GWCelery Documentation

Release 0.8.1

Leo Singer

Jul 29, 2019

Contents:

GWCelery is a simple and reliable package for annotating and orchestrating LIGO/Virgo alerts, built from widely used open source components. It is built on the Celery distributed task queue (hence the name). This is the design and reference manual for GWCelery.

GWCelery's responsibilities include:

1. Merging related candidates from multiple online LIGO/Virgo transient searches into "superevents"
2. Correlating LIGO/Virgo events with gamma-ray bursts, neutrinos, and supernovae
3. Launching automated follow-up analyses including data quality checks, rapid sky localization, automated parameter estimation, and source classification
4. Generating and sending preliminary machine-readable GCN notices
5. Sending updated GCN notices after awaiting human input
6. Automatically composing GCN Circulars

Note: If you are a scientist, student, educator, or astronomy enthusiast looking for information about LIGO/Virgo alerts and low-latency data products, then please see our LIGO/Virgo Public Alerts User Guide.

CHAPTER 1

Quick start

These instructions are suitable for installing GWCelery for development and testing on any machine.

1.1 To install

GWCelery requires Python >= 3.6.

The easiest way to install it is with `venv` and `pip`:

```
$ python -m venv --system-site-packages ~/gwcelery
$ source ~/gwcelery/bin/activate
$ pip install gwcelery
```

Hint: **Note:** GWCelery requires a fairly new version of *setuptools*. If you get an error message that looks like this:

```
pkg_resources.VersionConflict: (setuptools 0.9.8
(/usr/lib/python2.7/site-packages),
Requirement.parse('setuptools>=30.3.0'))
```

then run `pip install --upgrade setuptools` and try again.

1.2 To test

With `setup.py`:

```
$ python setup.py test
```

1.3 To start

Before starting GWCelery, you need to authenticate for access to GraceDB and LVAlert and make sure that you have a Redis server running. Once you have completed those steps, you can start each of the GWCelery manually.

1.3.1 Authentication

To authenticate for GraceDB, obtain grid credentials from the LSC DataGrid Client by running `ligo-proxy-init`:

```
$ ligo-proxy-init albert.einstein
```

To authenticate for LVAlert, first complete the LVAlert Account Activation form once for each LVAlert server that you intend to use (generally you only need "Playground" for development purposes). Make a note of the passwords and store them in your ~/.netrc file with appropriate file permissions:

```
$ echo > ~/.netrc
$ chmod 0600 ~/.netrc
$ echo machine lvalert.cgca.uwm.edu login albert.einstein password password-for-
↪production >> ~/.netrc
$ echo machine lvalert-playground.cgca.uwm.edu login albert.einstein password␣
↪password-for-playground >> ~/.netrc
$ echo machine lvalert-test.cgca.uwm.edu login albert.einstein password password-for-
↪test >> ~/.netrc
```

1.3.2 Redis

GWCelery requires a Redis database server for task bookkeeping. Your operating system's package manager may be able to install, configure, and automatically launch a suitable Redis server for you.

Debian, Ubuntu, `apt`

Debian or Ubuntu users can install and start Redis using `apt-get`:

```
$ sudo apt-get install redis
```

macOS, MacPorts

Mac users with MacPorts can install Redis using `port install`:

```
$ sudo port install redis
```

Use `port load` to start the server:

```
$ sudo port load redis
```

From source

If none of the above options are available, then you can follow the Redis Quick Start instructions to build redis from source and start a server:

```
$ wget http://download.redis.io/redis-stable.tar.gz
$ tar xvzf redis-stable.tar.gz
$ cd redis-stable
$ make -j
$ src/redis-server
```

1.3.3 Start GWCelery components manually

GWCelery itself consists of five Celery workers and one Flask web application. Start them all by running each of the following commands:

```
$ gwcelery worker -l info -n gwcelery-worker -Q celery -B --lvalert
$ gwcelery worker -l info -n gwcelery-exttrig-worker -Q exttrig -c 1
$ gwcelery worker -l info -n gwcelery-openmp-worker -Q openmp -c 1
$ gwcelery worker -l info -n gwcelery-superevent-worker -Q superevent -c 1
$ gwcelery worker -l info -n gwcelery-voevent-worker -Q voevent -P solo
$ gwcelery flask run
```

Hint: With these arguments, each of the commands above will run until you type Control-C. You may want to run each of them in a separate terminal, or in the background using screen or nohup.

CHAPTER 2

Design and anatomy of GWCelery

2.1 Conceptual overview

Several online gravitational-wave transient search pipelines (currently Gstlal, PyCBC, cWB, and oLIB) upload candidates in real time to GraceDB, the central database and web portal for low-latency LIGO/Virgo analyses. Whenever an event is uploaded or altered, GraceDB pushes machine-readable notifications through LVAlert, a pubsub system based on XMPP.

The business logic for selecting and sending alerts to astronomers resides not in GraceDB itself but in GWCelery. The role of GWCelery in the LIGO/Virgo alert infrastructure is to drive the workflow of aggregating and annotating gravitational-wave candidates and sending GCN Notices to astronomers.

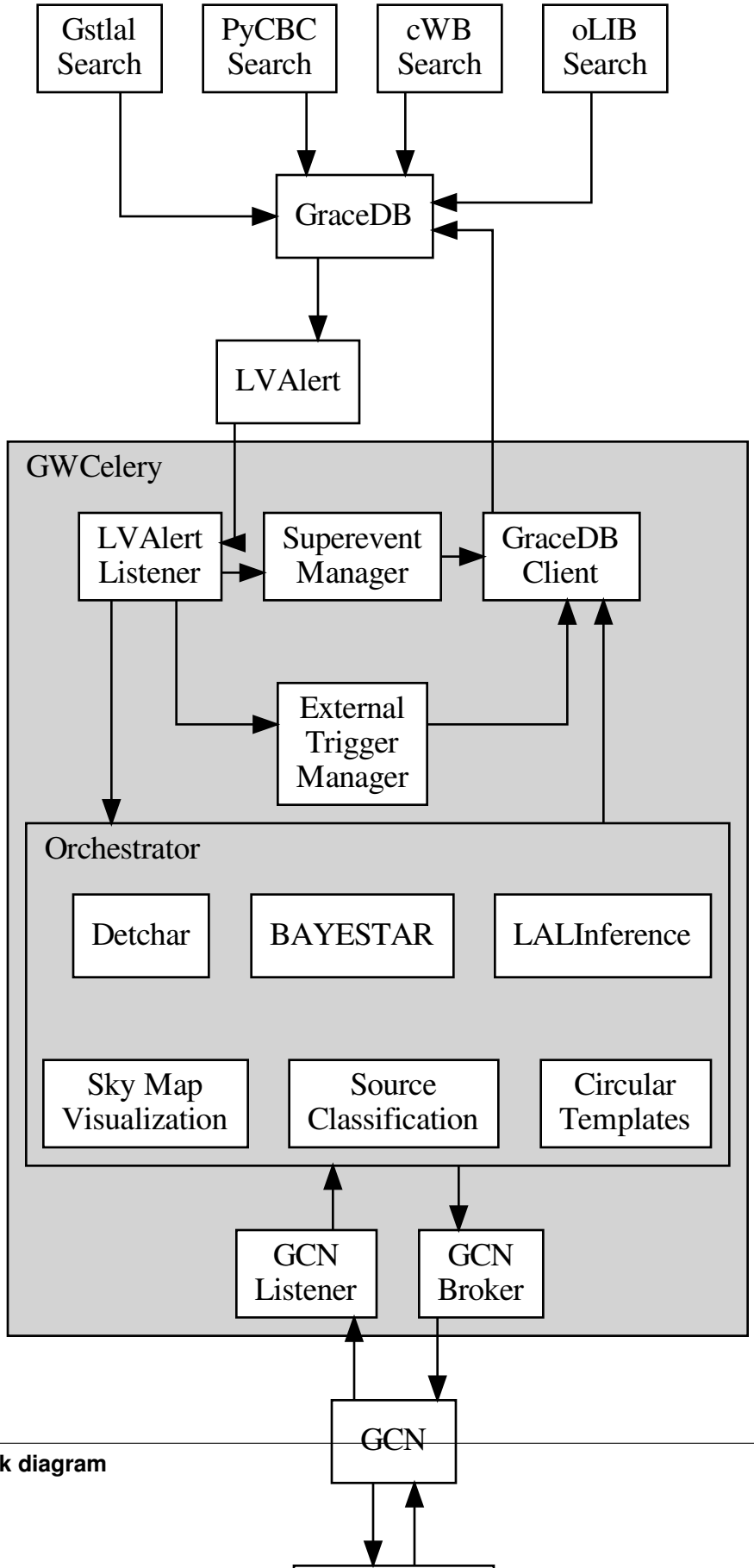
GWCelery interacts with GraceDB by listening for LVAlert messages and making REST API requests through the GraceDB client. GWCelery interacts with GCN by listening for and sending GCN Notices using the Comet VOEvent broker.

The major subsystems of GWCelery are:

- the LVAlert listener
- the GraceDB client
- the GCN listener
- the GCN broker
- the Superevent Manager, which clusters and merges related candidates into "superevents"
- the External Trigger Manager, which correlates gravitational-wave events with GRB, neutrino, and supernova events
- the Orchestrator, which executes the per-event annotation workflow

2.2 Block diagram

Below is a diagram illustrating the conceptual relationships of these subsystems. Nodes in the graph are hyperlinks to the relevant API documentation.


Gstlal Search
PyCBC Search
cWB Search
oLIB Search
GraceDB
LVAlert
GWCelery
LVAlert Listener
Superevent Manager
GraceDB Client
External Trigger Manager
Orchestrator
Detchar
BAYESTAR
LALInference
Sky Map Visualization
Source Classification
Circular Templates
GCN Listener
GCN Broker
GCN

2.3 Processes

A complete deployment of GWCelery (whether launched from the *shell* or from *HTCondor*) consists of several processes:

1. **Message Broker**

   Routes and distributes Celery task messages and stores results of tasks for later retrieval. See Choosing a Broker in the Celery manual for more details. For technical reasons, we use a Redis broker.

2. **Celery Beat**

   Scheduler for periodic tasks (the Celery equivalent of cron jobs). For more information, see Periodic Tasks in the Celery manual.

3. **Monitoring Console** (optional)

   You can optionally run Flower, a web monitoring console for Celery.

4. **OpenMP Worker**

   A Celery worker that has been configured to accept only computationally intensive tasks that use OpenMP parallelism. To route a task to the OpenMP worker, pass the keyword argument `queue='openmp'` to the `@app.task` decorator when you declare it.

   There are two tasks that run in the OpenMP queue:

   - *gwcelery.tasks.bayestar.localize()*
   - *gwcelery.tasks.skymaps.plot_volume()*

5. **Superevent Worker**

   A Celery worker that is dedicated to serially process triggers from low latency pipelines and create/modify superevents in GraceDB. There is only one task that runs on the Superevent queue:

   - *gwcelery.tasks.superevents.handle()*

6. **External Trigger Worker**

   A Celery worker that is dedicated to serially process external triggers from GRB alerts received from Fermi, Swift and neutrino alerts received from SNEWS and create/modify external trigger events in GraceDB:

   - `gwcelery.tasks.external_triggers.handle_gcn()`

7. **VOEvent Worker**

   A Celery worker that is dedicated to sending and receiving VOEvents. It runs an embedded instance of the Comet VOEvent broker, which is started and stopped using a set of custom Celery bootsteps. Note that the VOEvent worker must be started with the `--pool=solo` option so that tasks are executed in the same Python process that is running the VOEvent broker.

8. **General-Purpose Worker**

   A Celery worker that accepts all other tasks.

9. **Flask Web Application**

   A web application that provides forms to manually initiate certain tasks, including sending an update alert or creating a mock event.

2.4 Eternal tasks

GWCelery has a few long-running tasks that do not return because they have to keep open a persistent connection with some external service. These tasks are subclasses of `celery_eternal.EternalTask` or `celery_eternal.EternalProcessTask`.

- `gwcelery.tasks.lvalert.listen()`

These tasks run inside the general-purpose worker process described above, and are automatically started (and restarted as necessary) by Celery Beat.

2.5 Handlers

A recurring pattern in GWCelery is that an eternal task listens continuously to a remote connection, receives packets of data over that connection, and dispatches further handling to other tasks based on packet type.

A decorator is provided to register a function as a Celery task and also plug it in as a handler for one or more packet types. This pattern is used for both GCN notices and LVAlert message handlers.

2.5.1 GCN notices

GCN notice handler tasks are declared using the *gwcelery.tasks.gcn.handler()* decorator:

```
import lxml.etree
from gwcelery.tasks import gcn

@gcn.handler(gcn.NoticeType.FERMI_GBM_GND_POS,
             gcn.NoticeType.FERMI_GBM_FIN_POS)
def handle_fermi(payload):
    root = lxml.etree.fromstring(payload)
    # do work here...
```

2.5.2 LVAlert messages

LVAlert message handler tasks are declared using the *gwcelery.tasks.lvalert.handler()* decorator:

```
from gwcelery.tasks import lvalert

@lvalert.handler('cbc_gstlal',
                 'cbc_spiir',
                 'cbc_pycbc',
                 'cbc_mbtaonline')
def handle_cbc(alert):
    # do work here...
```

CHAPTER 3

Configuration

Like any Celery application, GWCelery's configuration options are stored at run time in a global configuration object, `gwcelery.app.conf`. There are options for Celery itself such as options that affect the task and result backends; these options are documented in the Configuration and defaults section of the Celery manual.

The configuration object also holds all of the options that are specific to GWCelery and affect the behavior of individual GWCelery tasks; examples include the GraceDB and LVAlert service URLs, GCN hostnames, and frame file types and channel names. For a list of all GWCelery-specific options, see the API documentation for the *`gwcelery.conf`* module.

GWCelery provides four preset configurations, one for each GraceDB server instance (production, deployment, testing, or playground). The default configuration preset is for the playground server, `gracedb-playground.ligo.org`. The recommended way to select a different preset is to set the `CELERY_CONFIG_MODULE` environment variable before starting the workers. For example, to configure GWCelery for production:

```
$ export CELERY_CONFIG_MODULE=gwcelery.conf.production
```

3.1 Authentication

There are a few files that must be present in order to provide authentication tokens for GraceDB and LValert.

GraceDB

You must provide valid LSC DataGrid credentials in order for requests to the GraceDB REST API to work. During development and testing, you can use your personal credentials obtained from the LSC DataGrid Client by running `ligo-proxy-init`. However, credentials obtained this way expire after a few days or whenever your machine's temporary directory is wiped (e.g., at system restart).

For production deployment, you should obtain a robot certificate and store it in a location such as `~/.globus/userkey.pem` and `~/.globus/usercert.pem`.

LVAlert

You must provide a valid username and password for LVAlert. You can request an account using the LVAlert Account Activation form. The LVAlert username and password should be stored in your netrc file.

CHAPTER 4

Running under HTCondor

The recommended way to start and stop GWCelery on the LIGO Data Grid cluster is using HTCondor. See the example HTCondor submit file gwcelery.sub. This submit file will start up Redis, the worker processes, the Flask web application, and Flower. It will create some log files and a Unix domain socket, so you should first navigate to a directory where you want these files to go. For example:

```
$ mkdir -p ~/gwcelery/var && cd ~/gwcelery/var
```

Then run the submit file as follows:

```
$ gwcelery.sub
Submitting job(s)........
8 job(s) submitted to cluster 293497.
```

To stop GWCelery, run the `condor_hold` command:

```
$ condor_hold -constraint 'JobBatchName == "gwcelery"'
All jobs matching constraint (JobBatchName == "gwcelery") have been held
```

To restart GWCelery, run `condor_release`:

```
$ condor_release -constraint 'JobBatchName == "gwcelery"'
All jobs matching constraint (JobBatchName == "gwcelery") have been released
```

Note that there is normally **no need** to re-submit GWCelery if the machine is rebooted, because the jobs will persist in the HTCondor queue.

4.1 Shortcuts

The following commands are provided as shortcuts for the above operations:

```
$ gwcelery condor submit
$ gwcelery condor rm
$ gwcelery condor q
$ gwcelery condor hold
$ gwcelery condor release
```

The following command is a shortcut for `gwcelery condor rm; gwcelery condor submit`:

```
$ gwcelery condor resubmit
```

4.2 Managing multiple deployments

There should generally be at most one full deployment of GWCelery per GraceDB server running at one time. The `gwcelery condor` shortcut command is designed to protect you from accidentally starting multiple deployments of GWCelery by inspecting the HTCondor job queue before submitting new jobs. If you try to start GWCelery a second time on the same host in the same directory, you will get the following error message:

```
$ gwcelery condor submit
error: GWCelery jobs are already running in this directory.
You must first remove exist jobs with "gwcelery condor rm".
To see the status of those jobs, run "gwcelery condor q".
```

However, there are situations where you may actually want to run multiple instances of GWCelery on the same machine. For example, you may want to run one instance for the 'production' GraceDB server and one for the 'playground' server. To accomplish this, just start the two instances of gwcelery in different directories. Here is an example:

```
$ mkdir -p production
$ pushd production
$ CELERY_CONFIG_MODULE=gwcelery.conf.production gwcelery condor submit
$ popd
$ mkdir -p playground
$ pushd playground
$ CELERY_CONFIG_MODULE=gwcelery.conf.playground gwcelery condor submit
$ popd
```

CHAPTER 5

Monitoring and Management

GWCelery supports a rich selection of management and monitoring tools. Here is an introduction to a few of them. For more Celery monitoring solutions, see the Celery monitoring and management guide.

5.1 Flower

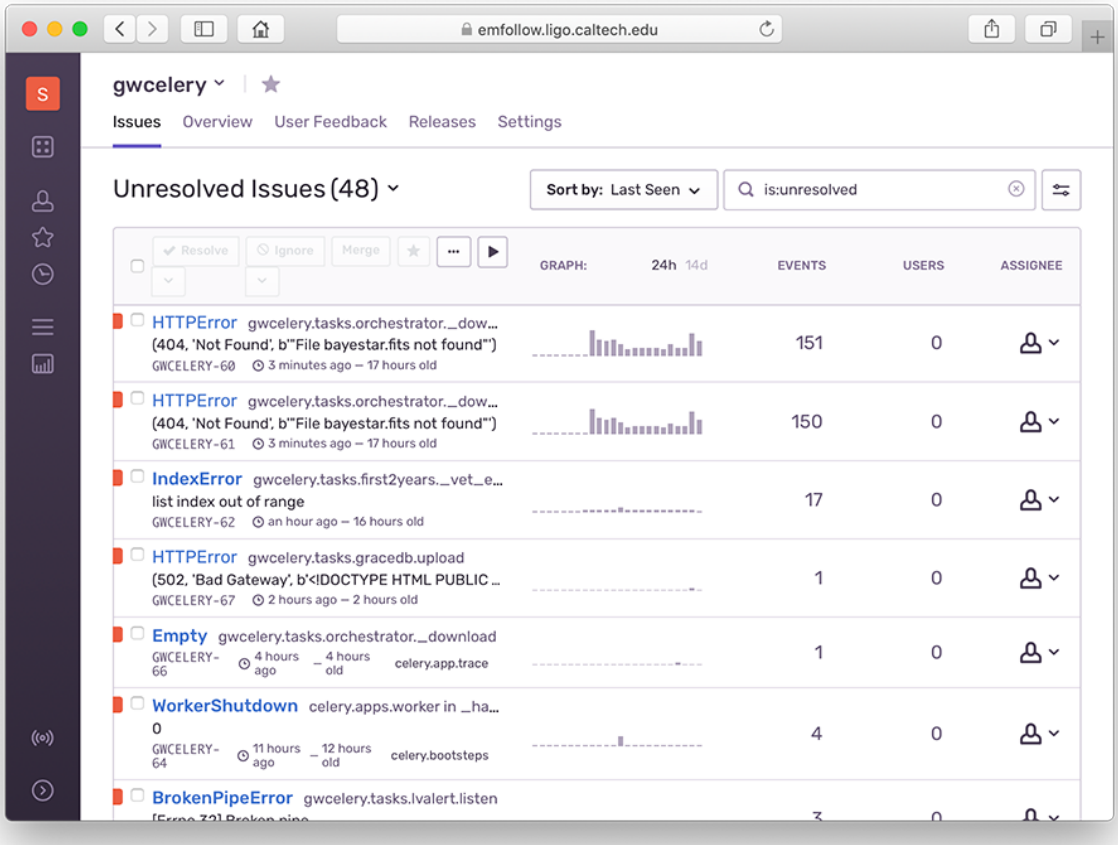
Flower is a dashboard for monitoring Celery tasks. To start Flower for monitoring during local development, run the following command and then navigate to http://localhost:5555/ in your browser:

```
$ gwcelery flower
```

5.2 Sentry

All warnings, errors, exceptions, and tasks failures are both logged to disk and uploaded instantly to Sentry, an error monitoring and reporting platform. The Sentry instance is installed on premise at Caltech. Sentry notifies GWCelery contributors by email when a new bug occurs.

