
Low-Latency Algorithm for Multi-messenger Astrophysics (LLAMA)

Release 3.4.16

Stefan Trklja Countryman

May 24, 2021

INFO FOR REVIEWERS

1	Introduction for Reviewers	3
2	Purpose	5
3	Documentation	7
3.1	Software Documentation	7
3.2	Review Wikis	7
3.3	Papers	7
3.4	Source Code	7
4	Testing on Review Server	9
4.1	Fake cases	9
4.2	Running Fake Cases	9
5	Contents of the Output Folder	11
5.1	Inputs	11
5.2	Main Outputs	11
5.3	Auxilliary Outputs	11
5.4	Real cases	12
6	Using the Docker Images	13
6.1	Installing Docker	13
6.2	Docker Cloud	14
6.3	Getting LLAMA Images	15
6.3.1	Choosing the Image	15
6.3.2	Getting/Updating the Image	15
6.3.3	Removing Images	16
6.4	Running a LLAMA Container	16
6.5	Mounting Directories	17
6.5.1	Mounting in MacOS/Linux	17
6.5.2	Mounting in Windows	17
6.6	Out of Memory/Disk	20
7	Running on Habanero	21
7.1	Interactive Habanero Jobs	21
7.2	Docker Hub Authentication with Singularity	22
7.3	Fixing Singularity Cache on Habanero	22
7.4	Load Singularity Module	23
7.5	Pull LLAMA Image	23
8	Local Installation	25

8.1	System Requirements	25
8.2	Installing Conda	25
9	Setting Up a Production Server	27
9.1	Install Docker	27
9.2	Install Docker Compose	27
9.3	Log in to Docker Cloud	28
9.4	Get <i>docker-compose.yml</i>	28
9.5	Starting LLAMA Production App	28
10	Introduction	29
11	How Data is Organized	31
11.1	Event Directories	31
11.2	GCN Notice Archive	31
11.3	Log Files	31
12	Running the Pipeline Automatically	33
13	Running the Pipeline Manually	35
13.1	Using Defaults with <i>skymap_info</i>	35
14	Sensitivity and Background Studies (O3B)	37
15	Sensitivity and Background Studies (pre-O3B)	39
15.1	Editing your <i>.bashrc</i>	40
15.1.1	Using the DigitalOcean API	40
15.2	LLAMA Scripts	40
15.3	Adding Your SSH Keys to the Droplets	40
15.4	Installing Requirements	41
15.5	DigitalOcean Administration Examples	41
15.5.1	Preparing a Server	42
15.6	Server Preparation Overview	42
15.7	Moving Data into Place	43
15.7.1	Running the Analysis in Parallel	45
15.8	Starting the Servers	45
15.9	Running the Analysis	46
15.10	Collecting Results	47
16	Developer Installation	49
16.1	Obtaining the LLAMA Source Code	49
16.2	Using the Docker Dev Image	50
16.2.1	List <i>bin/docker</i> Commands	50
16.2.2	Getting the Latest Development Image	51
16.2.3	Starting a Dev Container	51
16.2.4	Sketch of a Bug Fix	52
16.2.5	Destroying your Dev Container	52
17	Previous LLAMA Versions	55
17.1	Developer Installation (Pre-O3)	55
17.1.1	System Requirements	55
17.1.2	Introduction	55
17.1.3	Installing LLAMA Dependencies	56
17.1.4	Obtaining the LLAMA Software (pre-O3)	56
17.1.5	Setting up Configuration Files	57

17.1.6	Install git Hooks	58
17.1.7	Install LLAMA dependencies	58
17.1.8	Install LIGO Software	58
17.1.9	Install IceCube Offline Software	59
17.1.9.1	Installing IceCube Dependencies	60
17.1.9.2	Entering IceCube Credentials	60
17.1.10	Install MATLAB	62
17.2	Configuration and Authentication	63
17.2.1	Generate SSH Keys	63
17.2.2	GMail Authentication	63
17.2.3	LIGO Authentication	64
17.2.4	Setting Up SSL Certificates	64
17.2.5	Setting Up Passwords for Run Summary Pages	64
17.3	External Authentication	65
17.3.1	Bitbucket Authentication	65
17.3.2	git.ligo.org Authentication	65
17.3.3	Twilio Authentication	66
17.3.4	GW Astronomy Authentication	66
17.3.5	GCN Authentication	66
17.4	Turning on the Pipeline	66
17.4.1	Subscribing to LVAAlert Nodes	66
17.4.2	Starting Pipeline Daemons	67
17.4.2.1	Testing LVAAlert	68
17.4.3	Viewing Active Processes	68
17.4.4	Checking Pipeline Logs	68
17.4.5	Killing Pipeline Daemons	68
17.4.6	Deprecated: MATLAB Logs	69
17.5	Developing for LLAMA	69
17.5.1	Introduction to Development	69
17.5.1.1	Structure of the Pipeline	69
17.5.1.2	Adding Functionality	69
17.5.2	Developer Conventions and Best Practices	70
17.5.2.1	Data Format Conventions	70
17.5.2.2	Coding Best Practices	70
17.5.3	Testing	70
17.6	Appendix	71
17.6.1	Migrating to Conda	71
17.6.1.1	LIGO Wiki Documentation	71
17.6.1.2	Installing LIGO Software via Conda	71
17.6.2	Troubleshooting Installation	72
17.6.2.1	Creating a new user	72
17.6.2.2	Out of Memory	73
17.6.2.3	MATLAB Installation Troubleshooting	73
17.6.3	Install IceCube Offline Software with Root	73
17.6.4	Installing Ubuntu for Windows	76
17.6.5	Logging in to a Remote Server Using SSH	76
17.6.6	Getting LLAMA Software onto a Remote Server	76
17.6.7	SSH with X11 Forwarding	77
17.6.8	Using LLAMA	78
17.6.9	Documenting LLAMA	78
17.6.9.1	Overview	78
17.6.9.2	Publishing to gwen.com Website	78
17.6.9.3	Publishing Readme to Git Host	78
17.6.9.4	Publishing PDFs	79

17.6.9.5 Publishing HTML Web Pages	79
17.6.10 Troubleshooting LLAMA	79
17.6.11 Setting Up the Review Server	80
17.6.11.1 Provisioning the review server	80
17.6.11.2 Running the Review	80
17.6.12 Ideas for the Future	80
17.6.13 CVMFS	80
17.6.13.1 Advantages	81
17.6.13.2 Disadvantages	81
18 Academic Papers	83
18.1 Low-Latency Algorithm for Multi-messenger Astrophysics (LLAMA) with Gravitational-Wave and High-Energy Neutrino Candidates	83
18.2 Bayesian Multi-Messenger Search Method for Common Sources of Gravitational Waves and High-Energy Neutrinos	83
19 llama package	85
20 llama.batch package	93
21 llama.classes module	95
22 llama.cli module	99
23 llama.com package	105
23.1 llama.com.dl package	105
23.2 llama.com.do package	105
23.3 llama.com.email package	106
23.4 llama.com.gracedb package	106
23.5 llama.com.s3 package	112
23.6 llama.com.slack package	113
23.7 llama.com.utils module	114
24 llama.detectors module	117
24.1 Available Detectors	117
25 llama.dev package	119
25.1 llama.dev.background package	119
25.1.1 llama.dev.background.pvalue package	119
25.1.2 llama.dev.background.table package	120
25.1.3 llama.dev.background.table_singles package	120
25.2 llama.dev.clean package	120
25.3 llama.dev.data package	120
25.3.1 llama.dev.data.i3 package	120
25.4 llama.dev.docs package	121
25.4.1 llama.dev.docs.cli package	121
25.5 llama.dev.dv package	122
25.6 llama.dev.log package	123
25.6.1 llama.dev.log.lvalert package	123
25.7 llama.dev.upload package	123
26 llama.event package	127
27 llama.filehandler package	131
27.1 llama.filehandler.mixins module	140

