploaded to gracedb. See [Google Doc Comparison Sheet](#)

 Action #6: [Ongoing] Begin monitoring of the circulars and reports for updates to GRBs and other gamma-ray triggers. Make sure that information gets captured into GraCEDb.

 Action #7: Patrick to provide some hacked voevents which will lead (hopefully) to graceful failures.

 Action #8: add the logging module into pygcn so that we can easily manage logs, times, rolling, etc.

## Other Important Links

---

- [Webpage](#) of the International Virtual Observatory Alliance (IVOA).
- [Documentation page](#) for the PyGCN package used by the listener.
- [Documentation page](#) of the Gamma-ray Coordination Network (GCN) at the NASA Goddard Space Flight Center.
- [Archive](#) of GCN *circulars*, which are plaintext followups of the Notices from the wider astronomical community. In particular, T90s and X-Ray, optical and radio afterglow observations are all typically reported via GCN circulars.
- [Latest](#) events in GraceDB
- [All Swift events](#) in GraceDB
- [All Fermi events](#) in GraceDB
- [Archival table](#) of all Swift BAT GRBs
- [Archival table](#) of all Fermi LAT GRBs

None: GRBreview/GCNListener (last edited 2015-04-02 06:35:43 by AlexanderUrban)