For details about the Sentry logging configuration, see the *gwcelery.sentry* module or the Celery integration module in the Sentry SDK docs.



5.3 Flask

GWCelery includes a *Flask* web application that provides forms to manually initiate certain tasks.

To start Flask for monitoring during local development, run the following command and then navigate to http://localhost:5000/ in your browser:

```
$ gwcelery flask run
```

5.4 Nagios

Note: The GWCelery Nagios plugin is tailored to GWCelery and is not sufficiently general to use with other Celery applications.

The dashboard.ligo.org and monitor.ligo.org services use Nagios to monitor and report on the health of all of the components of the low-latency analysis infrastructure.

GWCelery provides the command `gwcelery nagios` to check the status of the application and provide a report in the format that Nagios expects.

You can run it manually from the command line:

```
$ gwcelery nagios
OK: GWCelery is running normally
```

To configure Nagios itself, see the Nagios configuration overview, or if GWCelery and Nagios are running on different hosts, the Nagios Remote Plugin Executor (NRPE) documentation.

5.5 Command-Line Tools

All Celery application provide command-line monitoring and management utilities, including the following:

- `gwcelery shell`: Start an interactive Python or IPython interpreter for interacting with Celery. All tasks as well as the *app* application instance are automatically imported and available as globals. Example:

```
$ gwcelery shell
Python 3.6.6 (default, Jun 28 2018, 05:43:53)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: download.s('coinc.xml', 'M6757').delay().get()
```

- `gwcelery call`: Call a task from the command line by passing it arguments in JSON format. The output is the unique identifier of the result. Example:

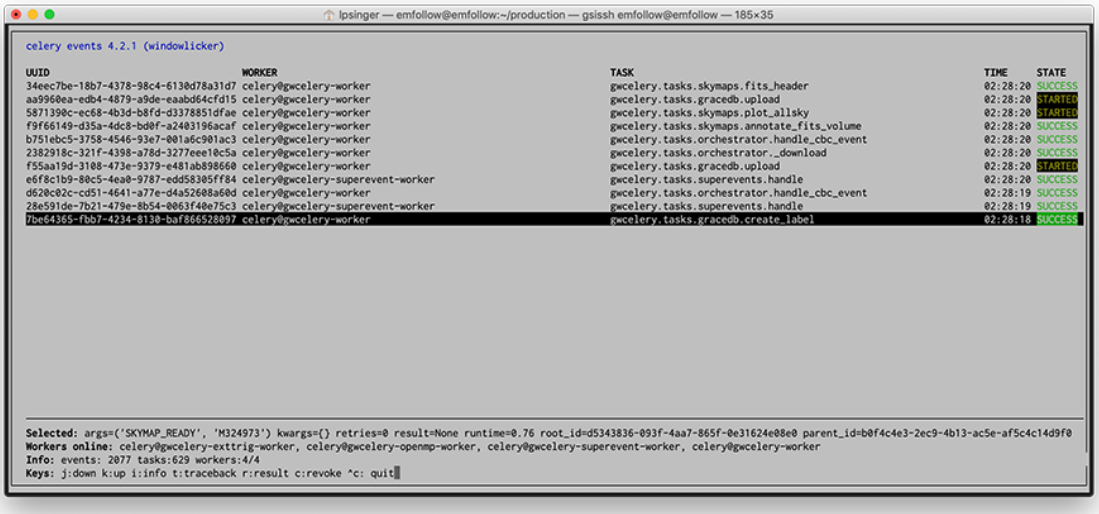
```
$ gwcelery call gwcelery.tasks.gracedb.download --args='["coinc.xml", "M6757"]'
d11099e7-75e5-4aa3-800b-b122b667757c
```

- `gwcelery result`: Get the result of a previously called task. Example:

```
$ gwcelery result ab4aa6d7-9f21-420c-8401-cbe6863cf7dc
(b'<?xml version=\'1.0\' encoding=\'utf-8\'?>\n<!DOCTYPE LIGO_LW SYSTEM "htt'
 b'p://ldas-sw.ligo.caltech.edu/doc/ligolwAPI/html/ligolw_dtd.txt">\n<LIGO_L'
 ...
 b'\t</Stream>\n\t</Table>\n</LIGO_LW>\n')
```

- `gwcelery events`: A text UI monitoring tool that offers functionality similar to Flower. Example:

```
$ gwcelery events
```


lpsinger — emfollow@emfollow:~/production — gsissh emfollow@emfollow — 185×35
celery events 4.2.1 (windowlicker)
UUID WORKER TASK TIME STATE
34eec7be-18b7-4378-98c4-6130d78a31d7 celery@gwcelery-worker gwcelery.tasks.skymaps.fits_header 02:28:20 SUCCESS
aa9960ea-edb4-4879-a9de-eaabd64cfd15 celery@gwcelery-worker gwcelery.tasks.gracedb.upload 02:28:20 STARTED
5871390c-ec68-4b3d-b8fd-d3378851dfae celery@gwcelery-worker gwcelery.tasks.skymaps.plot_allsky 02:28:20 STARTED
f9f66149-d35a-4dc8-bd0f-a2403196acaf celery@gwcelery-worker gwcelery.tasks.skymaps.annotate_fits_volume 02:28:20 SUCCESS
b751ebc5-3758-4546-93e7-001a6c901ac3 celery@gwcelery-worker gwcelery.tasks.orchestrator.handle_cbc_event 02:28:20 SUCCESS
2382918c-321f-4398-a78d-3277eee10c5a celery@gwcelery-worker gwcelery.tasks.orchestrator._download 02:28:20 SUCCESS
f55aa19d-3108-473e-9379-e481ab898660 celery@gwcelery-worker gwcelery.tasks.gracedb.upload 02:28:20 STARTED
e6f8c1b9-80c5-4ea0-9787-edd58305ff84 celery@gwcelery-superevent-worker gwcelery.tasks.superevents.handle 02:28:20 SUCCESS
d620c02c-cd51-4641-a77e-d4a52608a60d celery@gwcelery-worker gwcelery.tasks.orchestrator.handle_cbc_event 02:28:19 SUCCESS
28e591de-7b21-479e-8b54-0063f40e75c3 celery@gwcelery-superevent-worker gwcelery.tasks.superevents.handle 02:28:19 SUCCESS
7be64365-fbb7-4234-8130-baf866528097 celery@gwcelery-worker gwcelery.tasks.gracedb.create_label 02:28:18 SUCCESS
Selected: args=('SKYMAP_READY', 'M324973') kwargs={} retries=0 result=None runtime=0.76 root_id=d5343836-093f-4aa7-865f-0e31624e08e0 parent_id=b0f4c4e3-2ec9-4b13-ac5e-af5c4c14d9f0
Workers online: celery@gwcelery-exttrig-worker, celery@gwcelery-openmp-worker, celery@gwcelery-superevent-worker, celery@gwcelery-worker
Info: events: 2077 tasks:629 workers:4/4
Keys: j:down k:up i:info t:traceback r:result c:revoke ^c: quit

CHAPTER 6

API Reference

Celery application initialization.

gwcelery.**app = <Celery gwcelery>**
Celery application object.

6.1 gwcelery.conf module

GWCelery application configuration.

This module defines configuration variables and default values, including both generic options for Celery as well as options that control the behavior of specific GWCelery *tasks*.

To override the configuration, define the CELERY_CONFIG_MODULE environment variable to the fully qualified name of any Python module that can be located in sys.path, including any of the following presets:

- *gwcelery.conf.development*
- *gwcelery.conf.playground* (the default)
- *gwcelery.conf.production*
- *gwcelery.conf.test*

gwcelery.conf.**expose_to_public = False**
Set to True if events meeting the public alert threshold really should be exposed to the public.

gwcelery.conf.**lvalert_host = 'lvalert-playground.cgca.uwm.edu'**
LVAlert host.

gwcelery.conf.**gracedb_host = 'gracedb-playground.ligo.org'**
GraceDB host.

gwcelery.conf.**voevent_broadcaster_address = ':5342'**
The VOEvent broker will bind to this address to send GCNs. This should be a string of the form *host:port*. If *host* is empty, then listen on all available interfaces.

gwcelery.conf.**voevent_broadcaster_whitelist = []**
List of hosts from which the broker will accept connections. If empty, then completely disable the broker's broadcast capability.

gwcelery.conf.**voevent_receiver_address = ''**
The VOEvent listener will connect to this address to receive GCNs. For options, see GCN's list of available VOEvent servers. If this is an empty string, then completely disable the GCN listener.

gwcelery.conf.**superevent_d_t_start = {'gstlal': 1.0, 'mbtaonline': 1.0, 'pycbc': 1.0, 's**
Pipeline based lower extent of superevent segments. For cwb and lib this is decided from extra attributes.

gwcelery.conf.**superevent_d_t_end = {'gstlal': 1.0, 'mbtaonline': 1.0, 'pycbc': 1.0, 'spi**
Pipeline based upper extent of superevent segments For cwb and lib this is decided from extra attributes.

gwcelery.conf.**superevent_query_d_t_start = 100.0**
Lower extent of superevents query

gwcelery.conf.**superevent_query_d_t_end = 100.0**
Upper extent of superevents query

gwcelery.conf.**superevent_default_d_t_start = 1.0**
Default lower extent of superevent segments

gwcelery.conf.**superevent_default_d_t_end = 1.0**
Default upper extent for superevent segments

gwcelery.conf.**superevent_far_threshold = 0.0002777777777777778**
Maximum false alarm rate to consider events superevents.

gwcelery.conf.**preliminary_alert_far_threshold = {'burst': 3.1709791983764586e-08, 'cbc':**
Group specific maximum false alarm rate to consider sending preliminary alerts.

gwcelery.conf.**preliminary_alert_trials_factor = {'burst': 4.0, 'cbc': 5.0}**
Trials factor corresponding to trigger categories. For CBC and Burst, trials factor is the number of pipelines. CBC pipelines are gstlal, pycbc, mbtaonline, spiir-highmass, spiir-lowmass. Burst searches are cwb.allsky, cwb.bbh, cwb.imbh and olib.allsky.

gwcelery.conf.**orchestrator_timeout = 60.0**
The orchestrator will wait this many seconds from the time of the creation of a new superevent to the time that annotations begin, in order to let the superevent manager's decision on the preferred event stabilize.

gwcelery.conf.**pe_timeout = 105.0**
The orchestrator will wait this many seconds from the time of the creation of a new superevent to the time that parameter estimation begins, in case the preferred event is updated with high latency.

gwcelery.conf.**check_vector_prepost = {'CWB': [0.5, 0.5], 'Fermi': [2, 2], 'HardwareInjecti**
Seconds before and after the superevent start and end times which the DQ vector check will include in its check. Pipeline dependent.

gwcelery.conf.**uses_gatedhoft = {'CWB': True, 'Fermi': False, 'HardwareInjection': False,**
Whether or not a pipeline uses gated h(t). Determines whether or not the DMT-DQ_VECTOR will be analyzed for data quality.

gwcelery.conf.**llhoft_glob = '/dev/shm/kafka/{detector}_O2/*.gwf'**
File glob for playground low-latency h(t) frames. Currently points to O2 replay data.

gwcelery.conf.**llhoft_channels = {'H1:DMT-DQ_VECTOR': 'dmt_dq_vector_bits', 'H1:GDS-CALIB_S**
Low-latency h(t) state vector configuration. This is a dictionary consisting of a channel and its bitmask, as defined in *gwcelery.tasks.detchar*.

gwcelery.conf.**idq_channels = ['H1:IDQ-PGLITCH_OVL_32_2048', 'L1:IDQ-PGLITCH_OVL_32_2048']**
Low-latency iDQ p(glitch) channel names from O2 replay.

gwcelery.conf.**idq_pglitch_thresh = 0.95**
If P(Glitch) is above this threshold, and *idq_veto* for the pipeline is true, DQV will be labeled for the event.

gwcelery.conf.**idq_veto = {'CWB': False, 'Fermi': False, 'HardwareInjection': False, 'LIB**
If true for a pipeline, iDQ values above the threshold defined in `thres` will cause DQV to be labeled. Currently all False, pending iDQ review (should be done before O3).

gwcelery.conf.**p_astro_livetime = 14394240**
sec) corresponding to mean values of Poisson counts. (Used by *gwcelery.tasks.p_astro*)

Type livetime (units

gwcelery.conf.**p_astro_url = 'http://emfollow.ldas.cit/data/H1L1V1-mean_counts-1126051217-6**
URL for mean values of Poisson counts using which p_astro is computed. (Used by *gwcelery.tasks.p_astro*)

gwcelery.conf.**p_astro_thresh_url = 'http://emfollow.ldas.cit/data/H1L1V1-pipeline-far_snr-**
URL for pipeline thresholds on FAR and SNR. (Used by *gwcelery.tasks.p_astro*)

gwcelery.conf.**em_bright_url = 'http://emfollow.ldas.cit/data/em_bright_classifier.pickle'**
URL for trained RandomForestClassifier based on which em_bright classification is conducted. (Used by *gwcelery.tasks.em_bright*)

gwcelery.conf.**low_latency_frame_types = {'H1': 'H1_O2_llhoft', 'L1': 'L1_O2_llhoft', 'V1**
Types of low latency frames used in Parameter Estimation with LALInference (see *gwcelery.tasks.lalinference*) and in cache creation for detchar checks (see *gwcelery.tasks.detchar*).

gwcelery.conf.**high_latency_frame_types = {'H1': None, 'L1': None, 'V1': None}**
Types of high latency frames used in Parameter Estimation with LALInference and in cache creation for detchar checks. They do not exist for O2Replay data. (see *gwcelery.tasks.lalinference* and *gwcelery.tasks.detchar*)

gwcelery.conf.**strain_channel_names = {'H1': 'H1:GDS-CALIB_STRAIN_O2Replay', 'L1': 'L1:GDS**
Names of h(t) channels used in Parameter Estimation with LALInference (see *gwcelery.tasks.lalinference*)

gwcelery.conf.**state_vector_channel_names = {'H1': 'H1:GDS-CALIB_STATE_VECTOR', 'L1': 'L1**
Names of state vector channels used in Parameter Estimation with LALInference (see *gwcelery.tasks.lalinference*)

gwcelery.conf.**pe_threshold = 4.133597883597884e-07**
FAR threshold in Hz for Parameter Estimation. PE group now applies 1/(4 weeks) as a threshold. 86400 seconds = 1 day and 28 days = 4 weeks.

gwcelery.conf.**pe_results_path = '/home/docs/public_html/online_pe'**
Path to the results of Parameter Estimation (see *gwcelery.tasks.lalinference*)

gwcelery.conf.**pe_results_url = 'https://ldas-jobs.ligo.caltech.edu/~docs/online_pe/'**
URL of page where all the results of Parameter Estimation are outputted (see *gwcelery.tasks.lalinference*)

gwcelery.conf.**raven_coincidence_windows = {'GRB_Burst': [-600, 60], 'GRB_CBC': [-5, 1], 'S**
Time coincidence windows passed to ligo-raven. External events and superevents of the appropriate type are considered to be coincident if within time window of each other.

6.1.1 gwcelery.conf.development module

Application configuration for `gracedb-dev1.ligo.org`. Inherits all settings from *gwcelery.conf.test*, with the exceptions below.

gwcelery.conf.development.**gracedb_host = 'gracedb-dev1.ligo.org'**
GraceDB host.

gwcelery.conf.development.**sentry_environment = 'development'**
Record this *environment tag <https://docs.sentry.io/enriching-error-data/environments/>*)‘ in Sentry log messages.

6.1.2 gwcelery.conf.playground module

Application configuration for `gracedb-playground.ligo.org`.

gwcelery.conf.playground.**voevent_receiver_address = '45.58.43.186:8099'**
The VOEvent listener will connect to this address to receive GCNs. For options, see GCN's list of available VOEvent servers. If this is an empty string, then completely disable the GCN listener.

gwcelery.conf.playground.**sentry_environment = 'playground'**
Record this *environment tag <https://docs.sentry.io/enriching-error-data/environments/>*)‘ in Sentry log messages.

6.1.3 gwcelery.conf.production module

Application configuration for `gracedb.ligo.org`. Inherits all settings from *gwcelery.conf.playground*, with the exceptions below.

gwcelery.conf.production.**expose_to_public = True**
Set to True if events meeting the public alert threshold really should be exposed to the public.

gwcelery.conf.production.**lvalert_host = 'lvalert.cgca.uwm.edu'**
LVAlert host.

gwcelery.conf.production.**gracedb_host = 'gracedb.ligo.org'**
GraceDB host.

gwcelery.conf.production.**voevent_broadcaster_address = ':5341'**
The VOEvent broker will bind to this address to send GCNs. This should be a string of the form *host:port*. If *host* is empty, then listen on all available interfaces.

gwcelery.conf.production.**voevent_broadcaster_whitelist = ['capella2.gsfc.nasa.gov']**
List of hosts from which the broker will accept connections. If empty, then completely disable the broker's broadcast capability.

gwcelery.conf.production.**voevent_receiver_address = '68.169.57.253:8099'**
The VOEvent listener will connect to this address to receive GCNs. For options, see GCN's list of available VOEvent servers. If this is an empty string, then completely disable the GCN listener.

gwcelery.conf.production.**llhoft_glob = '/dev/shm/kafka/{detector}/*.gwf'**
File glob for low-latency h(t) frames.

gwcelery.conf.production.**low_latency_frame_types = {'H1': 'H1_llhoft', 'L1': 'L1_llhoft'**
Types of frames used in Parameter Estimation with LALInference (see *gwcelery.tasks.lalinference*) and in cache creation for detchar checks (see *gwcelery.tasks.detchar*).

gwcelery.conf.production.**high_latency_frame_types = {'H1': 'H1_HOFT_C00', 'L1': 'L1_HOFT_**
Types of high latency frames used in Parameter Estimation with LALInference (see *gwcelery.tasks.lalinference*) and in cache creation for detchar checks (see *gwcelery.tasks.detchar*).

gwcelery.conf.production.**idq_channels = ['H1:IDQ-PGLITCH_OVL_16_4096', 'L1:IDQ-PGLITCH_OVL_**
Low-latency iDQ p(glitch) channel names from live O3 frames

gwcelery.conf.production.**strain_channel_names = {'H1': 'H1:GDS-CALIB_STRAIN_CLEAN', 'L1':**
Names of h(t) channels used in Parameter Estimation with LALInference (see *gwcelery.tasks.lalinference*)

gwcelery.conf.production.**sentry_environment = 'production'**
Record this *environment tag <https://docs.sentry.io/enriching-error-data/environments/>*)‘ in Sentry log messages.

6.1.4 gwcelery.conf.test module

Application configuration for `gracedb-test.ligo.org`. Inherits all settings from *gwcelery.conf.playground*, with the exceptions below.

gwcelery.conf.test.**lvalert_host = 'lvalert-test.cgca.uwm.edu'**
LVAlert host.

gwcelery.conf.test.**gracedb_host = 'gracedb-test.ligo.org'**
GraceDB host.

gwcelery.conf.test.**sentry_environment = 'test'**
Record this *environment tag <https://docs.sentry.io/enriching-error-data/environments/>*)‘ in Sentry log messages.