28 llama.files package	143
28.1 llama.files.coinc_significance package	179
28.1.1 llama.files.coinc_significance.opa module	180
28.1.2 llama.files.coinc_significance.subthreshold module	185
28.1.3 llama.files.coinc_significance.utils module	192
28.2 llama.files.healpix package	194
28.2.1 llama.files.healpix.plotters module	198
28.2.2 llama.files.healpix.psf module	199
28.2.3 llama.files.healpix.skymap module	201
28.2.4 llama.files.healpix.utils module	206
28.3 llama.files.i3 package	211
28.3.1 llama.files.i3.json module	217
28.3.2 llama.files.i3.realtime_tools_stubs module	224
28.3.3 llama.files.i3.tex module	226
28.3.4 llama.files.i3.txt module	227
28.3.5 llama.files.i3.utils module	230
28.4 llama.files.lvc_skymap package	232
28.4.1 llama.files.lvc_skymap.utils module	234
28.5 llama.files.skymap_info package	235
28.5.1 llama.files.skymap_info.cli module	237
28.5.2 llama.files.skymap_info.utils module	238
28.6 llama.files.slack package	238
28.7 llama.files.advok module	241
28.8 llama.files.coinc_analyses module	242
28.9 llama.files.coinc_o2 module	248
28.10 llama.files.coinc_plots module	254
28.11 llama.files.fermi_grb module	269
28.12 llama.files.gen_draft_o2 module	271
28.13 llama.files.gracedb module	275
28.14 llama.files.gwastro module	279
28.15 llama.files.lvalert_advok module	282
28.16 llama.files.lvalert_json module	282
28.17 llama.files.lvc_gcn_xml module	283
28.18 llama.files.lvc_skymap_mat module	286
28.19 llama.files.lvc_skymap_txt module	287
28.20 llama.files.matlab module	287
28.21 llama.files.sms_receipts module	288
28.22 llama.files.team_receipts module	290
28.23 llama.files.timing_checks module	290
28.24 llama.files.uw_summary module	291
28.25 llama.files.ztf_trigger_list module	291
29 llama.flags package	293
29.1 llama.flags.cli module	294
30 llama.install package	297
30.1 llama.install.manifest module	297
31 llama.intent module	299
32 llama.listen package	301
32.1 llama.listen.gcn package	301
32.2 llama.listen.lvalert package	303
33 llama.lock module	305

34	llama.meta module	307
35	llama.pipeline module	309
35.1	Pipelines	309
36	llama.poll package	315
36.1	llama.poll.gracedb package	315
37	llama.run package	317
38	llama.serve package	323
38.1	llama.serve.gui package	323
38.1.1	llama.serve.gui.wsgi package	323
38.1.2	llama.serve.gui.domain module	324
38.2	llama.serve.jupyter package	324
38.2.1	llama.serve.jupyter.logs module	324
38.2.2	llama.serve.jupyter.utils module	324
39	llama.test package	325
39.1	llama.test.test_files package	325
39.2	llama.test.test_listeners package	325
39.2.1	llama.test.test_listeners.test_gcn module	325
39.3	llama.test.classes module	327
39.4	llama.test.test_bin module	329
39.5	llama.test.test_filehandler module	330
39.6	llama.test.test_pipeline module	334
39.7	llama.test.test_utils module	334
40	llama.utils module	337
41	llama.version module	347
42	llama.versioning module	349
43	llama.vetoes module	353
44	Performance Profiling	357
44.1	Combined Results	357
44.2	Unit Tests	357
44.3	Doctests	357
45	Source Code Plots	359
46	llama Command Line Interface	361
46.1	Named Arguments	361
46.2	subcommands (call one with ``-help`` for details on each)	362
47	llama batch	363
47.1	Named Arguments	363
47.2	choose pipeline (see ``llama.pipeline``)	364
47.3	logging settings	365
47.4	simulation configuration	366
48	llama com	369
48.1	Named Arguments	369
48.2	subcommands (call one with ``-help`` for details on each)	369

48.3	llama com do	370
48.3.1	Named Arguments	370
48.4	llama com gracedb	371
48.4.1	Positional Arguments	371
48.4.2	logging settings	372
48.5	llama com slack	372
48.5.1	Positional Arguments	372
48.5.2	Named Arguments	372
48.5.3	logging settings	373
49	llama dev	375
49.1	Named Arguments	375
49.2	subcommands (call one with ``--help`` for details on each)	375
49.3	llama dev background	376
49.3.1	Named Arguments	376
49.3.2	subcommands (call one with ``--help`` for details on each)	376
49.3.3	llama dev background pvalue	376
49.3.3.1	Named Arguments	377
49.3.4	llama dev background table	377
49.3.4.1	Positional Arguments	378
49.3.4.2	Named Arguments	378
49.4	llama dev clean	378
49.4.1	Named Arguments	378
49.4.2	filter runs and events (see: ``llama.run``)	379
49.4.3	logging settings	380
49.5	llama dev data	380
49.5.1	Named Arguments	380
49.5.2	subcommands (call one with ``--help`` for details on each)	380
49.5.3	llama dev data i3	381
49.5.3.1	Named Arguments	381
49.5.3.2	logging settings	381
49.6	llama dev docs	382
49.6.1	Named Arguments	382
49.6.2	subcommands (call one with ``--help`` for details on each)	382
49.6.3	llama dev docs cli	382
49.6.3.1	Positional Arguments	383
49.7	llama dev dv	383
49.7.1	Positional Arguments	383
49.7.2	Named Arguments	383
49.8	llama dev log	385
49.8.1	Named Arguments	385
49.8.2	subcommands (call one with ``--help`` for details on each)	385
49.8.3	llama dev log lvalert	385
49.8.3.1	Named Arguments	386
49.9	llama dev upload	386
49.9.1	Named Arguments	386
49.9.2	logging settings	387
50	llama event	389
50.1	Positional Arguments	389
50.2	choose pipeline (see ``llama.pipeline``)	389
51	llama files	391
51.1	Named Arguments	391

51.2	subcommands (call one with `` <code>--help</code> `` for details on each)	391
51.3	<code>llama files i3</code>	392
51.3.1	Named Arguments	392
51.3.2	output formats (specify at least 1)	393
52	<code>llama flags</code>	395
52.1	Named Arguments	395
52.2	filter runs and events (see: `` <code>llama.run</code> ``)	396
53	<code>llama install</code>	397
53.1	Named Arguments	397
53.2	logging settings	397
54	<code>llama listen</code>	399
54.1	Named Arguments	399
54.2	subcommands (call one with `` <code>--help</code> `` for details on each)	399
54.3	<code>llama listen gcn</code>	400
54.3.1	Named Arguments	400
54.3.2	logging settings	400
54.4	<code>llama listen lvalert</code>	401
54.4.1	Named Arguments	401
54.4.2	logging settings	401
55	<code>llama poll</code>	403
55.1	Named Arguments	403
55.2	subcommands (call one with `` <code>--help</code> `` for details on each)	403
55.3	<code>llama poll gracedb</code>	404
56	<code>llama run</code>	405
56.1	Named Arguments	405
56.2	choose pipeline (see `` <code>llama.pipeline</code> ``)	406
56.3	filter runs and events (see: `` <code>llama.run</code> ``)	408
56.4	logging settings	408
57	<code>llama serve</code>	411
57.1	Named Arguments	411
57.2	subcommands (call one with `` <code>--help</code> `` for details on each)	411
57.3	<code>llama serve gui</code>	412
57.3.1	Named Arguments	412
57.3.2	logging settings	412
57.4	<code>llama serve jupyter</code>	412
57.4.1	Named Arguments	413
57.4.2	logging settings	413
58	Indices and tables	415
	Python Module Index	417
	Index	419

LLAMA is a reliable and flexible multi-messenger astrophysics framework and search pipeline. It identifies astrophysical signals by combining observations from multiple types of astrophysical messengers and can either be run in online mode as a self-contained low-latency pipeline or in offline mode for post-hoc analyses. It was first used during Advanced LIGO's second observing run (O2) for the joint online LIGO/Virgo/IceCube Gravitational Wave/High-Energy Neutrino (GWHEN) search.

LLAMA's search algorithm has been upgraded to run during LIGO's third observing run (O3).

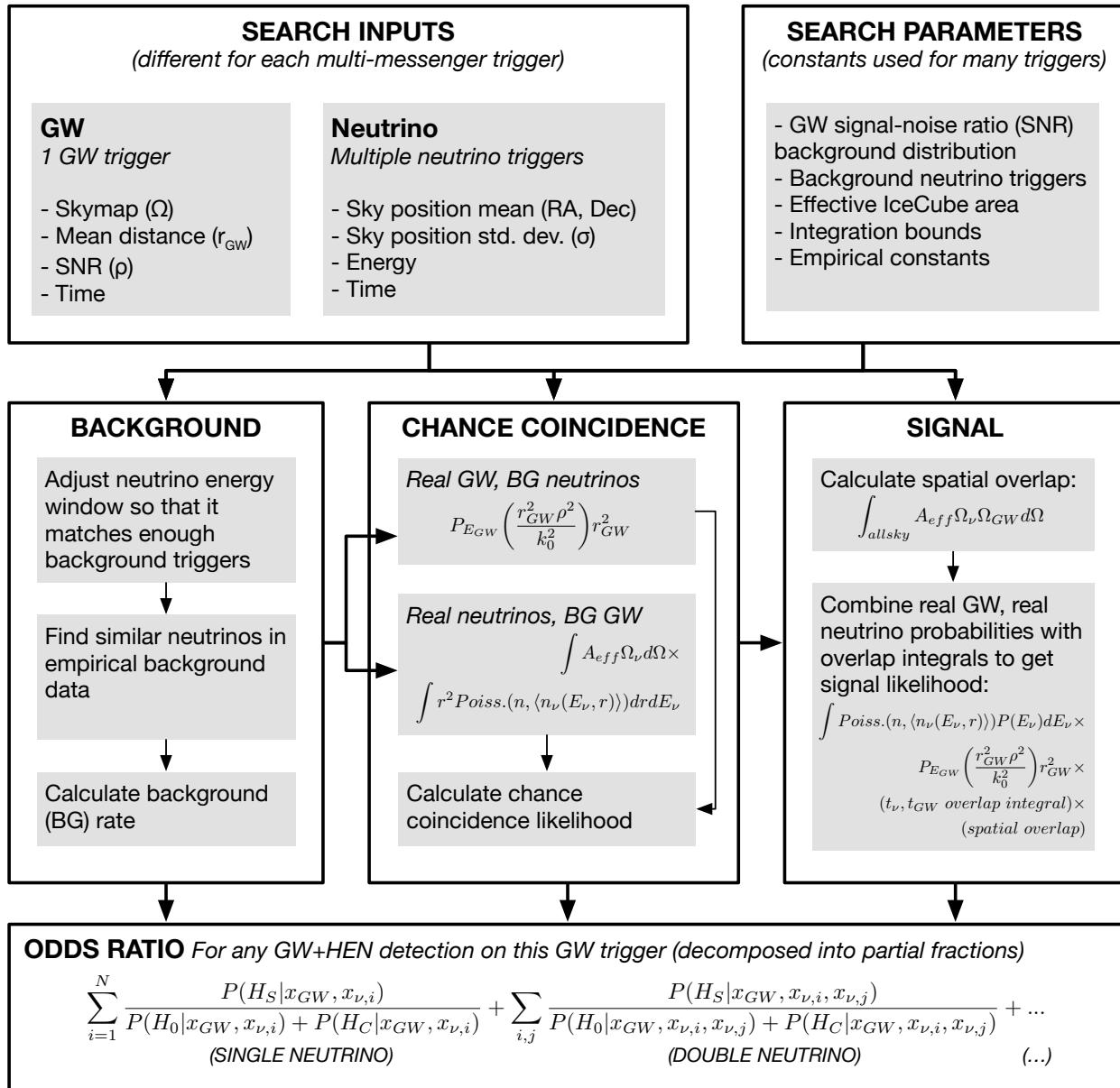


Fig. 1: Flowchart describing the significance calculation used in the O3 version of the pipeline.

The LLAMA team thanks the many researchers working in LIGO/Virgo, IceCube, and other astrophysics projects that enable this pipeline to operate successfully. In particular, they thank Scott Barthelmy and Leo Singer of NASA for providing helpful code and advice for working with GCN.

The authors are grateful to the IceCube Collaboration for providing neutrino datasets and support for testing this algorithm. The Columbia Experimental Gravity group is grateful for the generous support of Columbia University in

the City of New York and the National Science Foundation under grants PHY-1404462 and PHY-1708028. The authors are thankful for the generous support of the University of Florida and Columbia University in the City of New York.

**CHAPTER
ONE**

INTRODUCTION FOR REVIEWERS

This page describes how to test run the low-latency multi-messenger analysis pipeline. This pipeline currently focuses on gravitational wave (GW) and high-energy neutrino (HEN) searches. This pipeline listens GW events and upon a trigger initiates searching for HENs and calculates a coincidence significance for spatially and temporally correlated GW and HEN. It then creates a joint skymap for candidate multi-messenger neutrino events. If the significance is higher than the determined threshold for multi-messenger events it sends a multi-messenger event alert.

**CHAPTER
TWO**

PURPOSE

GWHEN-LLAMA (GWHEN search using Low-Latency Algorithm for Multi-messenger Astrophysics) is a low-latency GW+HEN analysis pipeline that ran during O2 (using previously reviewed offline analysis methods) with the goal of rapidly identifying joint GW+HEN sources. This review targets the upgrades made to the pipeline, the most major of which is an upgrade to the significance calculation algorithm (see documentation section below).