6.2 gwcelery.lvalert module

Embed a Comet LVAlert listener into a Celery worker by extending Celery with bootsteps.

gwcelery.lvalert.**add_worker_arguments**(*parser*)

gwcelery.lvalert.**install**(*app*)
Register the LVAlert subsystem in the application boot steps.

6.2.1 gwcelery.lvalert.bootsteps module

class gwcelery.lvalert.bootsteps.**Receiver**(*consumer*, *lvalert=False*, ***kwargs*)
Bases: gwcelery.lvalert.bootsteps.LVAlertBootStep

Run the global LVAlert receiver in background thread.

name = 'LVAlert client'

create(*consumer*)
Create the step.

start(*consumer*)

stop(*consumer*)

info(*consumer*)

6.2.2 gwcelery.lvalert.signals module

Definitions of custom Celery signals related to VOEvents.

These signals allow us to keep the VOEvent broker code decoupled from any GCN-specific logic. Notably, it allows us to keep all of the details of the GCN-specific “Notice Type” concept out of *gwcelery.voevent*.

gwcelery.lvalert.signals.**lvalert_received = <Signal: lvalert_received providing_args={'noc**
Fired whenever a VOEvent is received.

Parameters **xml_document** (comet.utility.xml.xml_document) – The XML document that was received. The raw file contents are available as xml_document.raw_bytes. The lxml.etree representation of the document is available as xml_document.element.

6.3 gwcelery.sentry module

Integration of the Celery logging system with Sentry.

gwcelery.sentry.**DSN = 'https://sentry.io/1425216'**
Sentry data source name (DSN).

gwcelery.sentry.**configure**()
Configure Sentry logging integration for Celery according to the official instructions.

Add the API key username/pasword pair to your netrc file.

6.4 gwcelery.tasks module

All Celery tasks are declared in submodules of this module.

6.4.1 gwcelery.tasks.bayestar module

Rapid sky localization with BAYESTAR.

(task) gwcelery.tasks.bayestar.**localize**(*coinc_psd*, *graceid*, *filename='bayestar.fits.gz'*, *disabled_detectors=None*)
Generate a rapid sky localization using BAYESTAR.

Parameters

- **coinc_psd** (*tuple*) – Tuple consisting of the byte contents of the input event's coinc.xml and psd.xml.gz files.
- **graceid** (*str*) – The GraceDB ID, used for FITS metadata and recording log messages to GraceDB.
- **filename** (*str, optional*) – The name of the FITS file.
- **disabled_detectors** (*list, optional*) – List of detectors to disable.

Returns The byte contents of the finished FITS file.

Return type bytes

Notes

This task is adapted from the command-line tool bayestar-localize-lvalert.

It should execute in a special queue for computationally intensive, multithreaded, OpenMP tasks.

6.4.2 gwcelery.tasks.circulars module

Generate and upload automated circulars.

(task) gwcelery.tasks.circulars.**create_initial_circular**(*graceid*)
Create and return circular txt.

(task) gwcelery.tasks.circulars.**create_emcoinc_circular**(*graceid*)
Create and return the em_coinc circular txt.

(task) gwcelery.tasks.circulars.**create_retraction_circular**(*graceid*)
Create and return retraction circular txt.

6.4.3 gwcelery.tasks.condor module

Submit and monitor HTCondor jobs[1].

Notes

Internally, we use the XML condor log format[2] for easier parsing.

References

exception gwcelery.tasks.condor.**JobAborted**
Bases: Exception

Raised if an HTCondor job was aborted (e.g. by condor_rm).

exception gwcelery.tasks.condor.**JobRunning**
Bases: Exception

Raised if an HTCondor job is still running.

exception gwcelery.tasks.condor.**JobFailed**(*returncode*, *cmd*, *output=None*, *stderr=None*)
Bases: subprocess.CalledProcessError

Raised if an HTCondor job fails.

(task) gwcelery.tasks.condor.**submit**(*submit_file*, *log=None*)
Submit a job using HTCondor.

Parameters

- **submit_file** (*str*) – Path of the submit file.
- **log** (*str*) – Used internally to track job state. Caller should not set.

Raises

- *JobAborted* – If the job was aborted (e.g. by running condor_rm).
- *JobFailed* – If the job terminates and returns a nonzero exit code.
- *JobRunning* – If the job is still running. Causes the task to be re-queued until the job is complete.

[1] http://research.cs.wisc.edu/htcondor/manual/latest/condor_submit.html
[2] http://research.cs.wisc.edu/htcondor/classad/refman/node3.html

Example

```
>>> submit.s('example.sub',
...          accounting_group='ligo.dev.o3.cbc.explore.test')
```

(task) gwcelery.tasks.condor.**check_output**(*args*, *log=None*, *error=None*, *output=None*, ***kwargs*)

Call a process using HTCondor.

Call an external process using HTCondor, in a manner patterned after `subprocess.check_output()`. If successful, returns its output on stdout. On failure, raise an exception.

Parameters

- **args** (*list*) – Command line arguments, as if passed to `subprocess.check_call()`.
- **error, output** (*log,*) – Used internally to track job state. Caller should not set.
- ****kwargs** – Extra submit description file commands. See the documentation for `condor_submit` for possible values.

Returns Captured output from command.

Return type str

Raises

- *JobAborted* – If the job was aborted (e.g. by running `condor_rm`).
- *JobFailed* – If the job terminates and returns a nonzero exit code.
- *JobRunning* – If the job is still running. Causes the task to be re-queued until the job is complete.

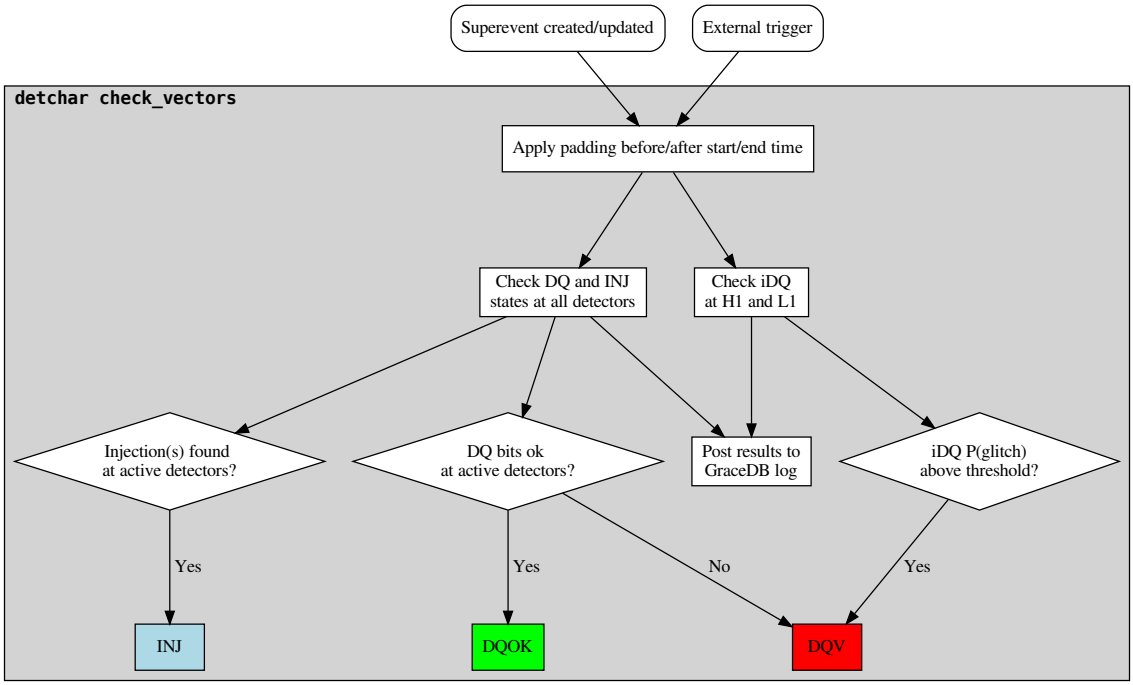
Example

```
>>> check_output.s(['sleep', '10'],
...                accounting_group='ligo.dev.o3.cbc.explore.test')
```

6.4.4 gwcelery.tasks.detchar module

Flow Chart

The flow chart below shows the decision process for the application of DQOK and DQV labels.



Data quality and detector characterization tasks.

These tasks are mostly focused on checking interferometer state vectors. By design, the [LIGO] and [Virgo] state vectors share the same definitions for the first 8 fields.

LIGO also has a [DMT] DQ vector that provides some additional instrumental checks.

References

`gwcelery.tasks.detchar.`**`dmt_dq_vector_bits`**
DMT DQ vector bits (LIGO only).

`gwcelery.tasks.detchar.`**`ligo_state_vector_bits`**
State vector bitfield definitions for LIGO.

`gwcelery.tasks.detchar.`**`virgo_state_vector_bits`**
State vector bitfield definitions for Virgo.

`gwcelery.tasks.detchar.`**`create_cache`**(*ifo*, *start*, *end*)
Find .gwf files and create cache. Will first look in the llhoft, and if the frames have expired from llhoft, will call gwdatafind.

Parameters

- **`ifo`** (*`str`*) – Interferometer name (e.g. `H1`).
- **`end`** (*`start,`*) – GPS start and end times desired.

Returns

Return type `glue.lal.Cache`

Example

```
>>> create_cache('H1', 1198800018, 1198800618)
[<glue.lal.CacheEntry at 0x7fbae6b71278>,
  <glue.lal.CacheEntry at 0x7fbae6ae5b38>,
  <glue.lal.CacheEntry at 0x7fbae6ae5c50>,
 ...
  <glue.lal.CacheEntry at 0x7fbae6b15080>,
  <glue.lal.CacheEntry at 0x7fbae6b15828>]
```

gwcelery.tasks.detchar.**generate_table**(*title*, *high_bit_list*, *low_bit_list*, *unknown_bit_list*)

Make a nice table which shows the status of the bits checked.

Parameters

- **title** (*str*) – Title of the table.
- **high_bit_list** (*list*) – List of bit names which are high.
- **low_bit_list** (*list*) – List of bit names which are low.
- **unknown_bit_list** (*list*) – List of bit names which are unknown.

Returns HTML string of the table.

Return type str

gwcelery.tasks.detchar.**dqr_json**(*state*, *summary*)

Generate DQR-compatible json-ready dictionary from process results, as described in `data-quality-report.design`.

Parameters

- **state** (*{'pass', 'fail'}*) – State of the detchar checks.
- **summary** (*str*) – Summary of results from the process.

Returns Ready to be converted into json.

Return type dict

gwcelery.tasks.detchar.**check_idq**(*cache*, *channel*, *start*, *end*)

Looks for iDQ frame and reads them.

Parameters

- **cache** (`glue.lal.Cache`) – Cache from which to check.
- **channel** (*str*) – which idq channel (pglitch)
- **end** (*start,*) – GPS start and end times desired.

Returns Tuple mapping iDQ channel to its maximum P(glitch).

Return type tuple

Example

```
>>> check_idq(cache, 'H1:IDQ-PGLITCH-OVL-100-1000',
              1216496260, 1216496262)
('H1:IDQ-PGLITCH-OVL-100-1000', 0.87)
```

gwcelery.tasks.detchar.**check_vector**(*cache*, *channel*, *start*, *end*, *bits*, *logic_type='all'*)

Check timeseries of decimals against a bitmask. This is inclusive of the start time and exclusive of the end time, i.e. [start, . . . , end).

Parameters

- **cache** (glue.lal.Cache) – Cache from which to check.
- **channel** (*str*) – Channel to look at, e.g. H1:DMT-DQ_VECTOR.
- **end** (*start,*) – GPS start and end times desired.
- **bits** (gwpy.TimeSeries.Bits) – Definitions of the bits in the channel.
- **logic_type** (*str, optional*) – Type of logic to apply for vetoing. If all, then all samples in the window must pass the bitmask. If any, then one or more samples in the window must pass.

Returns Maps each bit in channel to its state.

Return type dict

Example

```
>>> check_vector(cache, 'H1:GDS-CALIB_STATE_VECTOR', 1216496260,
                 1216496262, ligo_state_vector_bits)
{'H1:HOFT_OK': True,
 'H1:OBSERVATION_INTENT': True,
 'H1:NO_STOCH_HW_INJ': True,
 'H1:NO_CBC_HW_INJ': True,
 'H1:NO_BURST_HW_INJ': True,
 'H1:NO_DETCHAR_HW_INJ': True}
```

(task) gwcelery.tasks.detchar.**check_vectors**(*event*, *graceid*, *start*, *end*)

Perform data quality checks for an event and labels/logs results to GraceDB.

Depending on the pipeline, a certain amount of time (specified in *check_vector_prepost*) is appended to either side of the superevent start and end time. This is to catch DQ issues slightly before and after the event, such as that appearing in L1 just before GW170817.

A cache is then created for H1, L1, and V1, regardless of the detectors involved in the event. Then, the bits and channels specified in the configuration file (*llhoft_channels*) are checked. If an injection is found in the active detectors, 'INJ' is labeled to GraceDB. If an injection is found in any detector, a message with the injection found is logged to GraceDB. If no injections are found across all detectors, this is logged to GraceDB.

A similar task is performed for the DQ states described in the DMT-DQ_VECTOR, LIGO GDS-CALIB_STATE_VECTOR, and Virgo DQ_ANALYSIS_STATE_VECTOR. If no DQ issues are found in active detectors, 'DQOK' is labeled to GraceDB. Otherwise, 'DQV' is labeled. In all cases, the DQ states of all the state vectors checked are logged to GraceDB.

This skips MDC events.

Parameters

- **event** (*dict*) – Details of event.
- **graceid** (*str*) – GraceID of event to which to log.
- **end** (*start,*) – GPS start and end times desired.

Returns **event** – Details of the event, reflecting any labels that were added.

Return type dict

6.4.5 gwcelery.tasks.em_bright module

Qualitative source classification for CBC events.

(task) gwcelery.tasks.em_bright.**classifier_other**(*args*, *graceid*)

Returns the boolean probability of having a NS component and the probability of having non-zero disk mass. This method is used for pipelines that do not provide the data products necessary for computation of the source properties probabilities.

Parameters

- **args** (*tuple*) – Tuple containing (m1, m2, spin1z, spin2z, snr)
- **graceid** (*str*) – The graceid of the event

Returns JSON formatted string storing `HasNS` and `HasRemnant` probabilities

Return type str

Example

```
>>> em_bright.classifier_other((2.0, 1.0, 0.0, 0.0, 10.), 'S123456')
'{"HasNS": 1.0, "HasRemnant": 1.0}'
```

(task) gwcelery.tasks.em_bright.**classifier_gstlal**(*args*, *graceid*)

Returns the probability of having a NS component and the probability of having non-zero disk mass in the detected event. This method will be using the data products obtained from the weekly supervised learning runs for injections campaigns. The data products are in pickle formatted RandomForestClassifier objects. The method predict_proba of these objects provides us the probabilities of the coalesence being EM-Bright and existence of neutron star in the binary.

Parameters

- **args** (*tuple*) – Tuple containing (m1, m2, spin1z, spin2z, snr)
- **graceid** (*str*) – The graceid of the event

Returns JSON formatted string storing `HasNS` and `HasRemnant` probabilities

Return type str

Notes

This task would only work from within the CIT cluster.

6.4.6 gwcelery.tasks.external_triggers module

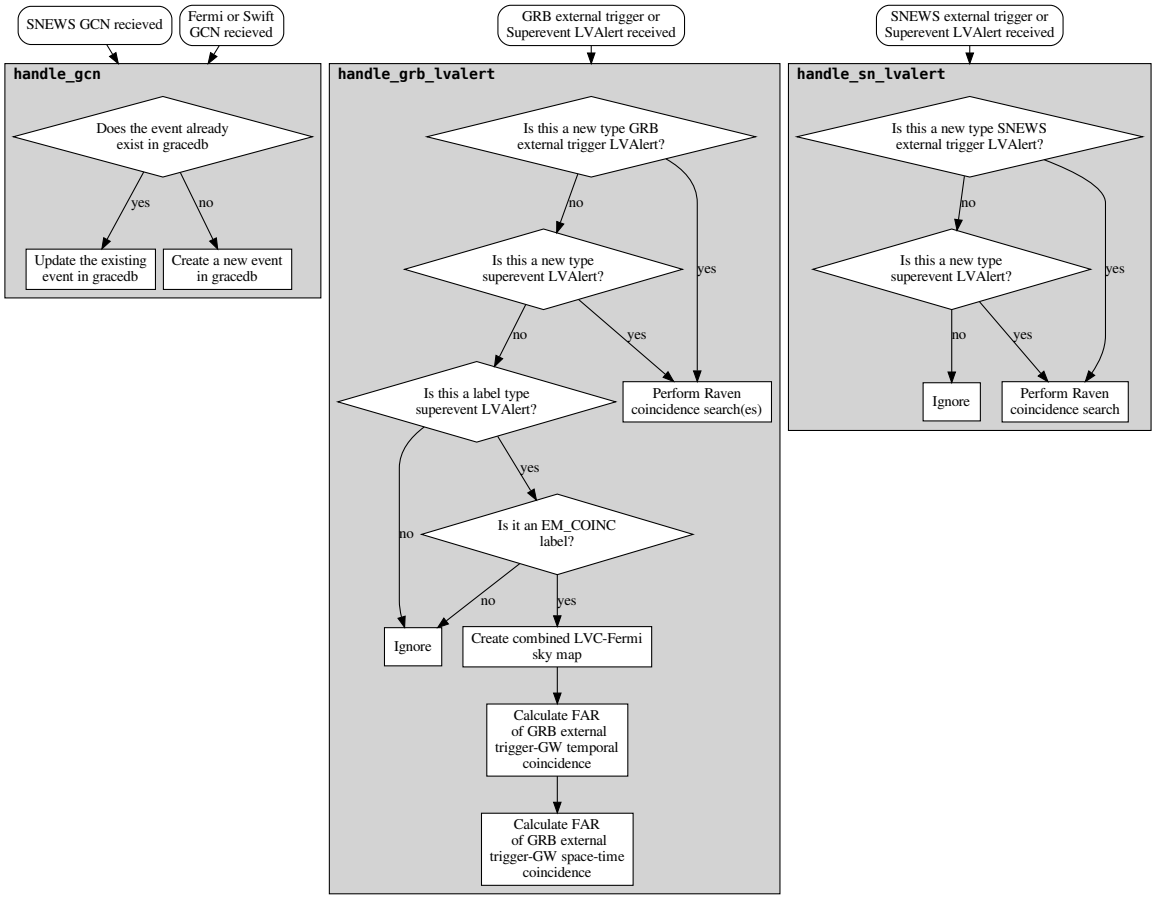
This module listens to the *GCNs* from SNEWS and the Fermi and Swift missions. It is also responsible for carrying out tasks related to external trigger-gravitational wave coincidences, including looking for temporal coincidences, creating combined GRB-GW sky localization probability maps, and computing their joint temporal and spatio-temporal false alarm rates.

There are two GCN and two LVAlert message handlers in the *~gwcelery.tasks.external_triggers* module:

- `handle_sn_gcn()` is called for each SNEWS GCN.
- *handle_grb_gcn()* is called for each Fermi and Swift GCN.
- `handle_sn_lvalert()` is called for each SNEWS external trigger and superevent LVAlert.

- *handle_grb_lvalert()* is called for each Fermi and Swift external trigger and superevent LVAlert.

Flow Chart



Tasks

(task) gwcelery.tasks.external_triggers.**handle_snews_gcn**(*payload*)
Handles the payload from SNEWS alerts. Prepares the alert to be sent to graceDB as 'E' events.

(task) gwcelery.tasks.external_triggers.**handle_grb_gcn**(*payload*)
Handles the payload from Fermi and Swift alerts. Prepares the alert to be sent to graceDB as 'E' events.

(task) gwcelery.tasks.external_triggers.**handle_grb_lvalert**(*alert*)
Parse an LVAlert message related to superevents/GRB external triggers and dispatch it to other tasks.