DOCUMENTATION

3.1 Software Documentation

The full software documentation for the pipeline is included in this document. An up-to-date version can always be found at gwhen.com¹. Additionally:

3.2 Review Wikis

Wikis are being maintained for [LIGO](#)² and [IceCube](#)³ reviewers. Detailed review-specific information can be found at these sites.

3.3 Papers

- The detailed technical structure of the pipeline can be found in [*this paper*](#).
- The coincidence calculation is based on [*a new proposed Bayesian methodology*](#).

3.4 Source Code

The complete source code can be accessed at [git.ligo.org](#)⁴, [bitbucket](#)⁵, and on [IceCube SVN](#)⁶ (authentication required in all cases; reviewers will need to use the git.ligo.org or IceCube SVN links).

¹ <http://gwhen.com>

² <https://wiki.ligo.org/GWHEN/WebHome>

³ <https://wiki.icecube.wisc.edu/index.php/GWHEN>

⁴ <https://git.ligo.org/stefco/multimessenger-pipeline>

⁵ <https://bitbucket.org/stefancountryman/multimessenger-pipeline/src>

⁶ <http://code.icecube.wisc.edu/svn/analyses/gwhen/>

TESTING ON REVIEW SERVER

First, SSH into the review server using the instructions on the [LIGO⁷](#) or [IceCube⁸](#) review wikis.

During the test the pipeline will be fully functional (except for sending alerts for high significance events and uploading data products, which it will not do in this testing environment).

4.1 Fake cases

There are fake created edge test scenarios for this review whose significance magnitudes can be ordered easily with respect to each other.

1. Precise GW skymap temporally and spatially coincident with a neutrino event
2. Coarse GW skymap temporally and spatially coincident with a neutrino event
3. Precise GW skymap temporally coincident with a neutrino event
4. Precise GW skymap spatially coincident with a neutrino event
5. Coarse GW skymap temporally coincident with a neutrino event

Expectations for significances: $a > b > e > c, d \sim 0$

4.2 Running Fake Cases

After ssh-ing in to `reviewers.gwhen.com`, you can run the fake test cases with:

```
~/llamatatest
```

Verbose output will print to the terminal. In particular, you'll be able to see analysis outputs in `~/local/share/llama/current_run/`, which contains four test triggers (each in their own directory). You can also find these files at <http://reviewers.gwhen.com/tests/>, which displays the contents of `~/local/share/llama/current_run/` in your web browser.

⁷ <https://wiki.ligo.org/GWHEN/WebHome>

⁸ <https://wiki.icecube.wisc.edu/index.php/GWHEN>

CONTENTS OF THE OUTPUT FOLDER

5.1 Inputs

- `icecube_neutrino_list.json` The list of IceCube neutrino triggers in the +/-500s window around the GW event.
- `lvc_gracedb_event_data.json` Metadata about the GW trigger pulled from GraceDB. For the test events, it's a minimal example with only a rho value, though in general this is just the JSON pulled from GraceDB.
- `lvc_initial_skymap.fits.gz` The LVC skymap selected for the analysis (an injection for the purpose of the review process).
- `skymap_info.json` Metadata specifying which skymap to use for `lvc_initial_skymap.fits.gz`. Can be parsed from a GCN Notice, LVAAlert, GraceDB, or manually created by a pipeline operator.

5.2 Main Outputs

- `coinc_significance_IceCube-LVC.json` The immediate output of the new significance calculation. Contains information on all inputs, search parameters, and outputs. Not designed to be especially human-readable.
- `coinc_scatterplot_LVC-IceCube.pdf` A joint skymap showing the neutrino triggers scattered over the GW skymap in PDF form.

5.3 Auxilliary Outputs

- `coinc_scatterplot_LVC-IceCube.png` A joint skymap showing the neutrino triggers scattered over the GW skymap in PNG form for upload to GraceDB.
- `IceCubeNeutrinoList.tex` The IceCube neutrinos in a LaTeX table together with their odds ratios calculated for the GW trigger.
- `IceCubeNeutrinoList.txt` Same, but in an ascii format used for GCN circulars.
- `coinc_scatterplot_summary_LVC-IceCube.tex` LaTeX file that can later be used to manually generate a summary PDF, `coinc_scatterplot_summary_LVC-IceCube.pdf`.
- `lvc_initial_skymap.hdf5` The same LVC skymap as above, but converted to an internal HDF5 representation.

5.4 Real cases

The pipeline can also be tested against real cases. Instructions for testing against real cases will be added soon.

USING THE DOCKER IMAGES

The easiest way to get a working copy of LLAMA on any platform is to use the prebuilt LLAMA Docker images. These builds are automatically created and tested from the latest source code and have no external dependencies other than Docker. Docker is kind of like a Linux virtual-machine; you run a Docker “container” on your computer as if it were a totally separate computer, allowing the container to run the same regardless of whether you’re using Windows, Linux, or MacOS.

One difference of convention between VMs and Docker is that you’re usually expected to erase your Docker container as soon as you are done working with it, whereas a VM image might be persisted across multiple use sessions (you’re not forced to use Docker this way, but in general it’s probably useful to remember that Docker containers are supposed to be somewhat more ephemeral than your average VM). You can read plenty more online if you’re interested, but that’s all you need to know to get started.

NOTE that if you are trying to use the LLAMA images on Habanero, you will need to download and run them using Singularity (a container manager like Docker which supports Docker images). Check out the :ref:`instructions for using Singularity with LLAMA images <singularity-llama>` as well as the Habanero Singularity cluster job example documentation⁹.

6.1 Installing Docker

First, you’ll need to install Docker for MacOS¹⁰, Windows¹¹, or Linux. Follow the official Docker instructions. You might need to disable other VM solutions (like VMWare) for this to work. *On Windows, you might be asked whether you want to use Linux or Windows virtual machines; make sure to choose Linux.*

On Linux, Docker should run automatically in the background as a daemon process. On Windows or MacOS, you will need to manually start Docker the way you would any other application (though you can set it to start automatically at startup in both cases).

You will interact with Docker through a command line client that communicates with the Docker daemon. What this means is that you will use a terminal to control Docker via textual commands. If you get error messages saying that the daemon isn’t running, you’ll need to manually start (or restart) the daemon as described above. Again, on MacOS and Windows, this would involve finding the program called Docker and launching it in the same way you would any other program. To access the command line interface, you can open:

- Terminal.app, iTerm.app, alacritty.app, etc on **MacOS**
- cmd.exe or PowerShell on **Windows** (*Note that on Windows you must not use Bash/Ubuntu/Windows Subsystem for Linux to issue Docker commands; use one of the native windows shells mentioned here*)
- xterm, alacritty, etc. on **Linux**

⁹ <https://confluence.columbia.edu/confluence/display/rcc/Habanero+++Job+Examples#Habanero-JobExamples-Singularity>

¹⁰ <https://docs.docker.com/docker-for-mac/install/>

¹¹ <https://docs.docker.com/docker-for-windows/install/>

After installing Docker, you can make sure the command line client is installed and the daemon running with:

```
docker run hello-world
```

which should spit out something like:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:b8ba256769a0ac28dd126d584e0a2011cd2877f3f76e093a7ae560f2a5301c00
Status: Downloaded newer image for hello-world:latest
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://hub.docker.com/>

For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

6.2 Docker Cloud

LLAMA software is saved in a Docker image (basically a snapshot of a working Linux server with LLAMA installed) on [Docker Cloud](https://cloud.docker.com/)¹². You'll need to make an account on Docker Cloud and share your username with Stef, who will add you to the list of contributors to the LLAMA Docker image. This will allow you to "pull" (i.e. download) copies of this image to your computer.

Once you've been added as a collaborator, you should be able to view the [LLAMA repository](#)¹³ on Docker Cloud.

Once this is set up, you can log in from the Docker daemon using the command line interface by running `docker login` and providing your Docker Cloud username and password.

¹² <https://cloud.docker.com/>

¹³ <https://cloud.docker.com/repository/docker/stefco/llama/>

6.3 Getting LLAMA Images

If you just want to run the pipeline, you can use the default image and skip ahead to the :ref:`running a LLAMA container <running-a-LLAMA-container>` section; LLAMA will automatically be pulled if it has not been downloaded already.

You'll need to pull LLAMA images from Docker Cloud in order to use them; this is basically like pulling down a hard drive image of a working LLAMA server (the “image”) which Docker can then run as if it were its own separate server (a “container”, which is the running version of the “image”). You can use the same command to update to the latest LLAMA image.

6.3.1 Choosing the Image

In most cases, you'll want to use stefco/llama:py37-play as your LLAMA Docker image, though a few other options do exist. You can probably skip to the next section unless you want to use one of those other versions (in which case you can substitute that image name for stefco/llama:py37-play where it appears).

The LLAMA images are named `stefco/llama:<TAG>`, where `<TAG>` is one of 3 values (at time of writing) depending on the way the image is configured:

1. `py37` just contains the LLAMA software; it is not configured to communicate with any external services. You will rarely use this as it mainly serves as a base for the other two tags.
2. `py37-play` is configured to pull IceCube data and communicate with the LLAMA team Slack. It will not upload results to IceCube Slack or GraceDB unless you configure it to do so, making it relatively safe for experimentation. It does have LIGO authentication software installed, so you *can* access GraceDB from it by running `kinit albert.einstein@LIGO.ORG` (with `albert.einstein` replaced by your LIGO username, of course) followed by `ligo-proxy-init -k`.
3. `py37-prod` is fully configured for automated production use. There's not really any reason to use this on your laptop unless you know what you're doing.

Additionally, you can access any successful tagged release of LLAMA by appending a dash followed by the version tag of the source code. For example, if you would like to explicitly use LLAMA version v2.28.8, you would append `-v2.28.8` to the end of the image name. The full image name for the `py37-play` tag (which, again, is most appropriate for personal use) would then be `stefco/llama:py37-play-v2.28.8`. If you omit the version, then the latest version will be chosen.

6.3.2 Getting/Updating the Image

You can pull (i.e. download) the latest version of the LLAMA image with:

```
docker pull stefco/llama:py37-play
```

The image is fairly large (several GB), so this will take a while. Note that this will download the latest version of the image even if you have an older version installed, so you can use this command to update to the latest version. See the previous section for more details on choosing a LLAMA image version.

6.3.3 Removing Images

Images are just starting points for containers; you can redownload them whenever you have a fast internet connection, so feel free to delete them when you need to free up space.

You can list installed images with

```
docker image list
```

Which will print something like the below, with each installed image listed on its own line:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
stefco/llama	py37-play	5dd57ee20554	About an hour ago	4.59GB

You can delete this by either referencing the TAG or IMAGE ID. For instance, you could delete the above with:

```
docker image rm stefco/llama:py37-play
```

or

```
docker image rm 5dd57ee20554
```

If you're not running any Docker containers and don't mind redownloading images, you can also delete **all** local Docker data with a single command:

```
docker system prune --all
```

(This might take a little while.)

6.4 Running a LLAMA Container

You can run an interactive LLAMA session using:

```
docker run -it --rm stefco/llama:py37-play bash
```

Note that whatever work you do will not be transferred to your host computer unless you share directories between the host and container. If you are creating analysis results that you want to copy to your local system, you can :ref:`mount Docker directories <mount-Docker-directories>` from your host machine to your local machine as described in the next section.

The previous command will download the LLAMA Docker image if it is not present locally and start it up. A quick breakdown of what each option is doing:

- `run` tells Docker to create a new container from the specified image and run commands on it.
- `-it` is the same as `-i -t`; `-i` tells Docker to set up an interactive session and `-t` tells Docker to create a pseudo-TTY (i.e. a terminal) for interaction. In other words, start up a command line in the LLAMA container so that we can use it like a normal server.
- `--rm` tells Docker to delete this container as soon as we exit from the interactive session, throwing out any changes we've made to the base image. You almost certainly will want to use `--rm` every time you `docker run`.
- `stefco/llama:py37-play` is the name of the container we want to use.
- `bash` is the starting command; you can omit `bash`, in which case you'll be thrown into the somewhat less feature-rich default `sh` shell. (Alternatively, you can specify another shell or command that should run instead of `bash`.)

You'll now have an interactive LLAMA prompt in front of you. You can use the :ref:`LLAMA Command Line Interface <LLAMA-Command-Line-Interface>`_, e.g. by running `llama` and reading the help documentation, or you can start up `ipython` and `import llama` to start using the library directly in iPython.

6.5 Mounting Directories

One of the nice things about Docker is that it lets you share and sync directories between host (your computer) and Docker container. This means you can do work in the container in a shared directory, and the output files will appear in the corresponding directory, and the output files will also appear in the corresponding shared folder in your host machine. Directories are only synced this way if you explicitly request it, so it's easy to avoid unpredictably contaminating your host machine no matter how badly you screw up a container (within reason).

Because paths are specified differently in Windows vs. UNIX-like operating systems, the instructions are slightly different based on the platform, though in both cases you add a `-v host_path:container_path` flag to a `docker run` command like the one given previously in :ref:`running a LLAMA container <running-a-LLAMA-container>`_.

Note that in both cases the directory on the host machine, `host_path`, must exist on the host machine and must be an absolute path, e.g. `/home/stef/dev` will work but a relative path like `dev` will not (at time of writing). `container_path` should resolve to a path in an existing directory; if `container_path` itself already exists, Docker will mount the host volume on top of it, effectively rendering the existing directory inaccessible (which might be desired behavior depending on your use case, but it's something to be aware of).

Note that, if you have a long-running container that you don't want to stop (perhaps because you forgot to mount a volume when you created it), you can still mount a volume on it. It's also possible to specify Docker volumes that persist between containers and are not directly accessible to the host machine; these have higher performance than the shared mounts discussed here and are good for persisting state/transferring data between containers in cases where convenient sharing with the host is not the top priority. Refer to the Docker API for information on how to do this.

6.5.1 Mounting in MacOS/Linux

On MacOS or Linux, you simply specify the host directory to mount followed by a colon and the path on the container. For example, if you're on MacOS and your username is `Stef`, you could mount your home directory to `/root/stef` in the container with `-v /Users/Stef:/root/stef`. The example given in :ref:`running a LLAMA container <running-a-LLAMA-container>`_ would look like:

```
docker run -it --rm -v /Users/Stef:/root/stef stefco/llama:py37-play bash
```

This is the same as the previous example, except now you'll be able to access and modify the contents of your home directory on your host computer from the container's `/root/stef` directory. You can of course use this to save analysis results on your local computer, allowing you to persist your analysis results even after exiting your interactive Docker session (which, as described above, will delete the container automatically when used with the `--rm` flag).