Notes

This LVAlert message handler is triggered by creating a new superevent or GRB external trigger event, or applying the EM_COINC label to any superevent:

- Any new event triggers a coincidence search with *gwcelery.tasks.raven.coincidence_search()*.
- The EM_COINC label triggers the creation of a combined GW-GRB sky map using *gwcelery.tasks.ligo_fermi_skymaps.create_combined_skymap()*.

(task) gwcelery.tasks.external_triggers.**handle_snews_lvalert**(*alert*)
Parse an LVAlert message related to superevents/SN external triggers and dispatch it to other tasks.

Notes

This LVAlert message handler is triggered by creating a new superevent or SN external trigger event, or applying the EM_COINC label to any superevent:

- Any new event triggers a coincidence search with *gwcelery.tasks.raven.coincidence_search()*.

(task) gwcelery.tasks.external_triggers.**handle_emcoinc_lvalert**(*alert*)
Parse an LVAlert message related to 'coincidence_far.json' file upload and then upload circular. We need a separate handler to prevent doubles from occurring by adding the task to both the handle_snews_lvalert and handle_grb_lvalert handlers.

Notes

This LVAlert message handler is triggered by uploading 'coincidence_far.json' file to any superevent:

- Any 'coincidence_far.json' file upload triggers *gwcelery.tasks.circulars.create_emcoinc_circular()*.

6.4.7 gwcelery.tasks.first2years module

Create mock events from the "First Two Years" paper.

(task) gwcelery.tasks.first2years.**pick_coinc**
Pick a coincidence from the "First Two Years" paper.

(task) gwcelery.tasks.first2years.**_vet_event**(*superevents*)

(task) gwcelery.tasks.first2years.**upload_event**
Upload a random event from the "First Two Years" paper.

After 2 minutes, randomly either retract or confirm the event to send a retraction or initial notice respectively.

6.4.8 gwcelery.tasks.ligo_fermi_skymaps module

Create and upload LVC-Fermi sky maps.

gwcelery.tasks.ligo_fermi_skymaps.**create_combined_skymap**(*graceid*)
Creates and uploads the combined LVC-Fermi skymap. This also uploads the external trigger skymap to the external trigger GraceDB page.

(task) gwcelery.tasks.ligo_fermi_skymaps.**get_preferred_skymap**(*graceid*)
Get the LVC skymap fits filename. If not available, will try again 10 seconds later, then 20, then 40, etc. until up to 10 minutes after initial attempt.

(task) gwcelery.tasks.ligo_fermi_skymaps.**combine_skymaps**(*skymap1filebytes*, *skymap2filebytes*)
This task combines the two input skymaps, in this case the external trigger skymap and the LVC skymap and writes to a temporary output file. It then returns the contents of the file as a byte array.

(task) gwcelery.tasks.ligo_fermi_skymaps.**external_trigger**(*graceid*)
Returns the associated external trigger GraceDB ID.

(task) gwcelery.tasks.ligo_fermi_skymaps.**external_trigger_heasarc**(*external_id*)
Returns the HEASARC fits file link

(task) gwcelery.tasks.ligo_fermi_skymaps.**get_external_skymap**(*heasarc_link*)
Download the Fermi sky map fits file and return the contents as a byte array. If not available, will try again 10 seconds later, then 20, then 40, etc. until up to 10 minutes after initial attempt.

6.4.9 gwcelery.tasks.gcn module

Tasks for sending, receiving, and processing Gamma-ray Coordinates Network [GCN] notices.

References

gwcelery.tasks.gcn.**handler = {<sphinx.ext.autodoc.importer._MockObject object>: [<@task:**
Function decorator to register a handler callback for specified GCN notice types. The decorated function is turned into a Celery task, which will be automatically called whenever a matching GCN notice is received.

Parameters

- ***keys** – List of GCN notice types to accept
- ****kwargs** – Additional keyword arguments for celery.Celery.task().

Examples

Declare a new handler like this:

```
@gcn.handler(gcn.NoticeType.FERMI_GBM_GND_POS,
             gcn.NoticeType.FERMI_GBM_FIN_POS)
def handle_fermi(payload):
    root = lxml.etree.fromstring(payload)
    # do work here...
```

exception gwcelery.tasks.gcn.**SendingError**
Bases: RuntimeError

A generic error associated with sending VOEvents.

(task) gwcelery.tasks.gcn.**send**(*message*)
Send a VOEvent to GCN.

This task will be retried several times if the VOEvent cannot be sent. See the Raises section below for circumstances that cause a retry.

Parameters **message** (*bytes*) – The raw VOEvent file contents.

Raises *SendingError* – If the VOEvent could not be sent because there were no network peers connected to the VOEvent broadcaster.

(task) gwcelery.tasks.gcn.**validate**(*payload*)
Check that the contents of a public LIGO/Virgo GCN matches the original VOEvent in GraceDB.

Notes

If the VOEvent broadcaster is disabled by setting *voevent_broadcaster_whitelist* to an empty list, then this task becomes a no-op.

6.4.10 gwcelery.tasks.gracedb module

Communication with GraceDB.

class gwcelery.tasks.gracedb.**RetryableHTTPError**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Exception class for server-side HTTP errors that we should retry.

gwcelery.tasks.gracedb.**catch_retryable_http_errors**(*f*)
Decorator to capture server-side errors that we should retry.

We retry HTTP status 502 (Bad Gateway), 503 (Service Unavailable), and 504 (Gateway Timeout).

gwcelery.tasks.gracedb.**task**(**args*, ***kwargs*)

(task) gwcelery.tasks.gracedb.**create_event**(**args*, ***kwargs*)
Create an event in GraceDB.

(task) gwcelery.tasks.gracedb.**create_label**(**args*, ***kwargs*)
Create a label in GraceDB.

(task) gwcelery.tasks.gracedb.**remove_label**(**args*, ***kwargs*)
Create a label in GraceDB.

(task) gwcelery.tasks.gracedb.**create_signoff**(**args*, ***kwargs*)
Create a label in GraceDB.

(task) gwcelery.tasks.gracedb.**create_tag**(**args*, ***kwargs*)
Create a tag in GraceDB.

(task) gwcelery.tasks.gracedb.**create_voevent**(**args*, ***kwargs*)
Create a VOEvent.

Returns The filename of the new VOEvent.

Return type str

(task) gwcelery.tasks.gracedb.**download**(**args*, ***kwargs*)
Download a file from GraceDB.

(task) gwcelery.tasks.gracedb.**expose**(**args*, ***kwargs*)
Expose an event to the public.

Notes

If *expose_to_public* is False, then this because a no-op.

(task) gwcelery.tasks.gracedb.**get_events**(**args*, ***kwargs*)
Get events from GraceDB.

(task) gwcelery.tasks.gracedb.**get_event**(**args*, ***kwargs*)
Retrieve an event from GraceDB.

(task) gwcelery.tasks.gracedb.**get_labels**(**args*, ***kwargs*)
Get all labels for an event in GraceDB.

(task) gwcelery.tasks.gracedb.**get_log**(**args*, ***kwargs*)
Get all log messages for an event in GraceDB.

(task) gwcelery.tasks.gracedb.**get_superevent**(**args*, ***kwargs*)
Retrieve a superevent from GraceDB.

(task) gwcelery.tasks.gracedb.**replace_event**(**args*, ***kwargs*)
Get an event from GraceDB.

(task) gwcelery.tasks.gracedb.**upload**(**args*, ***kwargs*)
Upload a file to GraceDB.

(task) gwcelery.tasks.gracedb.**get_superevents**(**args*, ***kwargs*)
List matching superevents in gracedb.

Parameters

- ***args** – arguments passed to `GraceDb.superevents()`
- ****kwargs** – keyword arguments passed to `GraceDb.superevents()`

Returns **superevents** – The list of the superevents.

Return type list

(task) gwcelery.tasks.gracedb.**update_superevent**(**args*, ***kwargs*)
Update superevent information. Wrapper around `updateSuperevent()`

Parameters

- **superevent_id** (*str*) – superevent uid
- **t_start** (*float*) – start of superevent time window, unchanged if None
- **t_end** (*float*) – end of superevent time window, unchanged if None
- **t_0** (*float*) – superevent t_0, unchanged if None
- **preferred_event** (*str*) – uid of the preferred event, unchanged if None

(task) gwcelery.tasks.gracedb.**create_superevent**(**args*, ***kwargs*)
Create new superevent in GraceDB with *graceid*

Parameters

- **graceid** (*str*) – graceid with which superevent is created.
- **t0** (*float*) – `t_0` parameter of superevent
- **t_start** (*float*) – `t_start` parameter of superevent
- **t_end** (*float*) – `t_end` parameter of superevent
- **category** (*str*) – superevent category

(task) gwcelery.tasks.gracedb.**add_event_to_superevent**(**args*, ***kwargs*)
Add an event to a superevent in GraceDB.

6.4.11 gwcelery.tasks.lalinference module

Source Parameter Estimation with LALInference.

exception gwcelery.tasks.lalinference.**NotEnoughData**
Bases: `Exception`

Raised if found data is not enough due to the latency of data transfer

(task) gwcelery.tasks.lalinference.**query_data**(*trigtime*)
Continues to query data until it is found with gwdatafind and return frametypes for the data. If data is not found in 86400 seconds = 1 day, raise NotEnoughData.

(task) gwcelery.tasks.lalinference.**upload_no_frame_files**(*request*, *exc*, *traceback*, *superevent_id*)
Upload notification when no frame files are found.

Parameters

- **request** (*Context (placeholder)*) – Task request variables
- **exc** (*Exception*) – Exception rased by condor.submit
- **traceback** (*str (placeholder)*) – Traceback message from a task
- **superevent_id** (*str*) – The GraceDB ID of a target superevent

(task) gwcelery.tasks.lalinference.**prepare_ini**(*frametype_dict*, *event*, *superevent_id=None*)
Determine an appropriate PE settings for the target event and return ini file content

gwcelery.tasks.lalinference.**pre_pe_tasks**(*event*, *superevent_id*)
Return canvas of tasks executed before parameter estimation starts

(task) gwcelery.tasks.lalinference.**dag_prepare**(*coinc_psd*, *ini_contents*, *rundir*, *superevent_id*)
Create a Condor DAG to run LALInference on a given event.

Parameters

- **coinc_psd** (*tuple*) – The tuple of the byte contents of `coinc.xml` and `psd.xml.gz`
- **ini_contents** (*str*) – The content of online_pe.ini
- **rundir** (*str*) – The path to a run directory where the DAG file exits
- **superevent_id** (*str*) – The GraceDB ID of a target superevent

Returns **submit_file** – The path to the .sub file

Return type str

(task) gwcelery.tasks.lalinference.**job_error_notification**(*request*, *exc*, *traceback*, *superevent_id*, *rundir*)
Upload notification when condor.submit terminates unexpectedly.

Parameters

- **request** (*Context (placeholder)*) – Task request variables
- **exc** (*Exception*) – Exception rased by condor.submit
- **traceback** (*str (placeholder)*) – Traceback message from a task
- **superevent_id** (*str*) – The GraceDB ID of a target superevent
- **rundir** (*str*) – The run directory for PE

(task) gwcelery.tasks.lalinference.**_upload_url**(*pe_results_path*, *graceid*)
Upload url of a page containing all of the plots.

(task) gwcelery.tasks.lalinference.**_get_result_contents**(*pe_results_path*, *filename*)
Return the contents of a PE results file by reading it from the local filesystem.

(task) gwcelery.tasks.lalinference.**clean_up**(*rundir*)
Clean up a run directory.

Parameters **rundir** (*str*) – The path to a run directory where the DAG file exits

(task) gwcelery.tasks.lalinference.**dag_finished**(*rundir*, *preferred_event_id*, *superevent_id*)

Upload PE results and clean up run directory

Parameters

- **rundir** (*str*) – The path to a run directory where the DAG file exits
- **preferred_event_id** (*str*) – The GraceDB ID of a target preferred event
- **superevent_id** (*str*) – The GraceDB ID of a target superevent

Returns **tasks** – The work-flow for uploading PE results

Return type canvas

(task) gwcelery.tasks.lalinference.**_download_psd**(*gid*)

Download `psd.xml.gz` and return its content. If that file does not exist, return None.

(task) gwcelery.tasks.lalinference.**start_pe**(*ini_contents*, *preferred_event_id*, *superevent_id*)

Run LALInference on a given event.

Parameters

- **ini_contents** (*str*) – The content of online_pe.ini
- **preferred_event_id** (*str*) – The GraceDB ID of a target preferred event
- **superevent_id** (*str*) – The GraceDB ID of a target superevent

6.4.12 gwcelery.tasks.lvalert module

LVAlert client.

gwcelery.tasks.lvalert.**handler = {'burst_cwb': [<@task: gwcelery.tasks.superevents.handle**

Function decorator to register a handler callback for specified LVAlert message types. The decorated function is turned into a Celery task, which will be automatically called whenever a matching LVAlert message is received.

Parameters

- ***keys** – List of LVAlert message types to accept
- ****kwargs** – Additional keyword arguments for `celery.Celery.task()`.

Examples

Declare a new handler like this:

```
@lvalert.handler('cbc_gstlal',
                 'cbc_spiir',
                 'cbc_pycbc',
                 'cbc_mbtaonline')
def handle_cbc(alert_content):
    # do work here...
```

6.4.13 gwcelery.tasks.orchestrator module

This module implements the alert orchestrator, which responsible for the vetting and annotation workflow to produce preliminary, initial, and update alerts for gravitational-wave event candidates.

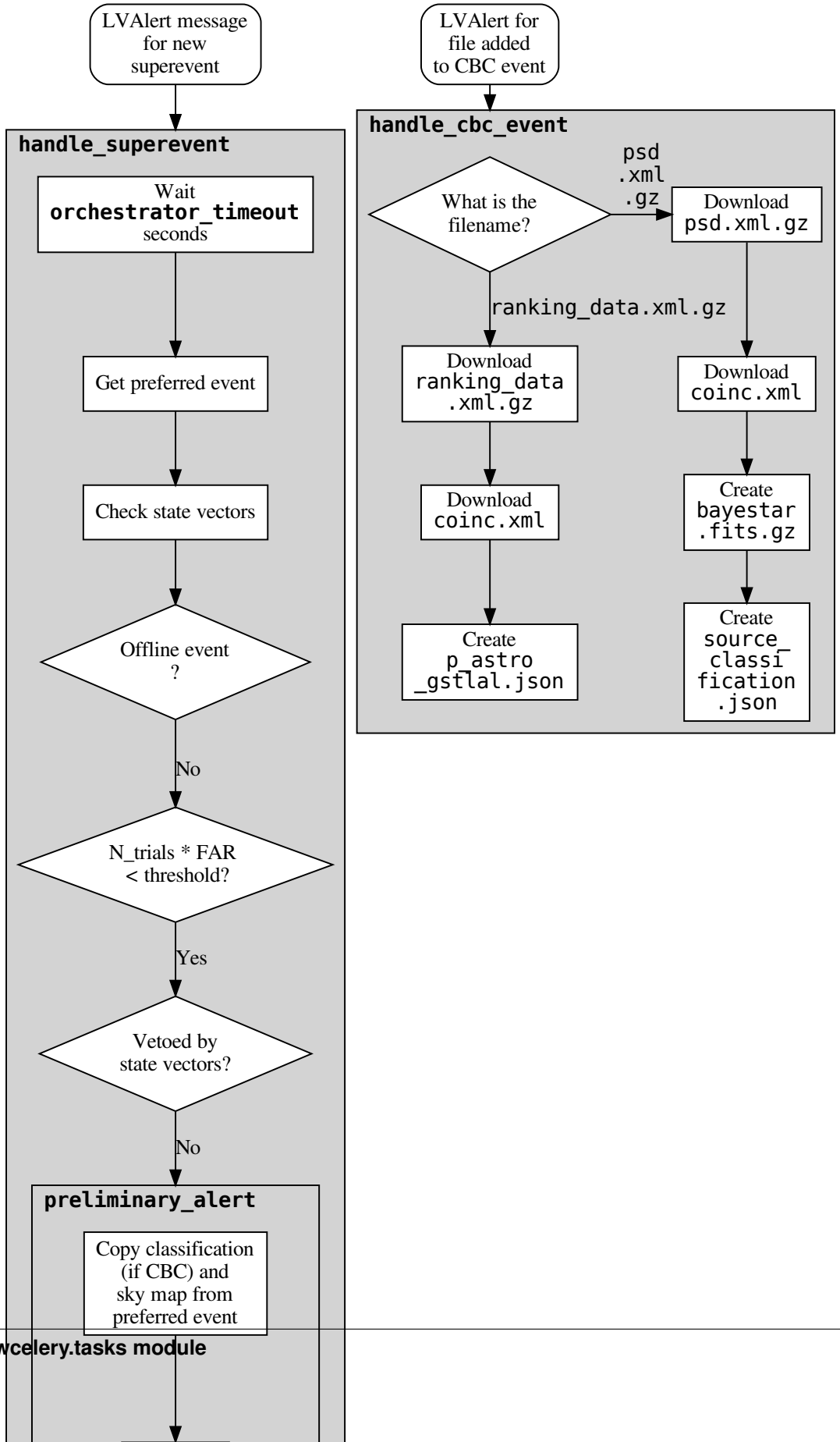
The orchestrator consists of two LVAlert message handlers:

- *handle_superevent()* is called for each superevent. It waits for a short duration of *orchestrator_timeout* seconds for the selection of the superevent by the *superevent manager* to stabilize, then performs data quality checks. If the data quality checks pass, then it calls *preliminary_alert()* to copy annotations from the preferred event and send the preliminary GCN notice.
- *handle_cbc_event()* is called for each CBC event. It performs some CBC-specific annotations that depend closely on the CBC matched-filter parameters estimates and that might influence selection of the preferred event: rapid sky localization with BAYESTAR and rapid source classification.

  Note that there is no equivalent of this task for burst events because both burst searches (cWB, LIB) have integrated source localization and have no other annotations.

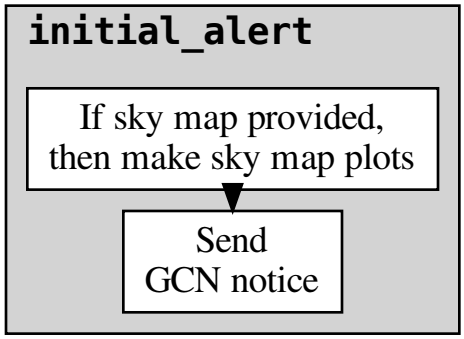
Preliminary Alerts

The flow chart below illustrates the operation of these two tasks.


LVAlert message for new superevent
LVAlert for file added to CBC event
handle_superevent
Wait orchestrator_timeout seconds
Get preferred event
Check state vectors
Offline event ?
No
N_trials * FAR < threshold?
Yes
Vetoed by state vectors?
No
preliminary_alert
Copy classification (if CBC) and sky map from preferred event
handle_cbc_event
What is the filename?
psd .xml .gz
Download psd.xml.gz
ranking_data.xml.gz
Download ranking_data .xml.gz
Download coinc.xml
Download coinc.xml
Create bayestar .fits.gz
Create p_astro _gstlal.json
Create source_ classi fication .json

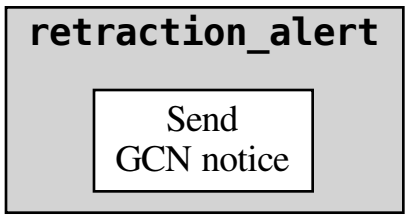
Initial and Update Alerts

The `initial_alert()` and `update_alert()` tasks create Initial and Update alerts respectively. At the moment, there is no handler or user interface to trigger these tasks, and they must be invoked manually (see *Command-Line Tools*). A flow chart for the initial alerts is shown below; the flow chart for update alerts is the same.



Retraction Alerts

Likewise, the `retraction_alert()` task creates Retraction alerts, and at the moment must be invoked manually. A flow chart is shown below.



Tasks

Tasks that comprise the alert orchestrator, which responsible for the vetting and annotation workflow to produce preliminary, initial, and update alerts for gravitational-wave event candidates.

(task) `gwcelery.tasks.orchestrator.`**`handle_selected_as_preferred`**(*alert*)

(task) gwcelery.tasks.orchestrator.**handle_superevent**(*alert*)
Schedule annotations for new superevents.

After waiting for a time specified by the *orchestrator_timeout* configuration variable for the choice of preferred event to settle down, this task performs data quality checks with *gwcelery.tasks.detchar.check_vectors()* and calls *preliminary_alert()* to send a preliminary GCN notice.

(task) gwcelery.tasks.orchestrator.**handle_cbc_event**(*alert*)
Perform annotations for CBC events that depend on pipeline-specific matched-filter parameter estimates.

Notes

This LVAlert message handler is triggered by updates that include the file `psd.xml.gz`. The table below lists which files are created as a result, and which tasks generate them.

File	Task
bayestar.multiorder.fits	*gwcelery.tasks.bayestar.localize()*
em_bright.json	gwcelery.tasks.em_bright.classifier()

(task) gwcelery.tasks.orchestrator.**handle_posterior_samples**(*alert*)
Generate multi-resolution and flat-resolution fits files and skymaps from an uploaded HDF5 file containing posterior samples.

(task) gwcelery.tasks.orchestrator.**_download**(**args*, ***kwargs*)
Download a file from GraceDB.

This works just like *gwcelery.tasks.gracedb.download()*, except that it is retried for both TimeoutError and URLError. In particular, it will be retried for 404 (not found) errors.

(task) gwcelery.tasks.orchestrator.**_update_if_dqok**(*superevent_id*, *event_id*)
Update *preferred_event* of *superevent_id* to *event_id* if *DQOK* label has been applied

(task) gwcelery.tasks.orchestrator.**_get_preferred_event**(*superevent_id*)
Determine preferred event for a superevent by querying GraceDB.

This works just like *gwcelery.tasks.gracedb.get_superevent()*, except that it returns only the preferred event, and not the entire GraceDB JSON response.

(task) gwcelery.tasks.orchestrator.**_create_voevent**(*classification*, **args*, ***kwargs*)
Create a VOEvent record from an EM bright JSON file.

Parameters

- **classification** (*tuple, None*) – A collection of JSON strings, generated by gwcelery.tasks.em_bright.classifier() and *gwcelery.tasks.p_astro.compute_p_astro()* or content of p_astro.json uploaded by gstlal respectively; or None
- ***args** – Additional positional arguments passed to *gwcelery.tasks.gracedb.create_voevent()*.
- ****kwargs** – Additional keyword arguments passed to *gwcelery.tasks.gracedb.create_voevent()*.

Returns The filename of the newly created VOEvent.

Return type str

(task) gwcelery.tasks.orchestrator.**preliminary_alert**(*event*, *superevent_id*)
Produce a preliminary alert by copying any sky maps.

This consists of the following steps:

1. Copy any sky maps and source classification from the preferred event to the superevent.
2. Create standard annotations for sky maps including all-sky plots by calling *gwcelery.tasks.skymaps.annotate_fits()*.
3. Create a preliminary VOEvent.
4. Send the VOEvent to GCN.
5. Apply the GCN_PRELIM_SENT label to the superevent.
6. Create and upload a GCN Circular draft.

(task) gwcelery.tasks.orchestrator.**_get_lowest_far**(*superevent_id*)
Obtain the lowest FAR of the events contained in the target superevent

(task) gwcelery.tasks.orchestrator.**parameter_estimation**(*far_event*, *superevent_id*)
Tasks for Parameter Estimation Followup with LALInference

This consists of the following steps:

1. Prepare and upload an ini file which is suitable for the target event.
2. Start Parameter Estimation if FAR is smaller than the PE threshold.

(task) gwcelery.tasks.orchestrator.**initial_or_update_alert**(*superevent_id*, *alert_type*, *skymap_filename=None*, *em_bright_filename=None*, *p_astro_filename=None*)
Create and send initial or update GCN notice.

Parameters

- **superevent_id** (*str*) – The superevent ID.
- **alert_type** (*{'initial', 'update'}*) – The alert type.
- **skymap_filename** (*str, optional*) – The sky map to send. If None, then most recent public sky map is used.
- **em_bright_filename** (*str, optional*) – The source classification file to use. If None, then most recent one is used.
- **p_astro_filename** (*str, optional*) – The p_astro file to use. If None, then most recent one is used.

(task) gwcelery.tasks.orchestrator.**initial_alert**(*superevent_id*, *skymap_filename=None*, *em_bright_filename=None*, *p_astro_filename=None*)
Produce an initial alert.

This does nothing more than call *initial_or_update_alert()* with `alert_type='initial'`.

Parameters

- **superevent_id** (*str*) – The superevent ID.
- **skymap_filename** (*str, optional*) – The sky map to send. If None, then most recent public sky map is used.
- **em_bright_filename** (*str, optional*) – The source classification file to use. If None, then most recent one is used.

- **p_astro_filename** (*str, optional*) – The p_astro file to use. If None, then most recent one is used.

(task) gwcelery.tasks.orchestrator.**update_alert**(*superevent_id*, *skymap_filename=None*, *em_bright_filename=None*, *p_astro_filename=None*)

Produce an update alert.

This does nothing more than call *initial_or_update_alert()* with `alert_type='update'`.

Parameters

- **superevent_id** (*str*) – The superevent ID.
- **skymap_filename** (*str, optional*) – The sky map to send. If None, then most recent public sky map is used.
- **em_bright_filename** (*str, optional*) – The source classification file to use. If None, then most recent one is used.
- **p_astro_filename** (*str, optional*) – The p_astro file to use. If None, then most recent one is used.

(task) gwcelery.tasks.orchestrator.**retraction_alert**(*superevent_id*)

Produce a retraction alert. This is currently just a stub and does nothing more than create and send a VOEvent.

6.4.14 gwcelery.tasks.p_astro module

Computation of `p_astro` by source category and utilities related to `p_astro.json` source classification files. See Kapadia et al (2019), arXiv:1903.06881, for details.

gwcelery.tasks.p_astro.**read_mean_values**()

Reads the mean values in the file pointed to by a url.

Returns **mean_values_dict** – mean values read from url file

Return type dictionary

(task) gwcelery.tasks.p_astro.**compute_p_astro**(*snr*, *far*, *mass1*, *mass2*, *pipeline*, *instruments*)

Task to compute *p_astro* by source category.

Parameters

- **snr** (*float*) – event's SNR
- **far** (*float*) – event's cfar
- **mass1** (*float*) – event's mass1
- **mass2** (*float*) – event's mass2
- **instruments** (*set*) – set of instruments that detected the event

Returns **p_astros** – JSON dump of the p_astro by source category

Return type str

Example

```
>>> p_astros = json.loads(compute_p_astro(files))
>>> p_astros
{'BNS': 0.999, 'BBH': 0.0, 'NSBH': 0.0, 'Terrestrial': 0.001}
```

(task) gwcelery.tasks.p_astro.**plot**(*contents*)

Make a visualization of the source classification.

Parameters **contents** (*str, bytes*) – The contents of the `p_astro.json` file.

Returns **png** – The contents of a PNG file.

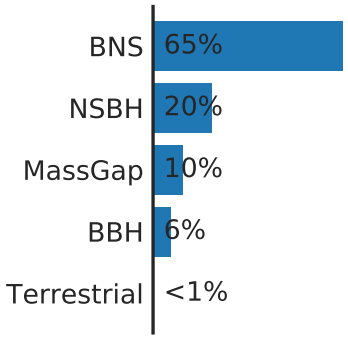
Return type bytes

Notes

The unusually small size of the plot (2.5 x 2 inches) is optimized for viewing in GraceDB's image display widget.

Examples

```
>>> from gwcelery.tasks import p_astro
>>> contents = '''
... {"Terrestrial": 0.001, "BNS": 0.65, "NSBH": 0.20,
... "MassGap": 0.10, "BBH": 0.059}
... '''
>>> p_astro.plot(contents)
```



(task) gwcelery.tasks.p_astro.**handle**(*alert*)

LVAlert handler to plot and upload a visualization of every `p_astro.json` that is added to a superevent.

6.4.15 gwcelery.tasks.raven module

Search for GRB-GW coincidences with ligo-raven.

gwcelery.tasks.raven.**calculate_coincidence_far**(*gracedb_id*, *group*)

Compute temporal coincidence FAR for external trigger and superevent coincidence by calling ligo.raven.search.calc_signif_gracedb.

Parameters

- **gracedb_id** (*str*) – ID of the superevent trigger used by GraceDB

- **group** (*str*) – CBC or Burst; group of the preferred_event associated with the gracedb_id superevent

(task) gwcelery.tasks.raven.**calc_signif**(*se_id*, *exttrig_id*, *tl*, *th*, *search*, *incl_sky*, *se_fitsfile=None*)

Calculate FAR of GRB exttrig-GW coincidence

gwcelery.tasks.raven.**coincidence_search**(*gracedb_id*, *alert_object*, *group=None*, *pipelines=[]*)

Perform ligo-raven search for coincidences. The ligo.raven.search.search method applies EM_COINC label on its own.

Parameters

- **gracedb_id** (*str*) – ID of the trigger used by GraceDB
- **alert_object** (*dict*) – lvalert['object']
- **group** (*str*) – Burst or CBC
- **pipelines** (*list*) – list of external trigger pipeline names

(task) gwcelery.tasks.raven.**search**(*gracedb_id*, *alert_object*, *tl=-5*, *th=5*, *group=None*, *pipelines=[]*)

Perform ligo-raven search for coincidences. The ligo.raven.search.search method applies EM_COINC label on its own.

Parameters

- **gracedb_id** (*str*) – ID of the trigger used by GraceDB
- **alert_object** (*dict*) – lvalert['object']
- **tl** (*int*) – number of seconds to search before
- **th** (*int*) – number of seconds to search after
- **group** (*str*) – Burst or CBC
- **pipelines** (*list*) – list of external trigger pipelines for performing coincidence search against

Returns

Return type list with the dictionaries of related gracedb events

(task) gwcelery.tasks.raven.**add_exttrig_to_superevent**(*raven_search_results*, *gracedb_id*)

Add external trigger to the list of em_events after ligo.raven.search.search finds a coincidence

Parameters

- **raven_search_results** (*list*) – list of dictionaries of each related gracedb trigger
- **gracedb_id** (*str*) – ID of either a superevent or external trigger

6.4.16 gwcelery.tasks.skymaps module

Annotations for sky maps.

(task) gwcelery.tasks.skymaps.**annotate_fits**(*filecontents*, *versioned_filename*, *graceid*, *tags*)

Perform annotations on a sky map.

This function downloads a FITS file and then generates and uploads all derived images as well as an HTML dump of the FITS header.

gwcelery.tasks.skymaps.**is_3d_fits_file**(*filecontents*)
Determine if a FITS file has distance information.

(task) gwcelery.tasks.skymaps.**annotate_fits_volume**(*filecontents*, **args*)
Perform annotations that are specific to 3D sky maps.

(task) gwcelery.tasks.skymaps.**fits_header**(*filecontents*, *filename*)
Dump FITS header to HTML.

(task) gwcelery.tasks.skymaps.**plot_allsky**(*filecontents*)
Plot a Mollweide projection of a sky map using the command-line tool ligo-skymap-plot.

(task) gwcelery.tasks.skymaps.**plot_volume**(*filecontents*)
Plot a 3D volume rendering of a sky map using the command-line tool ligo-skymap-plot-volume.

(task) gwcelery.tasks.skymaps.**flatten**(*filecontents*, *filename*)
Convert a HEALPix FITS file from multi-resolution UNIQ indexing to the more common IMPLICIT indexing using the command-line tool ligo-skymap-flatten.

(task) gwcelery.tasks.skymaps.**skymap_from_samples**(*samplefilecontents*)
Generate multi-resolution fits file from samples

6.4.17 gwcelery.tasks.superevents module

Superevents are an abstraction to unify gravitational-wave candidates from multiple search pipelines. Each superevent is intended to represent a single astrophysical event. A superevent consists of one or more event candidates, possibly from different pipelines, that are neighbors in time. At any given time, one event belonging to the superevent is identified as the *preferred event*.

This module provides the Superevent Manager, an LVAlert handler that creates and updates superevents whenever new events are uploaded to GraceDB.

Events are only considered for membership in a superevent if their false alarm rate is less than or equal to the value of the *superevent_far_threshold* configuration setting.

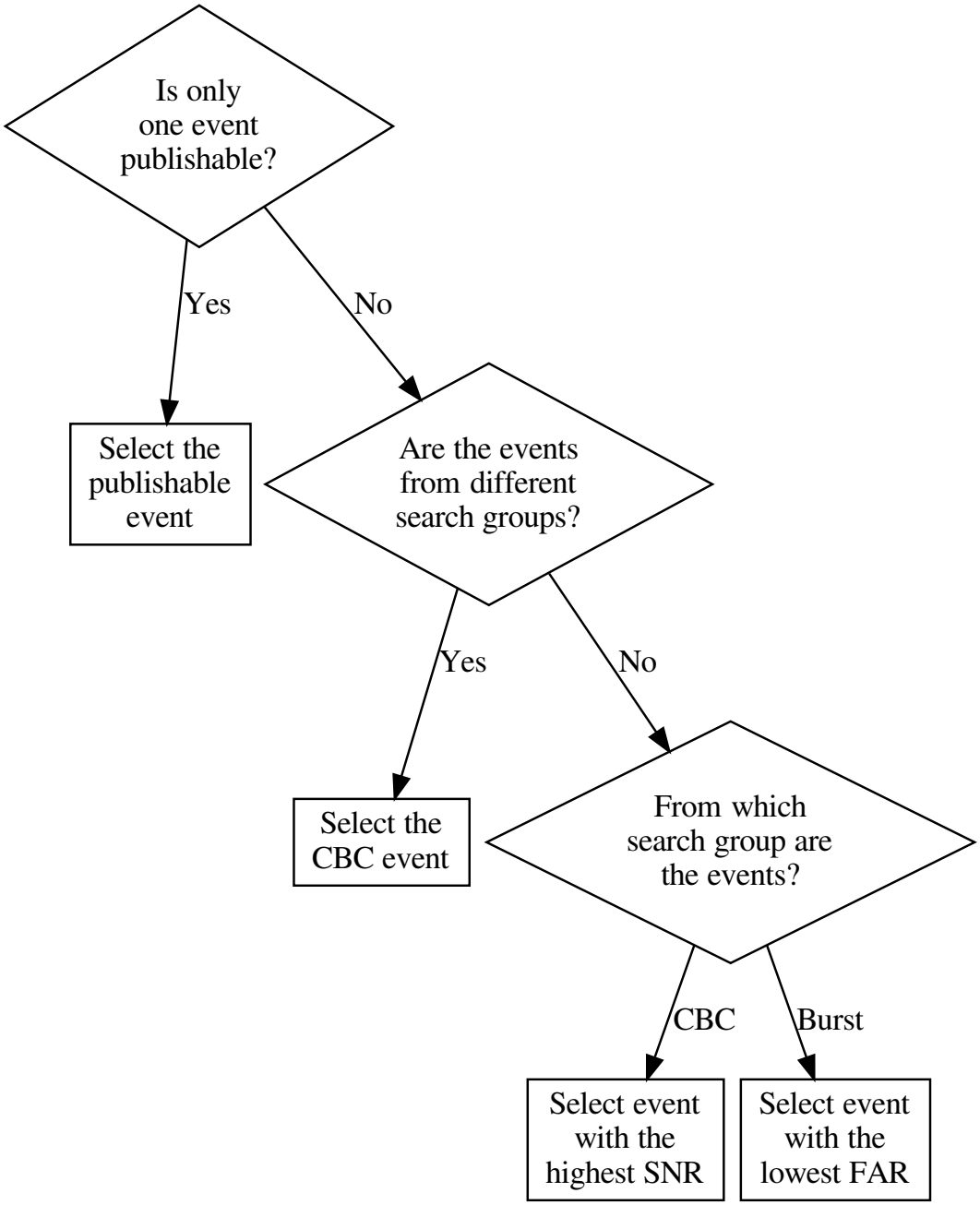
Each superevent has a time window described by a central time `t_0`, a start time `t_start`, and a end time `t_end`. The central time `t_0` is just the time of the preferred event. The start and end time are extended to encompass all of the events that belong to the superevent (see *get_ts()*).

Selection of the preferred event

When a new event is added to a superevent, it may or may not become the new preferred event. The preferred event is selected by considering the following factors in order to resolve any ties:

1. **Publishability**: Would the event eligible, as determined by the function *should_publish()*, for sending an automated public alert?
2. **Search group**: Is it a CBC event or a burst event? CBC events takes precendence.
3. **Significance**: For CBC events, which has the highest SNR? For burst events, which has the lowest FAR?

The selection of the preferred event from a pair of events is illustrated by the decision tree below.



Tasks

Module containing the functionality for creation and management of superevents.

- There is serial processing of triggers from low latency pipelines.

- Dedicated **superevent** queue for this purpose.
- Primary logic to respond to low latency triggers contained in *process()* function.

(task) gwcelery.tasks.superevents.**handle**(*payload*)
Respond to lvalert nodes from low-latency search pipelines and delegate to *process()* for superevent management.

(task) gwcelery.tasks.superevents.**process**(**args*, ***kwargs*)
Respond to *payload* and serially processes them to create new superevents, add events to existing ones and update superevent parameters.

Parameters **payload** (*dict*) – LVAlert payload

gwcelery.tasks.superevents.**get_ts**(*event*)
Get time extent of an event, depending on pipeline-specific parameters.

- For CWB, use the event's `duration` field.
- For oLIB, use the ratio of the event's `quality_mean` and `frequency_mean` fields.
- For all other pipelines, use the *superevent_d_t_start* and *superevent_d_t_start* configuration values.

Parameters **event** (*dict*) – Event dictionary (e.g., the return value from *gwcelery.tasks.gracedb.get_event()*).

Returns

- **t_0** (*float*) – Segment center time in GPS seconds.
- **t_start** (*float*) – Segment start time in GPS seconds.
- **t_end** (*float*) – Segment end time in GPS seconds.

gwcelery.tasks.superevents.**get_snr**(*event*)
Get the SNR from the LVAlert packet.

Different groups and pipelines store the SNR in different fields.

Parameters **event** (*dict*) – Event dictionary (e.g., the return value from *gwcelery.tasks.gracedb.get_event()*).

Returns **snr** – The SNR.

Return type float

gwcelery.tasks.superevents.**get_instruments**(*event*)
Get the participating instruments from the LVAlert packet.

Parameters **event** (*dict*) – Event dictionary (e.g., the return value from *gwcelery.tasks.gracedb.get_event()*).

Returns The set of instruments that contributed to the ranking statistic for the event.

Return type set

Notes

The number of instruments that contributed *data* to an event is given by the `instruments` key of the GraceDB event JSON structure. However, some pipelines (e.g. gstlal) have a distinction between which instruments contributed *data* and which were considered in the *ranking* of the candidate. For such pipelines, we infer which

pipelines contributed to the ranking by counting only the SingleInspiral records for which the chi squared field is non-empty.

gwcelery.tasks.superevents.**should_publish**(*event*)
Determine whether an event should be published as a public alert.

All of the following conditions must be true for a public alert:

- The event's `offline` flag is not set.
- The event's false alarm rate, weighted by the group-specific trials factor as specified by the *preliminary_alert_trials_factor* configuration setting, is less than or equal to *preliminary_alert_far_threshold*.

Parameters **event** (*dict*) – Event dictionary (e.g., the return value from *gwcelery.tasks.gracedb.get_event()*).

Returns **should_publish** – `True` if the event meets the criteria for a public alert or `False` if it does not.

Return type bool

gwcelery.tasks.superevents.**keyfunc**(*event*)
Key function for selection of the preferred event.

Return a value suitable for identifying the preferred event. Given events `a` and `b`, `a` is preferred over `b` if `keyfunc(a) < keyfunc(b)`, else `b` is preferred.

Parameters **event** (*dict*) – Event dictionary (e.g., the return value from *gwcelery.tasks.gracedb.get_event()*).

Returns **key** – The comparison key.

Return type tuple

Notes

Tuples are compared lexicographically in Python: they are compared element-wise until an unequal pair of elements is found.