6.5.2 Mounting in Windows

This is the same as in MacOS/Linux, but the path on the host must include the root volume name (e.g. `C:`). You can also use Windows-style back-slash directory separators for the host path, though the container path must still be in UNIX format (forward-slashes). Before you can mount a host directory, however, you'll have to tell Windows to allow the directory to be mounted. This can be done from the Docker daemon graphical control panel in the system tray:

You can then select the drives you wish to make available to Docker containers from your host machine. If you have a single Windows hard drive where you're storing your analysis results, you will most likely want to choose `C:`.

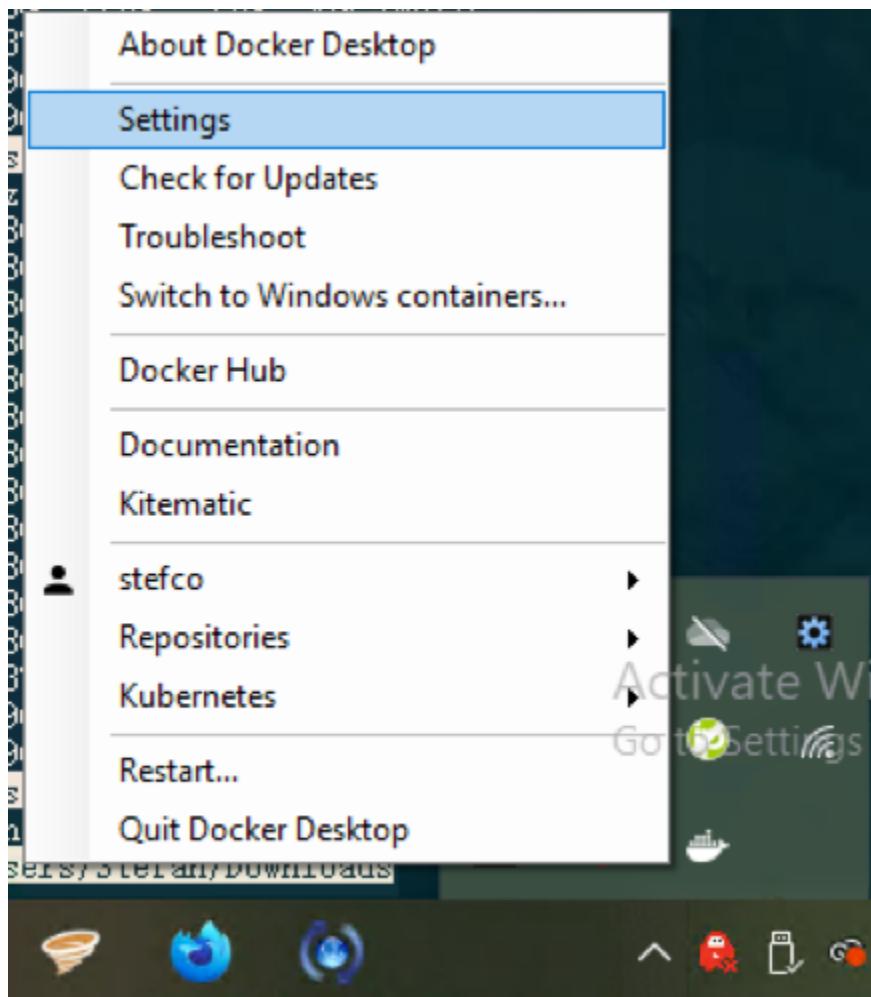


Fig. 1: The Docker daemon settings can be accessed from the system tray on Windows; click on the tray icon and select “settings” from the pop-up menu.

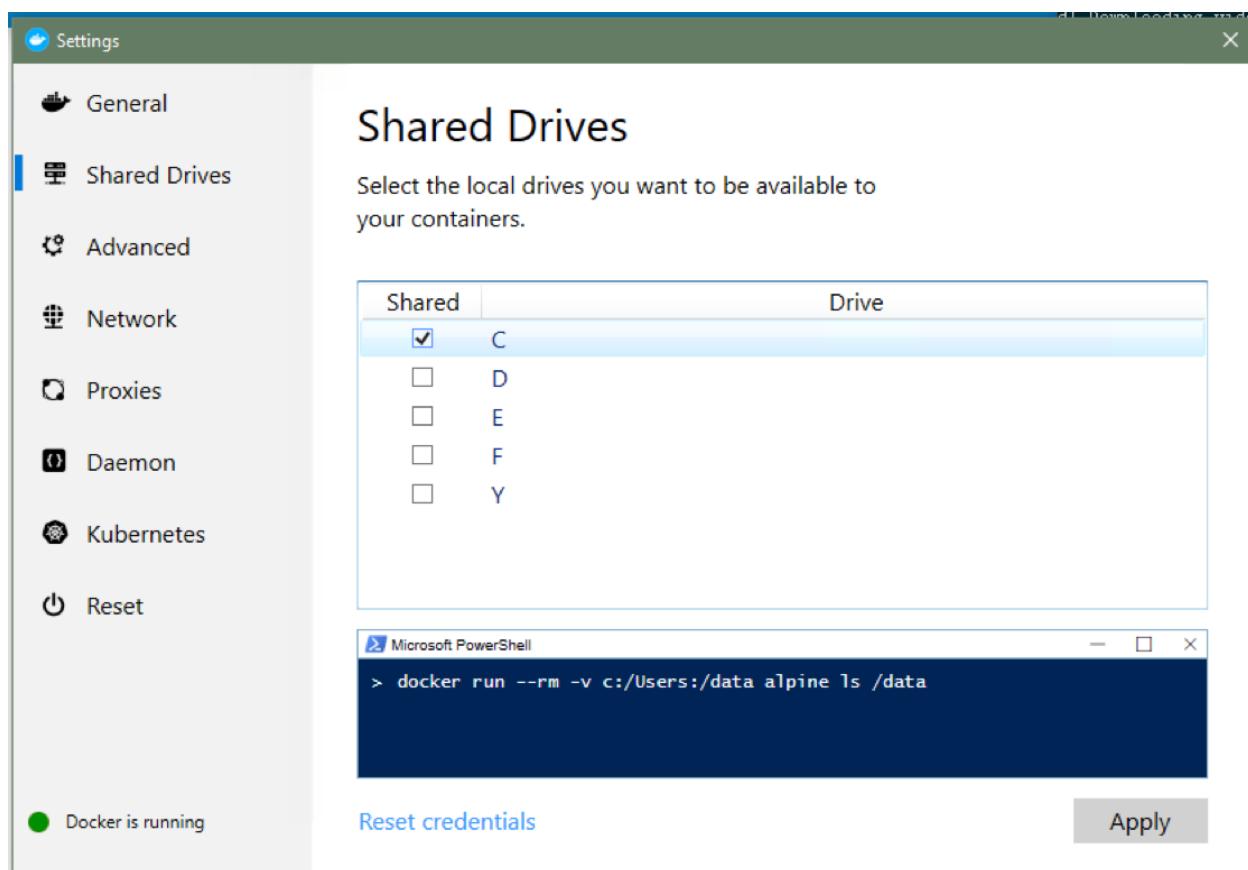


Fig. 2: Select the volumes you'd like to make available for sharing and hit **Apply** when done.