6.5 gwcelery.tools module

Custom Celery subcommands. The subcommand that is implemented in `gwcelery.tools.`*`submodule`* can be invoked from the command line as `gwcelery` *`submodule`*.

6.5.1 gwcelery.tools.condor module

Shortcuts for HTCondor commands to manage deployment of GWCelery on LIGO Data Grid clusters.

These commands apply to the GWCelery instance that is running in the current working directory.

gwcelery.tools.condor.**get_constraints**()

gwcelery.tools.condor.**run_exec**(**args*)

gwcelery.tools.condor.**running**()
Determine if GWCelery is already running under HTCondor.

gwcelery.tools.condor.**submit**()
Submit all GWCelery jobs to HTCondor (if not already running).

gwcelery.tools.condor.**resubmit**()
Remove any running GWCelery jobs and resubmit to HTCondor.

gwcelery.tools.condor.**rm**()
Remove all GWCelery jobs.

gwcelery.tools.condor.**hold**()
Put all GWCelery jobs on hold.

gwcelery.tools.condor.**release**()
Release all GWCelery jobs from hold status.

gwcelery.tools.condor.**q**()
Show status of all GWCelery jobs.

class gwcelery.tools.condor.**CondorCommand**(*app=None*, *get_app=None*, *no_color=False*, *stdout=None*, *stderr=None*, *quiet=False*, *on_error=None*, *on_usage_error=None*)
Bases: celery.bin.base.Command

Shortcuts for HTCondor commands to manage deployment of GWCelery on LIGO Data Grid clusters.

These commands apply to the GWCelery instance that is running in the current working directory.

add_arguments(*parser*)

run(*func=None*, ***kwargs*)

6.5.2 gwcelery.tools.flask module

Flask web application for manually triggering certain tasks.

gwcelery.tools.flask.**main**()
Flask web application for manually triggering certain tasks.

class gwcelery.tools.flask.**FlaskCommand**(*app=None*, *get_app=None*, *no_color=False*, *stdout=None*, *stderr=None*, *quiet=False*, *on_error=None*, *on_usage_error=None*)
Bases: celery.bin.base.Command

Flask web application for manually triggering certain tasks.

add_arguments(*parser*)

run(**args*, *flask_args=()*, ***kwargs*)

6.5.3 gwcelery.tools.nagios module

A Nagios plugin for monitoring GWCelery.

class gwcelery.tools.nagios.**NagiosPluginStatus**
Bases: enum.IntEnum

Nagios plugin status codes.

OK = 0

WARNING = 1

CRITICAL = 2

UNKNOWN = 3

exception gwcelery.tools.nagios.**NagiosCriticalError**
Bases: Exception

An exception that maps to a Nagios status of *CRITICAL*.

gwcelery.tools.nagios.**get_active_queues**(*inspector*)

gwcelery.tools.nagios.**get_active_lvalert_nodes**(*inspector*)

gwcelery.tools.nagios.**get_expected_queues**(*app*)

gwcelery.tools.nagios.**get_expected_lvalert_nodes**(*app*)

gwcelery.tools.nagios.**get_active_voevent_peers**(*inspector*)

gwcelery.tools.nagios.**check_status**(*app*)

class gwcelery.tools.nagios.**NagiosCommand**(*app=None*, *get_app=None*, *no_color=False*, *stdout=None*, *stderr=None*, *quiet=False*, *on_error=None*, *on_usage_error=None*)
Bases: celery.bin.base.Command

A Nagios plugin for monitoring GWCelery.

run(***kwargs*)

6.6 gwcelery.util module

Miscellaneous utilities that are useful inside many different tasks.

class gwcelery.util.**PromiseProxy**(**args*, ***kwargs*)
Bases: object

gwcelery.util.**NamedTemporaryFile**(*content=None*, ***kwargs*)
Convenience wrapper for tempfile.NamedTemporaryFile() that writes some data to the file before handing it to the calling code.

Parameters

- **content** (*str*, *bytes*, *None*) – Initial contents of the file.
- ****kwargs** – Additional keyword arguments to pass to tempfile.NamedTemporaryFile().

6.7 gwcelery.voevent module

Embed a Comet VOEvent broker and subscriber into a Celery worker by extending Celery with bootsteps.

gwcelery.voevent.**install**(*app*)
Register the VOEvent subsystem in the application boot steps.

6.7.1 gwcelery.voevent.bootsteps module

class gwcelery.voevent.bootsteps.**Reactor**(*consumer*, ***kwargs*)
Bases: gwcelery.voevent.bootsteps.VOEventBootStep

Run the global Twisted reactor in background thread.

The Twisted reactor is a global run loop that drives all Twisted services and operations. This boot step starts the Twisted reactor in a background thread when the Celery consumer starts, and stops the thread when the Consumer terminates.

name = 'Twisted reactor'

create(*consumer*)
Create the step.

start(*consumer*)

stop(*consumer*)

class gwcelery.voevent.bootsteps.**Broadcaster**(*consumer*, ***kwargs*)
Bases: gwcelery.voevent.bootsteps.TwistedService

Comet-based VOEvent broadcaster.

Run a Comet-based VOEvent broadcaster (comet.protocol.broadcaster.VOEventBroadcasterFactory). Starts after the *Reactor* bootstep.

A few *configuration options* are available:

- voevent_broadcaster_address: The address to bind to, in *host*:*port* format.
- **voevent_broadcaster_whitelist: A list of hostnames, IP addresses, or** CIDR address ranges from which to accept connections.

The list of active connections is made available inspection with the gwcelery inspect stats command under the voevent-broker-peers key.

name = 'VOEvent broadcaster'

create_service(*consumer*)

info(*consumer*)

class gwcelery.voevent.bootsteps.**Receiver**(*consumer*, ***kwargs*)
Bases: gwcelery.voevent.bootsteps.TwistedService

VOEvent receiver.

Run a Comet-based VOEvent receiver (comet.protocol.subscriber.VOEventSubscriberFactory). Starts after the *Reactor* bootstep.

A few *configuration options* are available:

- voevent_receiver_address: The address to connect to, in *host*:*port* format.

The list of active connections is made available inspection with the gwcelery inspect stats command under the voevent-receiver-peers key.

name = 'VOEvent receiver'

requires = ('celery.worker.consumer:Connection', step:Twisted reactor{('celery.worker.

create_service(*consumer*)

info(*consumer*)

6.7.2 gwcelery.voevent.logging module

Integration between the Celery, Twisted, and Comet logging systems.

gwcelery.voevent.logging.**after_setup_logger**(*logger*, *loglevel*, ***kwargs*)
Celery signal handler to set up capturing of all log messages from Comet and Twisted.

- Celery uses the Python standard library's `logging` module. Twisted has its own separate logging facility. Use Twisted's `PythonLoggingObserver` to forward all Twisted log messages to the Python `logging` module.
- Comet uses the Twisted logging facility, but has its own separate management of log severity level (e.g., *info*, *debug*). Set Comet's log level to match Celery's.

6.7.3 gwcelery.voevent.util module

VOEvent-related utilities.

gwcelery.voevent.util.**get_host_port**(*address*)
Split a network address of the form `host:port`.

Parameters **network** (*str*) – The network address.

Returns

- **host** (*str*) – The hostname, or an empty string if missing.
- **port** (*int, None*) – The port number, or None if missing.

gwcelery.voevent.util.**get_local_ivo**(*app*)
Create an IVOID to identify this application in VOEvent Transport Protocol packets.

Returns A local IVOID composed of the machine's fully qualified domain name and the Celery application name (for example, *ivo://emfollow.ligo.caltech.edu/gwcelery*).

Return type str

gwcelery.voevent.util.**get_network**(*address*)
Find the IP network prefix for a hostname or CIDR notation.

Parameters **address** (*str*) – A hostname, such as `ligo.org`, or an IP address prefix in CIDR notation, such as `127.0.0.0/8`.

Returns An object representing the IP address prefix.

Return type ipaddress.IPv4Network

6.7.4 gwcelery.voevent.signals module

Definitions of custom Celery signals related to VOEvents.

These signals allow us to keep the VOEvent broker code decoupled from any GCN-specific logic. Notably, it allows us to keep all of the details of the GCN-specific "Notice Type" concept out of *gwcelery.voevent*.

gwcelery.voevent.signals.**voevent_received = <Signal: voevent_received providing_args={'xml**
Fired whenever a VOEvent is received.

Parameters **xml_document** (comet.utility.xml.xml_document) – The XML document that was received. The raw file contents are available as `xml_document.raw_bytes`. The `lxml.etree` representation of the document is available as `xml_document.element`.

CHAPTER 7

Contributing

Contributors may familiarize themselves with Celery itself by going through the First Steps with Celery tutorial.

7.1 Development model

GWCelery operates on a fork-and-merge development model (see GitLab basics for an introduction).

To contribute to GWCelery development, follow these steps:

1. Create a personal fork of GWCelery.
2. Make your changes on a branch.
3. Open a merge request.

Note that GWCelery uses fast-forward merges.

7.2 Where new code should go

New code will generally consist of adding Celery tasks. Tasks are organized by functionality into submodules of `gwcelery.tasks`. If your new task does not match with one of the existing submodules, please create a new submodule.

7.3 Guidelines for tasks

- **Tasks should be short.** When deciding where a new task should go, start from the following loose rules of thumb:
    1. If it's less than a screenful of code, and related to functionality in an existing module, then put the code in a new task in that module.

2. If it's up to a few screenfuls of code, or not related to functionality in an existing module, then try to break it into a few smaller functions or tasks and put it in a new module.
3. If it's more than a few screenfuls of code, or adds many additional dependencies, then it should go in a separate package.

See also the note on Granularity in the Celery manual's Tips and Best Practices section.

- **Tasks should avoid saving files to disk.** Output should be placed directly in GraceDB. Temporary files that are written in `/tmp` are OK but should be cleaned up promptly.

  See also the Celery manual's notes on Data locality and State.

- **Dependencies should be installable by pip.** Dependencies of tasks should be listed in the requirements.txt file so that they are installed automatically when GWCelery is installed with pip.

  There are two extra steps involved in making changes to the dependencies:

  1. The Sphinx-generated documentation (that is to say, this manual) is generally built without most of the dependencies installed. Whenever you add a new package to requirements.txt, you should also add any modules that are imported from that package to the `autodoc_mock_imports` list in the Sphinx configuration file, doc/conf.py.
  2. We use pipenv to make the precise versions of packages reproducible in our deployment. If you make changes to requirements.txt, then run `pipenv update` and commit the changes to Pipfile.lock.

7.4 Unit tests

Unit tests and code coverage measurement are run automatically for every branch and for every merge request. New code contributions must have 100% test coverage. Modifications to existing code must not decrease test coverage. To run the unit tests and measure code coverage, run the following commands in the top directory of your local source checkout:

```
$ pip install pytest-cov
$ python setup.py test --addopts='--cov --cov-report html'
```

This will save a coverage report that you can view in a web browser as `htmlcov/index.html`.

7.5 Code style

Code should be written in the **PEP 8** style and must pass linting by Flake8. To check code style, run the following commands in the top of your source directory:

```
$ pip install flake8 pep8-naming
$ flake8 --show-source .
```

7.6 Documentation

Documentation strings should be written in the Numpydoc style.

To build the documentation, run the following command in the top of your source directory:

```
$ python setup.py build_sphinx
```

Then to view the documentation, open the file `build/sphinx/html/index.html` in your favorite web browser.

CHAPTER 8

Deployment

8.1 Continuous deployment

GWCelery is automatically deployed using GitLab's continuous deployment features, configured through the project's .gitlab-ci.yml file. Deployment can be managed through the GitLab project's Environments page.

Python dependencies in the deployment environment are managed automatically using pipenv.

There are two instances of GWCelery that are running on the LIGO-Caltech computing cluster and that are managed in this manner:

- **Playground**: The playground instance is re-deployed *on every push to master that passes the unit tests*. It uses the `gwcelery.conf.playground` configuration preset.
- **Production**: The production instance is re-deployed *only when manually triggered through GitLab*. It uses the `gwcelery.conf.production` configuration preset.

When we observe that the Playground instance shows correct end-to-end behavior, we have the option of triggering a re-deployment to Production. Deployment to production should preferably occur at a release. The procedure for performing a release is described below.

Danger: It is possible to start an interactive session inside the GWCelery production environment by logging in to the LIGO-Caltech cluster, but this measure should be **reserved for emergencies only**.

Any manual changes to the environment **may disrupt the logging and monitoring subsystems**. Any files that are manually changed, added to, or removed from the deployment environment **will not be captured in version control** and may be **rolled back without warning** the next time that the continuous deployment is triggered.

8.2 Making a new release

We always prepare releases from the tip of the `master` branch. GitLab is configured through the project's .gitlab-ci.yml file to automatically build and push any tagged release to the Python Package Index (PyPI). Follow these steps

when issuing a release in order to maintain a consistent and orderly change log.

1. **Check the pipeline status.** Before you begin, first make sure that the unit tests, documentation, and packaging jobs are passing. Consult the project's GitLab pipeline status to make sure that all of the continuous integration jobs are passing on `master`.

   If necessary, fix any bugs that are preventing the pipeline from passing, push the changes to master, and repeat until all jobs pass.

2. **Update the change log.** The first subsection of the change log file, CHANGES.rst, should have the title *`MAJOR.MINOR.PATCH`* `(unreleased)`, where *`MAJOR.MINOR.PATCH`* will be the version number of the new release. Review the git commit log.

   Make any necessary changes to CHANGES.rst so that this subsection of the change log accurately summarizes all of the significant changes since the last release and is free of spelling, grammatical, or reStructuredText formatting errors.

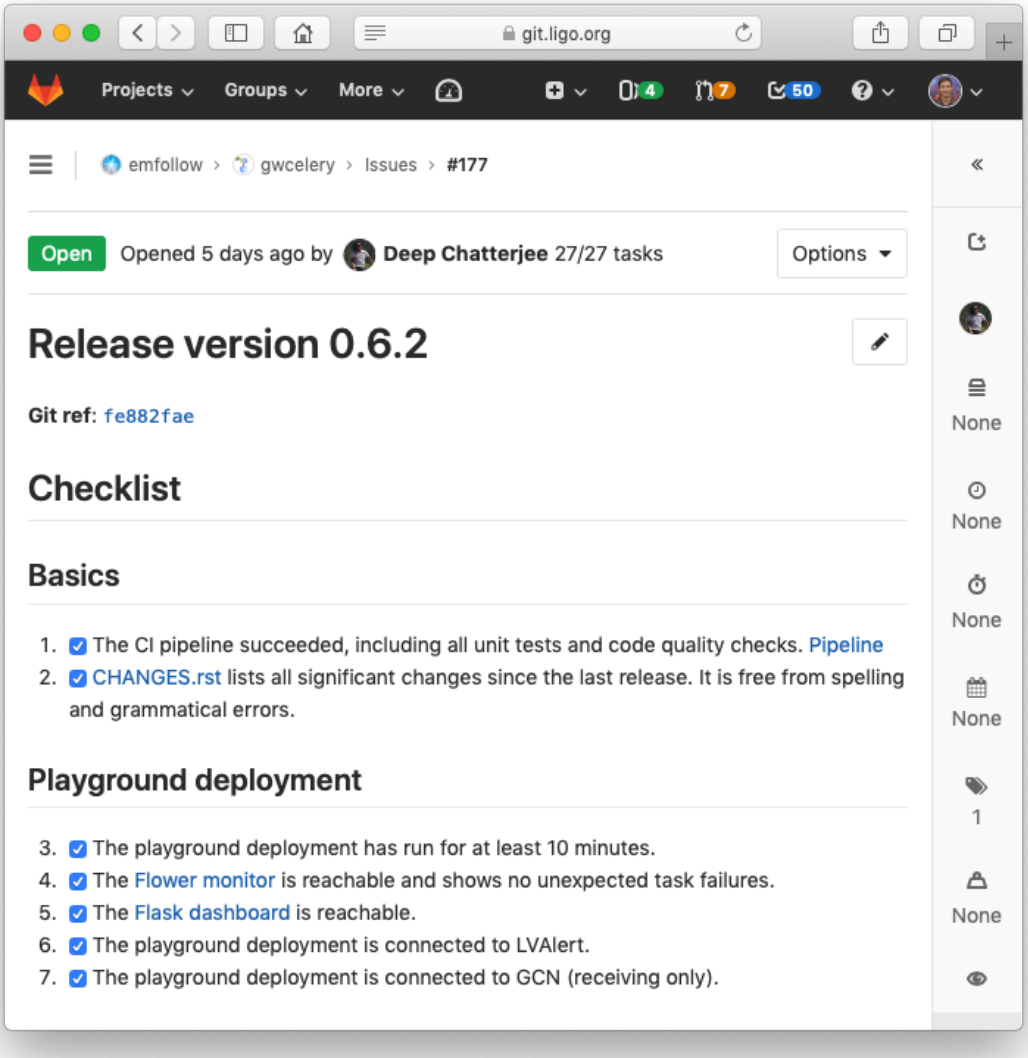
   Review the list of changes and make sure that the new version number is appropriate. We follow SemVer *very* loosely, and also generally bump at least the minor version number at the start of a new LSC/Virgo engineering or observing run.

   Commit and push any corrections to CHANGES.rst to `master`.

3. **Complete the acceptance tests.** Our acceptance tests consist of a manual checklist for verifying that the pipeline satisfies certain requirements on the playground environment. The checklist is maintained as a GitLab issue template and is under version control in the special directory .gitlab/issue_templates.

   Create a new issue in GitLab. Set the title to `Release version` *`MAJOR.MINOR.PATCH`*. In the `Choose a template` dropdown menu, select `Create a Release`. The description field will be automatically populated with the checklist. Submit the issue.

   Complete the items in the checklist and check them off one by one on the release issue before proceeding to the next step.



4. **Tag the release.** Change the title of the first section of CHANGES.rst to *MAJOR.MINOR.PATCH* (*YYYY-MM-DD*) where *YYYY-MM-DD* is today's date. Commit with the message `Update changelog for version MAJOR.MINOR.PATCH; closes #N`, where *N* is the release issue's number.

   Create a git tag to mark the release by running the following command:

   ```
   $ git tag vMAJOR.MINOR.PATCH -m "Version MAJOR.MINOR.PATCH"
   ```

5. **Create a change log section for the next release.** Add a new section to CHANGES.rst with the title *NEXT_MAJOR.NEXT_MINOR.NEXT_PATCH* (unreleased), where *NEXT_MAJOR.NEXT_MINOR.NEXT_PATCH* is a provisional version number for the next release. Add a single list item with the text `No changes yet.` Commit with the message `Back to development.`

6. **Push the new tag and updated change log.** Push the new tag and updated change log:

   ```
   git push && git push --tags
   ```

7. Wait a couple minutes, and then verify that the new release has been published on our PyPI project page, https://pypi.org/project/gwcelery/.

8. If desired, navigate to the GitLab project's Environments page and trigger a deployment to production.

CHAPTER 9

Changelog

9.1 0.8.1 (2019-07-29)

- Downgrade lalsuite to 6.59.
- Revert change that tried to fix incorrect key for querying external events. The keys were correct before.

9.2 0.8.0 (2019-07-26)

- Assign `gwcelery.tasks.skymaps.plot_volume` tasks a reduced Celery priority as compared to `gwcelery.tasks.bayestar.localize` so that the latter are given preference. This ought to speed up the preparation of preliminary GCN notices because only the latter are required for GCNs but both kinds of tasks compete for slots in the resource-intensive OpenMP queue.
- Reduce priority for CBC annotation tasks for events that do not pass the public alert threshold.
- Update lalsuite to 6.60.
- Ensure gracedb calls to create and update superevents are retried in the event of transient GraceDB API errors.
- Update ligo-raven version to 1.15. Apply EM_COINC label in raven.py to give more control and prevent race conditions.
- Use the space-time coincidence FAR as the default for RAVEN, use the temporal coincidence FAR when sky maps are not available.
- Check if GRB is sub-threshold, set search to be 'SubGRB'. Pass search through external triggers pipeline and RAVEN.
- Tune Celery's `result_expires` setting from its default value of one day to five minutes. Since we pass large byte strings as task arguments and return values, one day is too long to keep task tombstones in the database. This adjustment should reduce the memory footprint of the Redis server during periods with very high rates of GraceDB uploads.

  The downside is that task details will remain browsable in Flower for a much shorter period.

- Remove p_astro_gstlal.py module, corresponding test modules, and documentation; p_astro will be reported as a pipeline product from gstlal. The computation of p_astro for all other pipelines is unaffected.
- Fix EM_COINC bug where it is being over-applied to superevents.
- Fix bug where wrong key was called for querying external events.

9.3 0.7.1 (2019-07-12)

- The initial alert workflow will now consider only `*.fits.gz` sky maps and not `*.fits` sky maps for GCN Notices. It was an oversight that we did not exclude `*.fits` files from the list of extensions to consider when we updated the handling of multiresolution sky maps.
- Catch and retry HTTP 429 ("Too Many Requests") errors from GraceDB.
- Enable Sentry integration for Tornado in order to capture errors from the Flower console.
- Fix file extensions for LALInference sky map PNG files: they should be named `LALInference.png`, not `LALInference.multiorder.png`.
- Increase the Redis server's log verbosity in order to help diagnose Redis client connection dropouts.
- Run sky map plotting and annotation tasks asynchronously so that they do not block sending preliminary alerts. Their outputs are only for human consumption; they are not needed in order to prepare GCN Notices.

9.4 0.7.0 (2019-06-21)

- Trigger a preliminary alert for a superevent upon the first time that the preferred event is set to an event that meets the public alert criterion.

  This fixes a longstanding issue that has prevented automated preliminary alerts from being sent so far. The preferred event *at the instant that the timeout ended* did not meet the public alert criterion, but a preferred event that was selected some tens of seconds later did.
- Decrease preliminary alert timeout to one minute.
- The combined effect of these changes should be to decrease the latency for producing preliminary alerts from 7 minutes to 2 minutes.

9.5 0.6.3 (2019-06-14)

- Work around a Celery canvas bug that prevented LALInference postprocessing from completing.
- Fix a copy-paste error that caused `DQV` and `INJ` labels to be ignored when determining whether to send a preliminary alert.
- Move RAVEN time coincidence windows to the application configuration.
- Document the acceptence tests checklist in the instructions for preparing a release.
- Update ligo-raven to version 1.14.

9.6 0.6.2 (2019-06-07)

- Add a dependency on dnspython to silence the following warning message from SleekXMPP:

```
DNS: dnspython not found. Can not use SRV lookup.
```

- Pin some recently updated dependencies of Celery that caused unit test failures: amqp <= 2.4.2, kombu <= 4.5.0, vine <= 1.3.0.
- Prevent subthreshold GRBs with low reliability from being processed as external events.
- Add a task in orchestrator.py to generate FITS files and sky map images automatically whenever an HDF5 posterior samples file is uploaded.
- Remove special-case handling of single-instrument events. Now, the eligibility of an event for a public alert is determined only on the basis of its false alarm rate.
- Run parameter estimation on nodes dedicted to online-PE.
- Emcoinc circular is triggered when RAVEN uploads a coincident FAR.
- Pin scipy since scipy>=1.3.0 removes an interpolation function which lalinference postprocessing requires.

9.7 0.6.1 (2019-05-24)

- Work around a bug in the Sentry Python SDK that caused excessive reporting of certain GraceDB exceptions that are listed in tasks' `autoretry_for` settings. See getsentry/sentry-python#370.
- Change the name of BAYESTAR localization files to `bayestar.multiorder.fits` to distinguish them from flat-resolution HEALPix files, which are still named `bayestar.fits.gz`.
- Reimplement LVAlert listener as a Celery bootstep to avoid needing to track a singleton task using a Redis lock, because Redis locks do not play nicely with Redis persistence. The `--lvalert` command line option must now be passed in order to enable the LVAlert listener.
- Turn on Redis database persistence so that Celery task state is preserved across restarts.
- Add `expose_to_public` setting to disable exposing GraceDB events to the public in all environments except for production.
- Update to the latest version of GWPy and un-pin Matplotlib because GWPy now supports Matplotlib 3.1.
- Pin LALSuite to version 6.54 because LALInference in LALSuite 6.55 is not compatible with Python 3.

9.8 0.6.0 (2019-05-20)

- Work around a bug in complex Celery canvases (see celery/celery#5512) that prevented initial GCN notices from being sent. As a side effect of this workaround, the initial, update, and retraction canvases will not automatically expose events to the public.

  The preliminary alert canvas still *does* expose events to the public, so under normal circumstances, the follow-up advocate should not have to manually do that. However, if the event has not been exposed to the public for whatever reason, then the follow-up advocate should expose it to the public manually before applying the `ADVOK` label. See emfollow/followup-advocate-guide!2.
- Reduce the false alarm rate threshold for parameter estimation to decrease cluster load.
- Remove redundant LVAlert subscription in handle_lvalert_grb to prevent double calls to RAVEN.

- Read template weights for P_astro from hdf5 file using h5py for speedup.
- Require matplotlib < 3.1 becuase matplotlib 3.1 breaks importing gwpy:

```
/usr/local/lib/python3.7/site-packages/gwpy/plot/rc.py:79: in <module>
    rcParams.get('text.latex.preamble', []) + tex.MACROS),
E   TypeError: can only concatenate str (not "list") to str
```

- Make `gwcelery.tasks.gracedb.get_superevents` and `gwcelery.tasks.gracedb.get_events` take any number of keyword arguments to be passed to corresponding client methods.
- Update the superevent `t_0` field whenever the preferred event changes.

9.9 0.5.7 (2019-05-13)

- If the VOEvent broker is disabled by setting `voevent_broker_whitelist` to an empty list, then suppress the normal error message that would occur when attempting to send a VOEvent when there are no broker connections.
- Rearrange preliminary alert workflow so that sky map plots are generated for the newly added FITS file rather than an older FITS file that coincidentally has the same name.
- Have `gwcelery.detchar.check_vectors` task apply all GraceDB log messages in order to increase robustness to recoverable GraceDB API errors.
- Port over majority of P_astro code from gwcelery to the p-astro package.
- Use cleaned data for parameter estimation.
- The `DQOK` and `DQV` labels should be mutually exclusive. When `gwcelery.tasks.detchar.check_vectors` adds one of the `DQOK` or `DQV` labels, it will now first remove the other label.
- Change exception in VOEevent parsing of Fermi subtreshold alerts to match real incoming alerts.
- Update Celery to 4.3.0.
- Automatically select the most up-to-date calibration uncertainties for parameter estimation.

9.10 0.5.6 (2018-05-08)

- Extend the `orchestrator_timeout` to 300s and the `pe_timeout` to 345s. The previous timeout was not sufficient for the online pipelines to upload all of their possible candidates, hence the extension.

9.11 0.5.5 (2019-05-03)

- Cycle through llhoft, high latency frames, and low latency frames in detchar's cache creation.
- Add explanations on options in online_pe.jinja2 for those who start parameter estimation based on the ini files uploaded to GraceDB.
- Calculate horizon distance with psd.xml.gz to determine the upper limit of distance prior for parameter estimation.
- Start parameter estimation when the lowest FAR of the events in a superevent is lower than the threshold.
- Update the calibration uncertainties used for parameter estimation.

- Handle an exception in VOEvent parsing of Fermi subthreshold alerts due to different param names.
- Stop uploading corner plots of intrinsic parameters.
- Connect to different GCN servers to receive alerts in the production and playground environments, because GCN does not support multiple receiver connections from the same client IP address to the same server.
- Change the preferred event assignment logic to not let accidental candidates like G330298 which have low FAR but high SNR values to become the preferred event. From now on, `superevents.should_publish` takes maximum precedence for selecting the preferred event. The same is also used by orchestrator to expose events.
- Update RAVEN coinc FAR task call which uses string params versus un-pickleable class object params.
- Make sure to consume the entire response from every GraceDB API request. This will ensure that GraceDB API call has completed before the pipeline continues, and will fix errors like we encountered with S190426c where the pipeline would march along before uploads had finished.
- Apply ADVREQ label earlier in the preliminary alert workflow.
- Update LALSuite to version 6.54. We are now using a stable version again instead of a nightly build.
- Add Nagios checks for GCN connectivity.
- Improve uploaded comments so that it is easily understood which event has triggered parameter estimation.

9.12 0.5.4 (2019-05-01)

- Provide a value for terrestrial count for P_astro for non-gstlal pipelines that is consistent with the FAR threshold used.

9.13 0.5.3 (2019-04-17)

- Update ligo-followup-advocate to 0.0.28.
- Stop using unreviewed cleaned data for parameter estimation.
- Update detchar check to analyze full template duration for CBC events.

9.14 0.5.2 (2019-04-15)

- Fix typo in `gracedb.get_instruments`: there was the attribute lookup `single.ifo`, which should have been the dictionary lookup `single[ifo]`.
- Fix `gwcelery.tasks.p_astro_other.choose_snr` for gstlal. This method did not previously expect to be called for gstlal, since it is typically only invoked for other pipelines. However, there is one case when `choose_snr` is invoked for gstlal, which is when the ranking_data file from gstlal is corrupted with NaNs, causing P_astro for gstlal to fail. Thus, choose_snr has now been fixed to also handle gstlal as a pipeline.

9.15 0.5.1 (2019-04-12)

- Changed default for em-bright from 2.83 to 3.0 M_sun to be consistent with notices.

9.16 0.5.0 (2019-04-12)

- Give permissions to read the files under parameter estimation run directories to non-owner people so that rota people can check their progresses. The naming convention of the run directories changed.
- EM-Bright ML classification requires review. Until then, give answer based on low-latency estimates.
- Compute P_astro with mass-based template weighting. Template weights are now keyed on template parameters, rather than bin numbers. This should make P_astro immune to binning conventions.
- Add form to manually send a preliminary GCN Notice.
- Fix a typo in `gwcelery.sub` that caused the Flower dashboard to fail to start.
- Round iDQ p(glitch) to 3 decimal places in GraceDB log message.
- Switch log telemetry from the on-premise instance of Sentry at Caltech to a cloud-hosted subscription to sentry.io.
- In the playground configuration, the `gwcelery.tasks.gcn.validate` task was producing false alarms because the GCN receiver was receiving VOEvents from the production instance, which would certainly differ in content from VOEvents in the playground instance. Fix this by having `gwcelery.tasks.gcn.validate` discard all VOEvents if the VOEvent broadcaster is disabled.
- Update ligo-followup-advocate to 0.0.27.
- Wait for 1 minute before parameter estimation in case the preferred event is updated with high latency.
- Ensure that P_astro accounts for very loud MBTA and PyCBC events, whose FAR saturate at certain low values depending on instrument combination, but whose SNRs can increase indefinitely.
- When a user triggers a Preliminary or Update alert through the Flask interface, create a GraceDB log message to record the username.
- The Flask interface will now show a confirmation dialog before sending any alerts.
- Add a terrifying warning to the Flask interface to make it clear that the interface is live.

9.17 0.4.3 (2019-04-05)

- Now that LIGO/Virgo alerts are public, switch the GCN listener that we use to confirm receipt of our own GCN Notices from a managed, private connection to an anonymous, public connection.
- Migrate the Flask and Flower dashboards from ldas-jobs.ligo.caltech.edu to emfollow.ligo.caltech.edu. The new URLs are:
  - https://emfollow.ligo.caltech.edu/gwcelery
  - https://emfollow.ligo.caltech.edu/flower
  - https://emfollow.ligo.caltech.edu/playground/gwcelery
  - https://emfollow.ligo.caltech.edu/playground/flower

  Remove the htaccess file from our public_html directory, since the reverse proxy configuration is now the responsibility of system administrators.
- Display the GWCelery version number in the Flask application.
- Add visualizations for `p_astro.json` source classification files.

9.18 0.4.2 (2019-04-05)

- Calculation of number of instruments is now unified across superevent manager and orchestrator using gracedb method `get_number_of_instruments`.
- Enable automated preliminary alerts for all pipelines because disabling them in the orchestrator introduced some issues due to the criteria for releasing a public alert drifting away from the definition of a the preferred event of a superevent. We will instead trust pipelines that are still under review will upload events to the playground rather than the production environment.

9.19 0.4.1 (2019-04-02)

- Fixed normalization issues with p_astro_gstlal.py; normalization was being applied in the wrong places during Bayes factor computation.
- Require celery < 4.3.0 because that version breaks the nagios unit tests.
- Update false alarm rate trials factors for preliminary alerts.
- Enable sending GCN notices for fully automated preliminary alerts.
- Add threshold_snr option in online_pe.jinja2, which is used to determine the upper limit of distance prior.
- Use the same criteria to decide whether to expose an event publicly in GraceDB as we use to decide whether to issue a public alert.
- Do not issue public alerts for single-instrument GW events.
- Disable automated preliminary alerts for all pipelines but gstlal and cWB due to outstanding review items for the other pipelines.

9.20 0.4.0 (2019-03-29)

- This is the penultimate release before LIGO/Virgo observing run 3 (O3).
- Make detchar results easier to read by formatting as HTML table.
- Allow iDQ to label DQV onto events based on p(glitch). Adjustable by pipeline.
- Move functions in tasks/lalinference.py to lalinference_pipe.py in lalsuite.
- Take into account calibration errors in automatic Parameter Estimation.
- Do not use margphi option for automatic Parameter Estimation with ROQ waveform since that option is not compatible with ROQ likelihood.
- Adjust WSGI middleware configuration to adapt to a change in Werkzeug 0.15.0 that broke redirects on form submission in the Flask app. See https://github.com/pallets/werkzeug/pull/1303.
- Use the new `ligo.lw` module for reading gstlal's `ranking_data.psd.xml.gz` files, because these files are now written using the new LIGO-LW format that uses integer row IDs.
- Use clean data for parameter estimation.
- Use production accounting group for PE runs on gracedb events.
- Change threshold from log-likelihood equals 6 to a dynamic threshold that ensures that all gstlal events uploaded to gracedb get assigned a P_astro value.

9.21 0.3.1 (2019-03-18)

- Fix a bug in translating keys from `source_classification.json` to keyword arguments for `GraceDB.createVOEvent` that caused VOEvents to be missing the `HasNS` and `HasRemnant` fields.
- FAR threshold for sending preliminary notices for CBC is changed to 1 per 2 months.
- Upload log files when LALInference parameter estimation jobs fail or are aborted.
- Changed the filename `source_classification.json` to `em_bright.json`.
- Change condor log directory from /var/tmp to ~/.cache/condor since gwcelery workers have separate /var/tmp when they are running as condor jobs and that causes problems when gwcelery tries to read log files.
- Limit the maximum version of gwpy to 0.14.0 in order to work around a unit test failure that started with gwpy 0.14.1. See https://git.ligo.org/emfollow/gwcelery/issues/95.
- Upload a diff whenever a LIGO/Virgo VOEvent that we receive from GCN does not match the original that we sent.
- Wait for low-latency or high-latency frame files being transferred to the cluster before parameter estimation starts.

9.22 0.3.0 (2019-03-01)

- Fixed exponent in the expression of foreground count in p_astro_other task.
- Run the sky map postprocessing and add the `PE_READY` tag when LALInference finishes.
- Include `EM_COINC` triggered circulars to upload to the superevent page.
- p-astro reads mean values from a file on CIT, new mass-gap category added. Removed redundant functions from p_astro_gstlal module.
- Continuous deployment on the Caltech cluster now uses a robot keytab and `gsissh` instead of SSH keys and vanilla `ssh` because the new my.ligo.org SSH key management does not support scripted access.
- Improve the isolation between the production and playground instances of GWCelery by deploying them under two separate user accounts on the Caltech cluster.
- Add functionality for em_bright task to query `emfollow/data` for trained machine learning classifier and report probabilities based on it.

9.23 0.2.6 (2019-02-12)

- Report an environment tag to Sentry corresponding to the GWCelery configuration module (`production`, `test`, `playground`, or `development`) in order to differentiate log messages from different deployments.
- The `gwcelery condor` command now identifies jobs that it owns by matching both the job batch name and the working directory. This makes it possible to run multiple isolated instances of GWCelery under HTCondor on the same cluster in different working directories.
- Change the conditions for starting parameter estimation. For every CBC superevent, create an `online_pe.ini` file suitable for starting LALInference. However, only start LALInference if the false alarm rate is less than once per 2 weeks.
- Determine PSD segment length for LALInference automatically based on data availability and data quality.

- Add a Flask-based web interface for manually triggering certain tasks such as sending updated GCN notices.

9.24 0.2.5 (2019-02-01)

- Pass along the GWCelery version number to Sentry.
- Upload stdout and stderr when dag creation fails and notifications when submitted job fails in Parameter Estimation
- Allow detchar module's `create_cache` to use gwdatafind when frames are no longer in llhoft.
- The Nagios monitoring plugin will now report on the status of LVAlert subscriptions.
- Change trials factor to 5 for both CBC and Burst categories. CBC includes the 4 CBC pipelines. Burst includes the 4 searches performed in total by the 2 Burst pipelines. An additional external coincidence search.
- Automatically set up PE ini file depending on source parameters reported by detection pipelines.

9.25 0.2.4 (2018-12-17)

- Fix broken links in log messages due to changes in GraceDB URL routes.
- Whenever we send a public VOEvent using GCN, also make the corresponding VOEvent file in GraceDB public.
- Don't include Mollweide projection PNG file in VOEvents. The sky map visualizations take longer to generate than the FITS files themselves, so they were unnecessarily slowing down the preliminary alerts.
- Preliminary GCN FAR threshold is modified to be group (CBC, Burst, Test) specific.

9.26 0.2.3 (2018-12-16)

- Update frame type used in LALInference Parameter Estimation.
- Handle cases where `p_astro_gstlal.compute_p_astro` returns NaNs by falling back to `p_astro_other.compute_p_astro`.
- Fix a bug that prevented annotations that are specific to 3D sky maps from being performed for multi-resolution FITS files.
- Fetch the graceid for the new event added from the gracedb logs since superevent packet does not provide information as to which event is added in case of type event_added.

9.27 0.2.2 (2018-12-14)

- Add error handling for nonexistent iDQ frames in detchar module.

9.28 0.2.1 (2018-12-14)

- Update detchar module configuration for ER13.

9.29 0.2.0 (2018-12-14)

- This is the release of GWCelery for ER13.
- Run two separate instances of Comet, one to act as a broker and one to act as a client. This breaks a cycle that would cause retransmission of GRB notices back to GCN.
- Fix a race condition that could cause preliminary alerts to be sent out for events for which data quality checks had failed.
- Unpin the `redis` package version because recent updates to Kombu and Billiard seem to have fixed the Nagios unit tests.
- Start the Comet VOEvent broker as a subprocess intead of using `multiprocessing` and go back to using PyGCN instead of Comet as the VOEvent client. This is a workaround for suspected instability due to a bad interaction between `redis-py` and `multiprocessing`.
- Reset Matplotlib's style before running `ligo-skymap-plot` and `ligo-skymap-plot-volume`. There is some other module (probably in LALSuite) that is messing with the rcparams at module scope, which was causing Mollweide plots to come out with unusual aspect ratios.
- Run `check_vectors` upon addition of an event to a superevent if the superevent already has an `DQV` label.
- Do not check the DMT-DQ_VECTOR for pipelines which use gated h(t).
- Remove static example VOEvents from the Open Alert Users Guide. We never used them because activating sample alerts got help until ER13.
- Disable running the Orchestrator for test events for ER13. After ER13 is over, we need to carefully audit the code and make sure that test events are handled appropriately.
- Enable public GraceDB entries and public GCNs for mock (MDC) events. For **real** events in ER13, disable public preliminary GCNs. Instead, advocate signoffs will trigger making events and GCN notices public: `ADVOK` for initial notices and `ADVNO` for retraction notices.
- Include source classification output (BNS/NSBH/BBH/Terrestrial) in GCN Notices.