Now, assuming your windows username is `Stef` and you want to mount your home directory to `/root/stef` on the Docker container, you would add the volume mount in the same way as was done in the MacOS/Linux example, though again you'll use Windows-style path syntax (again, remember to use absolute paths rather than relative paths):

```
docker run -it --rm -v C:/Users/Stef:/root/stef stefco/llama:py37-play bash
```

Or, using Windows path style for the host path,

```
docker run -it --rm -v C:\Users\Stef:/root/stef stefco/llama:py37-play bash
```

Note that for Windows hosts the first colon is used to represent the local volume, and so the second colon is the one that separates the host path from the container mount path.

6.6 Out of Memory/Disk

If you are getting a `MemoryError` when you run memory-intensive parts of the analysis, you can either add more memory through your Docker settings or add `:ref:`swap space <swap-space>`` to your container; this can be done with the linked instructions within an interactive container session.

You can also increase the amount of disk space available to Docker from the Docker daemon control panel, though you are unlikely to need much space if you are doing one-off activities like processing public events from an observing run.

RUNNING ON HABANERO

If you're trying to run on the Habanero cluster, first read the [Choosing the Image](#) section to learn how to pick the correct LLAMA image for your job.

7.1 Interactive Habanero Jobs

Log in to Habanero using:

```
ssh uni@habanero.rcs.columbia.edu
```

Where `uni` is your Columbia UNI (i.e. username). You'll be asked for your Columbia password. If you can't log in this way, ask Zsuzsa whether you have access to Habanero.

When you first log in to Habanero, you will be using a log in node; you are running on a shared server that puts severe limits on how much computing power you can use. You are supposed to use this node only for submitting batch jobs or, if you want to do interactive work (which we will below), for starting an interactive session on a high-powered node.

You can start an interactive session with:

```
srun --pty -t 0-01:00 -A geco /bin/bash
```

This will take a few seconds to start up. Once it does, you'll be running on an interactive node with much more available computing power. It's a good idea to do this pretty much any time you want to do anything on Habanero. Of course, the above command is hard to memorize, so you'll probably want to add the following to your `~/.bashrc`:

```
alias interactive='srun --pty -t 0-01:00 -A geco /bin/bash'
```

Refresh your `~/.bashrc` with `source ~/.bashrc`. Now, you can start an interactive session simply by running `interactive`, and it will execute the same command we ran above.

Once you are finished with your interactive session, you can stop it by running `exit`, and you will be returned to the login node. **BETWARE that Habanero likes to boot users out of interactive sessions after a few hours, and that there is no way to log out of an interactive session without killing it.** Habanero is designed as a batch-processing cluster; if you need a persistent server, spin one up using DigitalOcean or something like it.

7.2 Docker Hub Authentication with Singularity

You'll probably want to use the private LLAMA Docker images, which have all of the LLAMA software installed and certain authentication credentials pre-configured. To do this, you will need access to the private [Docker Hub LLAMA repository](#)¹⁴. (Note that if you can't view the previous link, it probably means you need to create Docker Hub login credentials and then ask Stef to add you as a collaborator to that repository.) To provide Docker Hub login credentials to Singularity, you'll need to set the `SINGULARITY_DOCKER_USERNAME` and `SINGULARITY_DOCKER_PASSWORD` environmental variables to your Docker Hub username and password, respectively. The easiest way to do this is to add the following two lines to your `~/.bashrc`:

```
# put this in ~/.bashrc
export SINGULARITY_DOCKER_USERNAME="username"
export SINGULARITY_DOCKER_PASSWORD="password"
```

Where, of course, you replace `username` with your username and `password` with your password. Reload your `~/.bashrc` to make these changes take effect with:

```
source ~/.bashrc
```

7.3 Fixing Singularity Cache on Habanero

Unfortunately, Habanero only lets you keep a small amount of data in your home directory. This is a problem for Singularity, which keeps its downloaded data in `~/.singularity`; LLAMA's huge docker images will rapidly use up your space quota, and your attempts to pull images will fail; this is true even if you try to save your downloaded images somewhere with space, since the files will first be downloaded to `~/.singularity` anyway.

The solution to this problem is to make your own directory in our much larger group data volume, `/rigel/geco/users`, and create a symbolic link pointing from the default `~/.singularity` path to one stored in your directory on the group volume. First, make your directory in the group volume (if you haven't already). Run the following, **replacing uni with your Columbia UNI**:

```
mkdir /rigel/geco/users/uni
```

You'll get an error if the directory exists or if you are not part of the GECo group on Habanero. If this second case is the issue, bug Zsuzsa to get you in the group.

Now, you might already have a `~/.singularity` directory set up; if that's the case, you can go ahead and remove it, since it generally only contains cached data.

```
rm -rf ~/.singularity
```

Next, make a new Singularity directory in you `/rigel/geco` directory, again **replacing ``uni`` in the below command with your Columbia UNI**:

```
mkdir /rigel/geco/users/uni/.singularity
```

Finally, you can create a symbolic link telling the operating system to use this new directory whenever it wants to use `~/.singularity` (again, **replace uni with your Columbia UNI**):

```
ln -s /rigel/geco/users/uni/.singularity ~/.singularity
```

¹⁴ <https://hub.docker.com/repository/docker/stefco/llama>

7.4 Load Singularity Module

You'll need to load the Singularity module before you'll have access to the command line interface. Module loading is Habanero's way of letting you choose which programs are available; a lot of stuff that's installed is only available for use by calling `module load <modulename>`. Let's load the Singularity module:

```
module load singularity
```

7.5 Pull LLAMA Image

Now, we're finally ready to get our LLAMA image. Again, if you need help choosing an image, refer to the [Choosing the Image](#) section. Let's say you want to use the latest version of the play image (used for processing data without pushing it to IceCube or other collaborators), called `stefco/llama:py37-play`. First, navigate to your personal directory in `/rigel/geco` (replace uni with your Columbia UNI):

```
cd /rigel/geco/users/uni
```

You can now pull the image and save it in Singularity's format as `llama-play.sif` using the following command (though be warned, **it will take a VERY long time for Singularity to build the image after pulling it**):

```
singularity pull llama-play.sif docker://stefco/llama:py37-play
```

Again, this command will take a very long time; even after it has finished copying data from Docker Hub, it will spend a while combining the data into a single output image, `llama-play.sif`. Don't worry if it seems like it's frozen; as long as it hasn't exited with an error, it's probably running okay.

Once this command finishes, run `ls`; you should see your new image, called `llama-play.sif`. You can run this image using:

```
singularity run llama-play.sif
```


LOCAL INSTALLATION

These installation instructions are for more advanced users who want to install LLAMA on their local machine without the use of Docker. See :ref:`the developer instructions <developer-instructions>` for information on developer dependencies and tools as well as further background documentation.

This section is not guaranteed to be as up-to-date as the Docker instructions because it is subject to more frequent change. The official provisioning/installation procedure can always be recovered from the continuous integration procedures used to create the Docker images.

8.1 System Requirements

Make sure you have at least 4GB of memory (physical or virtual; :ref:`swap space <swap-space>` is fine) and 15GB of free space on your file system.