9.30 0.1.7 (2018-11-27)

- Pin the `redis` package version at <3 because the latest version of redis breaks the Nagios unit tests.
- Ditch our own homebrew VOEvent broker and use Comet instead.
- In addition to traditional flat, fixed-nside sky maps, BAYESTAR will now also upload an experimental multiresolution format described in LIGO-G1800186-v4.

9.31 0.1.6 (2018-11-14)

- Update URL for static example event.

9.32 0.1.5 (2018-11-13)

- Add tasks for submitting HTCondor DAGs.

- Add a new module, `gwcelery.tasks.lalinference`, which provides tasks to start parameter estimation with LALInference and upload the results to GraceDB.
- Depend on lalsuite nightly build from 2018-11-04 to pick up changes to LALInference for Python 3 support.
- Send static example VOEvents from the Open Alert Users Guide. This will provide a stream of example alerts for astronomers until GraceDB is ready for public access.
- Add trials factor correction to the event FAR when comparing against FAR threshold to send out preliminary GCN.
- Require that LIGO/Virgo VOEvents that we receive from GCN match the original VOEvents from GraceDB byte-for-byte, since GCN will now pass through our VOEvents without modification.

9.33 0.1.4 (2018-10-29)

- Work around a bug in astropy.visualization.wcsaxes that affected all-sky plots when Matplotlib's `text.usetex` rcparam is set to `True` (https://github.com/astropy/astropy/issues/8004). This bug has evidently been present since at least astropy 1.3, but was not being triggered until recently: it is likely that some other package that we import (e.g. lalsuite) is now globally setting `text.usetex` to `True`.
- A try except is added around updateSuperevent to handle a bad request error from server side when updating superevent parameters which have nearby values.
- Send automatic preliminary alerts only for events with a false alarm rate below a maximum value specified by a new configuration variable, `preliminary_alert_far_threshold`.
- State vector vetoes will not suppress processing of preliminary sky maps and source classification. They will still suppress sending preliminary alerts.
- Set `open_alert` to `True` for all automated VOEvents.

9.34 0.1.3 (2018-10-26)

- Preliminary GCN is not sent for superevents created from offline gw events.
- Add `dqr_json` function to `gwcelery.tasks.detchar`, which uploads a DQR-compatible json to GraceDB with the results of the detchar checks.
- Depend on ligo.skymap >= 0.0.17.
- Fix a bug in sending initial, update, and retraction GCN notices: we were sending the VOEvent filenames instead of the file contents.

9.35 0.1.2 (2018-10-11)

- Setted `vetted` flag to true for all initial, update, and retraction alerts that are triggered by GraceDB signoffs.
- Write GraceDB signoffs, instead of just labels, to simulate initial and retraction alerts for mock events, because merely creating the `ADVNO` or `ADVOK` label does not cause GraceDB to erase the `ADVREQ` label. This change makes mock alerts more realistic.
- Change filename of cWB sky maps from `skyprobcc_cWB.fits` to `cWB.fits.gz` for consistency with other pipelines.

- Any time that we send a VOEvent, first change the GraceDB permissions on the corresponding superevent so that it is visible to the public. Note that this has no effect during the ongoing software engineering runs because LVEM and unauthenticated access are currently disabled in GraceDB.

9.36 0.1.1 (2018-10-04)

- Use the `public` tag instead of the `lvem` tag to mark preliminary sky maps for public access rather than LV-EM partner access. Note that GraceDB has not yet actually implemented unauthenticated access, so this should have no effect during our ongoing software engineering runs.
- Add `check_idq` function to detchar module, which reads probabilities generated by iDQ.
- Automated `DQV` labels should not trigger retraction notices because they prevent preliminary notices from being sent in the first place.
- The criterion for selecting a superevent's preferred event now prefers multiple-detector events to single-detector events, with precedence over source type (CBC versus burst). Any remaining tie is broken by using SNR for CBC and FAR for Burst triggers.
- By default, initial and update alerts will find and send the most recently added public sky map.
- The initial and update sky maps no longer perform sky map annotations, because they would only be duplicating the annotations performed as part of the preliminary alert.
- Mock events now include example initial and retraction notices. Two minutes after each mock event is uploaded, there will be either an `ADVOK` or an `ADVNO` label applied at random, triggering either an initial or a retraction notice respectively.
- Depend on ligo-gracedb >= 2.0.1 in order to pull in a bug fix for VOEvents with ProbHasNS or ProbHasRemnant set to 0.0.
- Use the `sentry-sdk` package instead of the deprecated `raven` package for Sentry integration.

9.37 0.1.0 (2018-09-26)

- Separated the external GCN listening handlers into two: one that listens to GCNs about SNEWS triggers and another that listens to Fermi and Swift.
- Fixed calls to the raven temporal coincidence search so that search results separate SNEWS triggers from Fermi and Swift.
- Add space-time FAR calculation for GRB and GW superevent coincidences. This only runs when skymaps from both triggers are available to download.
- Add human vetting for initial GCN notices. For each new superevent that passes state vector checks, the `ADVREQ` label is applied. Rapid response team users should set their GraceDB notification preferences to alert them on `ADVREQ` labels. If a user sets the `ADVOK` label, then an initial notice is issued. If a user sets the `ADVNO` label, then a retraction notice is issued.
- Update the LVAlert host for gracedb-playground.ligo.org.
- Add experimental integration with Sentry for log aggregation and error reporting.
- Track API and LVAlert schema changes in ligo-gracedb 2.0.0.

9.38 0.0.31 (2018-09-04)

- Refactor external trigger handling to separate it from the orchestrator.
- Fixed a bug in the VOEvent broker to only issue "iamalive" messages after sending the first VOEvent.
- Pass group argument to set time windows appropriately when performing raven coincidence searches. Search in the [-600, 60]s range and [-5, 1]s range around external triggers for Burst events and CBC events respectively. Similarly, search in the [-60, 600]s and [-1, 5]s range around Burst and CBC events for external triggers.
- Compute and upload FAR for GRB external trigger/superevent coincidence upon receipt of the EM_COINC label application to a superevent.
- Add continuous integration testing for Python 3.7, and run test suite against all supported Python versions (3.6, 3.7).
- Update ligo.skymap to 0.0.15.

9.39 0.0.30 (2018-08-02)

- Manage superevents for production, test, and MDC events separately.
- Add some more validation of LIGO/Virgo VOEvents from GCN.
- Remove now-unused task `gwcelery.tasks.orchestartor.continue_if`.
- Add `check_vectors` run for external triggers.
- Change the preferred event selection criteria for burst events to be FAR instead of SNR.
- Add `gwcelery nagios` subcommand for Nagios monitoring.
- Incorporate Virgo DQ veto streams into `check_vectors`
- Update ligo-raven to 1.3 and ligo-followup-advocate to 0.0.11.

9.40 0.0.29 (2018-07-31)

- Add a workflow graph to superevents module documentation.
- Add `gwcelery condor resubmit` as a shortcut for `gwcelery condor rm; gwcelery condor submit`.
- Fix deprecation warning due to renaming of `ligo.gracedb.rest.Gracedb.createTag` to `ligo.gracedb.rest.Gracedb.addTag`.
- Update ligo-gracedb to 2.0.0.dev1.

9.41 0.0.28 (2018-07-25)

- Add injection checks to `check_vector`.
- Bitmasks are now defined symbolically in `detchar`.
- Refactor configuration so that it is possible to customize settings through an environment variable.

9.42 0.0.27 (2018-07-22)

- The preferred event for superevents is now decided based on higher SNR value instead of lower FAR in the case of a tie between groups.
- A check for the existence of the gstlal trigger database is performed so that compute_p_astro does not return None.

9.43 0.0.26 (2018-07-20)

- Fix spelling of the label that is applied to events after p_astro finishes, changed from `P_ASTRO_READY` to `PASTRO_READY`.
- Run p_astro calculation for mock events.
- Overhaul preliminary alert pipeline so that it is mostly feature complete for both CBC and Burst events, and uses a common code path for both types. Sky map annotations now occur for both CBC and Burst localizations.
- Switch to using the pre-registered port 8096 for receiving proprietary LIGO/Virgo alerts on emfollow.ligo.caltech.edu. This means that the capability to receive GCNs requires setting up a site configuration in advance with Scott Barthelmey.

  Once we switch to sending public alerts exclusively, then we can switch back to using port 8099 for anonymous access, requiring no prior site configuration.

9.44 0.0.25 (2018-07-19)

- Reintroduce pipeline-dependent pre/post peeks for `check_vector` after fixing issue where pipeline information was being looked for in the wrong dictionary.
- `check_vector` checks all detectors regardless of instruments used, but only appends labels based on active instruments.
- Fix a few issues in the GCN broker:
  - Decrease the frequency of keepalive ("iamalive" in VOEvent Transport Protocol parlance) packets from once a second to once a minute at the request of Scott Barthelmey.
  - Fix a possible race condition that might have caused queued VOEvents to be thrown away unsent shortly after a scheduled keepalive packet.
  - Consume and ignore all keepalive and ack packets from the client so that the receive buffer does not overrun.
- Add `p_astro` computation for `gstlal` pipeline. The copmutation is launched for all cbc_gstlal triggers.

9.45 0.0.24 (2018-07-18)

- Revert pipeline-dependent pre/post peeks for `check_vector` because they introduced a regression: it caused the orchestrator failed without running any annotations.

9.46 0.0.23 (2018-07-18)

- Add timeout and keepalive messages to GCN broker.
- Update ligo-gracedb to 2.0.0.dev0 and ligo.skymap to 0.0.12.
- Add superevent duration for gstlal-spiir pipeline.
- Fix fallback for determining superevent duration for unknown pipelines.
- Make `check_vector` pre/post peeks pipeline dependent.

9.47 0.0.22 (2018-07-11)

- Process gstlal-spiir events.
- Create combined LVC-Fermi skymap in case of coincident triggers and upload to GraceDB superevent page. Also upload the original external trigger sky map to the external trigger GraceDB page.
- Generalize conditional processing of complex canvases by replacing the `continue_if_group_is()` task with a more general task that can be used like `continue_if(group='CBC')`.
- Add a `check_vector_prepost` configuration variable to control how much padding is added around an event for querying the state vector time series.

  This should have the beneficial side effect of fixing some crashes for burst events, for which the bare duration of the superevent segment was less than one sample.

9.48 0.0.21 (2018-07-10)

- MBTA events in GraceDB leave the `search` field blank. Work around this in `gwcelery.tasks.detchar.check_vectors` where we expected the field to be present.
- Track change in GraceDB JSON response for VOEvent creation.

9.49 0.0.20 (2018-07-09)

- After fixing some minor bugs in code that had not yet been tested live, sending VOEvents to GCN now works.

9.50 0.0.19 (2018-07-09)

- Rewrite the GCN broker so that it does not require a dedicated worker.
- Send VOEvents for preliminary alerts to GCN.
- Only perform state vector checks for detectors that were online, according to the preferred event.
- Exclude mock data challenge events from state vector checks.

9.51 0.0.18 (2018-07-06)

- Add detector state vector checks to the preliminary alert workflow.

9.52 0.0.17 (2018-07-05)

- Undo accidental configuration change in last version.

9.53 0.0.16 (2018-07-05)

- Stop listening for three unnecessary GCN notice types: `SWIFT_BAT_ALARM_LONG`, `SWIFT_BAT_ALARM_SHORT`, and `SWIFT_BAT_KNOWN_SRC`.
- Switch to SleekXMPP for the LVAlert client, instead of PyXMPP2. Because SleekXMPP has first-class support for publish-subscribe, the LVAlert listener can now automatically subscribe to all LVAlert nodes for which our code has handlers. Most of the client code now lives in a new external package, sleek-lvalert.

9.54 0.0.15 (2018-06-29)

- Change superevent threshold and mock event rate to once per hour.
- Add `gracedb.create_label` task.
- Always upload external triggers to the 'External' group.
- Add rudimentary burst event workflow to orchestrator: it just generates VOEvents and circulars.
- Create a label in GraceDB whenever `em_bright` or `bayestar` completes.

9.55 0.0.14 (2018-06-28)

- Fix typo that was causing a task to fail.
- Decrease orchestrator timeout to 15 seconds.

9.56 0.0.13 (2018-06-28)

- Change FAR threshold for creation of superevents to 1 per day.
- Update ligo-followup-advocate to >= 0.0.10. Re-enable automatic generation of GCN circulars.
- Add "EM bright" classification. This is rudimentary and based only on the point mass estimates from the search pipeline because some of the EM bright classifier's dependencies are not yet ready for Python 3.
- Added logic to select CBC events as preferred event over Burst. FAR acts as tie breaker when groups for preferred event and new event match.
- BAYESTAR now adds GraceDB URLs of events to FITS headers.

9.57 0.0.12 (2018-06-28)

- Prevent receiving duplicate copies of LVAlert messages by unregistering redundant LVAlert message types.
- Update to ligo-followup-advocate >= 0.0.9 to update GCN Circular text for superevents. Unfortunately, circulars are still disabled due to a regression in ligo-gracedb (see https://git.ligo.org/lscsoft/gracedb-client/issues/7).
- Upload BAYESTAR sky maps and annotations to superevents.
- Create (but do not send) preliminary VOEvents for all superevents. No vetting is performed yet.

9.58 0.0.11 (2018-06-27)

- Submit handler tasks to Celery as a single group.
- Retry GraceDB tasks that raise a `TimeoutError` exception.
- The superevent handler now skips LVAlert messages that do not affect the false alarm rate of an event (e.g. simple log messages).

  (Note that the false alarm rate in GraceDB is set by the initial event upload and can be updated by replacing the event; however replacing the event does not produce an LVAlert message at all, so there is no way to intercept it.)
- Added a query kwarg to superevents method to reduce latency in fetching the superevents from gracedb.
- Refactored getting event information for update type events so that gracedb is polled only once to get the information needed for superevent manager.
- Renamed the `set_preferred_event` task in gracedb.py to `update_superevent` to be a full wrapper around the `updateSuperevent` client function. Now it can be used to set preferred event and also update superevent time windows.
- Many `cwb` (extra) attributes, which should be floating point numbers, are present in lvalert packet as strings. Casting them to avoid embarassing TypeErrors.
- Reverted back the typecasting of far, gpstime into float. This is fixed in https://git.ligo.org/lscsoft/gracedb/issues/10
- CBC `t_start` and `t_end` values are changed to 1 sec interval.
- Added ligo-raven to run on external trigger and superevent creation lvalerts to search for coincidences. In case of coincidence, EM_COINC label is applied to the superevent and external trigger page and the external trigger is added to the list of em_events in superevent object dictionary.
- `cwb` and `lib` nodes added to superevent handler.
- Events are treated as finite segment window, initial superevent creation with preferred event window. Addition of events to superevents may change the superevent window and also the preferred event.
- Change default GraceDB server to https://gracedb-playground.ligo.org/ for open public alert challenge.
- Update to ligo-gracedb >= 1.29dev1.
- Rename the `get_superevent` task to `get_superevents` and add a new `get_superevent` task that is a trivial wrapper around `ligo.gracedb.rest.GraceDb.superevent()`.

9.59 0.0.10 (2018-06-13)

- Model the time extent of events and superevents using the `glue.segments` module.
- Replace GraceDB.get with GraceDB.superevents from the recent dev release of gracedb-client.
- Fix possible false positive matches between GCNs for unrelated GRBs by matching on both TrigID (which is generally the mission elapsed time) and mission name.
- Add the configuration variable `superevent_far_threshold` to limit the maximum false alarm rate of events that are included in superevents.
- LVAlert handlers are now passed the actual alert data structure rather than the JSON text, so handlers are no longer responsible for calling `json.loads`. It is a little bit more convenient and possibly also faster for Celery to deserialize the alert messages.
- Introduce `Production`, `Development`, `Test`, and `Playground` application configuration objects in order to facilitate quickly switching between GraceDB servers.
- Pipeline specific start and end times for superevent segments. These values are controlled via configuration variables.

9.60 0.0.9 (2018-06-06)

- Add missing LVAlert message types to superevent handler.

9.61 0.0.8 (2018-06-06)

- Add some logging to the GCN and LVAlert dispatch code in order to diagnose missed messages.

9.62 0.0.7 (2018-05-31)

- Ingest Swift, Fermi, and SNEWS GCN notices and save them in GraceDB.
- Depend on the pre-release version of the GraceDB client, ligo-gracedb 1.29.dev0, because this is the only version that supports superevents at the moment.

9.63 0.0.6 (2018-05-26)

- Generate GCN Circular drafts using ligo-followup-advocate.
- In the continuous integration pipeline, validate PEP8 naming conventions using pep8-naming.
- Add instructions for measuring test coverage and running the linter locally to the contributing guide.
- Rename `gwcelery.tasks.voevent` to `gwcelery.tasks.gcn` to make it clear that this submodule contains functionality related to GCN notices, rather than VOEvents in general.
- Rename `gwcelery.tasks.dispatch` to `gwcelery.tasks.orchestrator` to make it clear that this module encapsulates the behavior associated with the "orchestrator" in the O3 low-latency design document.
- Mock up calls to BAYESTAR in test suite to speed it up.

- Unify dispatch of LVAlert and GCN messages using decorators. GCN notice handlers are declared like this:

```
import lxml.etree
from gwcelery.tasks import gcn

@gcn.handler(gcn.NoticeType.FERMI_GBM_GND_POS,
             gcn.NoticeType.FERMI_GBM_FIN_POS)
def handle_fermi(payload):
    root = lxml.etree.fromstring(payload)
    # do work here...
```

  LVAlert message handlers are declared like this:

```
import json
from gwcelery.tasks import lvalert

@lvalert.handler('cbc_gstlal',
                 'cbc_pycbc',
                 'cbc_mbta')
def handle_cbc(alert_content):
    alert = json.loads(alert_content)
    # do work here...
```

- Instead of carrying around the GraceDB service URL in tasks, store the GraceDB host name in the Celery application config.
- Create superevents by simple clustering in time. Currently this is only supported by the `gracedb-dev1` host.

9.64 0.0.5 (2018-05-08)

- Disable socket access during most unit tests. This adds some extra assurance that we don't accidentally interact with production servers during the unit tests.
- Ignore BAYESTAR jobs that raise a `DetectorDisabled` error. These exceptions are used for control flow and do not constitute a real error. Ignoring these jobs avoids polluting logs and the Flower monitor.

9.65 0.0.4 (2018-04-28)

- FITS history and comment entries are now displayed in a monospaced font.
- Adjust error reporting for some tasks.
- Depend on newer version of `ligo.skymap`.
- Add unit tests for the `gwcelery condor submit` subcommand.

9.66 0.0.3 (2018-04-27)

- Fix some compatibility issues between the `gwcelery condor submit` subcommand and the format of `condor_q -totals -xml` with older versions of HTCondor.

9.67 0.0.2 (2018-04-27)

- Add `gwcelery condor submit` and related subcommands as shortcuts for managing GWCelery running under HTCondor.

9.68 0.0.1 (2018-04-27)

- This is the initial release. It provides rapid sky localization with BAYESTAR, sky map annotation, and sending mock alerts.
- By default, GWCelery is configured to listen to the test LVAlert server.
- Sending VOEvents to GCN/TAN is disabled for now.

CHAPTER 10

License

The GWCelery Logo is a composite of Celery2 by Tiia Monto and Lorentzian Wormhole by Kes47 from Wikimedia Commons (CC BY-SA 3.0).

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.
```

(continues on next page)

(continued from previous page)

```
  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.
```

(continues on next page)

(continued from previous page)

```
  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
```

(continues on next page)

(continued from previous page)

```
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
```

(continues on next page)

(continued from previous page)

```
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation.  If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission.  For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this.  Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

                            NO WARRANTY
```

(continues on next page)

(continued from previous page)

```
  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
```

(continues on next page)

(continued from previous page)

```
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.
```

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

GWCelery is open source and is licensed under the *GNU General Public License v2 or later*.

Bibliography

[LIGO] https://wiki.ligo.org/Calibration/TDCalibReview

[Virgo] https://dcc.ligo.org/G1801125/

[DMT] https://wiki.ligo.org/DetChar/DmtDqVector

[GCN] https://gcn.gsfc.nasa.gov

Python Module Index

g

Index

H

I

J

K

L

T

U

V

W