8.2 Installing Conda

LLAMA depends on LIGO tools that are only distributed via conda, so you'll need to install an Anaconda python distribution to get up and running (*developer notes on Conda <migrating-to-conda>*): *Conda installs are done on a per-user basis, so you won't need to use sudo for any of the below.* Start by installing the latest version of Conda:

```
curl -0 https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh  
bash Miniconda3-latest-Linux-x86_64.sh
```

Log out and log back in again, then activate `conda-forge` and install LIGO tools:

```
conda activate  
conda config --add channels conda-forge  
# old method: use LIGO's environment  
# wget -q https://git.ligo.org/lscsoft/conda/raw/master/environment-py36.yml  
# conda env create -f environment-py36.yml  
curl -0 https://raw.githubusercontent.com/stefco/llama-env/master/llama-py36.yml  
conda env create -f llama-py36.yml
```

Activate the LIGO virtualenv (**NOTE: You will need to do this every time you want to use this python setup! Consider putting this command in your .bashrc file.**):

```
conda activate llama-py36
```

Clone the LLAMA repository into `~/dev/multimessenger-pipeline`:

```
mkdir -p ~/dev  
cd ~/dev  
git clone git@bitbucket.org:stefancountryman/multimessenger-pipeline.git  
cd multimessenger-pipeline
```

Fetch all data files (make sure you have `git-lfs` installed)

```
git lfs fetch  
git lfs checkout
```

Install dependencies:

```
curl -O https://raw.githubusercontent.com/stefco/llama-env/master/requirements.txt  
pip install -r requirements.txt
```

Install the pipeline in developer mode:

```
python setup.py develop
```

Confirm installation succeeded by seeing if you can print the help command from the command-line interface (CLI):

```
llama --help
```

Optionally, run LLAMA's test suite to make sure things are working okay (though note that many tests will fail if you haven't :ref:`entered your authentication credentials for external services <config-and-auth>`):

```
make test
```

That's it! All important llama tools can be accessed at the command line as subcommands of the `llama` command; run `llama --help` to see what's available. The `llama` CLI follows the same structure as the `llama` python modules, which you can import into your python scripts.

SETTING UP A PRODUCTION SERVER

You can set up a production server environment on a Debian server as follows. You can skip the installation steps for Docker and Docker Compose if you already have them installed.

9.1 Install Docker

The LLAMA production environment runs in a Docker¹⁵ container to massively simplify deployment and reproducibility. On Debian:

```
curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add -
apt-key fingerprint 0EBFCD88
|
add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/debian \
    $(lsb_release -cs) \
    stable"
apt-get update
apt-get install -y docker-ce docker-ce-cli containerd.io
docker run hello-world
```

9.2 Install Docker Compose

We use Docker Compose¹⁶ to turn on all LLAMA components at once and to keep them running. Install Docker Compose with:

```
curl -L \
    "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" \
    -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
docker-compose --version
```

¹⁵ <https://docs.docker.com/install/linux/docker-ce/debian/>

¹⁶ <https://docs.docker.com/compose/install/>

9.3 Log in to Docker Cloud

The LLAMA production docker images are in a private repository on Docker Cloud¹⁷. You will need to be added to the `stefco/llama` Docker Cloud repository (contact Stefan Countryman for details) and log in on the production server using `docker login`:

```
docker login
```

You'll be prompted for your Docker Cloud login credentials; enter them to log in.

9.4 Get `docker-compose.yml`

`docker-compose.yml` is a file that describes how to combine Docker containers together; since LLAMA has a few moving parts that need to work together, it saves us the trouble of having to remember what steps are necessary to turn the pipeline on and keep it running. You can think of it as a shortcut for calling a ton of `docker` commands every time we want to start up our app.

You can always pull the latest version of `docker-compose.yml` from the [LLAMA repository](#)¹⁸ with this command:

```
pushd ~ && git archive \  
  --remote=git@bitbucket.org:stefancountryman/multimessenger-pipeline.git \  
  HEAD docker-compose.yml \  
 | tar -x && popd
```

9.5 Starting LLAMA Production App

Fortunately, all the hard work is done for you in `docker-compose.yml`; you just need to start it with `docker-compose up`:

```
docker-compose up
```

¹⁷ <https://cloud.docker.com>

¹⁸ <https://bitbucket.org/stefancountryman/multimessenger-pipeline/src/master/>

**CHAPTER
TEN**

INTRODUCTION

This guide covers the operation of the pipeline, both in production and for the purpose of running sensitivity studies (simulated background and signal events for various populations). It also describes troubleshooting scenarios and strategies.

HOW DATA IS ORGANIZED

Persistent data goes into `$XDG_DATA_HOME/llama` or, if `$XDG_DATA_HOME` is not defined, into `~/.local/share/llama`. This includes analysis outputs, archived alerts, and log files. This directory will be called the “output directory” or `$OUTPUT_DIR` in this section.

11.1 Event Directories

Events each get their own directory whose names are their respective LIGO/Virgo GraceIDs (i.e. their unique event IDs on GraceDB). Event directories for the current LIGO/Virgo run are all held in `$OUTPUT_DIR/current_run`. Event directories for old events/manual runs, if they are saved, will be in different run directories, each of which should be located in `$OUTPUT_DIR/past_runs`, so that, for example, GW170817, the world’s first BNS merger detection, can be found on the current LLAMA analysis server in `~/.local/share/llama/past_runs/events-o2/G298048`.

11.2 GCN Notice Archive

All GCN notices heard by the server while `gcnd` is running will, by default, be archived to `$OUTPUT_DIR/gcn` (this includes non-LIGO/Virgo triggers). If you wish to view these alerts, you can look in this directory, which contains subdirectories `YEAR-MONTH/DAY` (to avoid having too many files in a single directory).

11.3 Log Files

Logs are stored in `$OUTPUT_DIR/logs`. While formats and contents for different log files differ based on which program created them, they are generally named after the script that created them. The LLAMA update daemon, `llama run`, writes to logfiles with `llamad` in their names; the GCN listener, `llama listen gcn`, writes to `gcn.log`. It’s often useful to know which log file was written to most recently (e.g. to find out whether the script you ran is faithfully logging output as expected); you can do this with `ls -ltr`.

CHAPTER
TWELVE

RUNNING THE PIPELINE AUTOMATICALLY

This section will be written later.

RUNNING THE PIPELINE MANUALLY

Several crucial files can be generated manually (that is to say, without relying on automated trigger acquisition and pipeline execution). This section describes common usage scenarios.

`skymap_info` is a command line script for creating new event directories along with a new `skymap_info.json` file describing the gravitational wave event and specifying the GW skymap that is to be used. Creating this file is the first step in any GW multimessenger analysis in LLAMA (see the documentation for `llama.files.skymap_info.SkymapInfo` for more information).

Ordinarily this file is generated by one of the listener scripts (e.g. the ones listening for new LVAAlerts from LIGO/Virgo's GraceDB or the GCN Notice listener). The `skymap_info` script can be used to generate this file manually, however, using either the event's GraceID (it's unique ID on GraceDB) or using a VOEvent XML file (as generated on GraceDB and distributed as a GCN Notice). There are many options associated with making a new `skymap_info.json` file (run `skymap_info -h` to see full documentation of the command line interface), but by default, all you need to specify is either the GraceID or a VOEvent file and everything else will be inferred.

13.1 Using Defaults with `skymap_info`

In the simplest case, you can create a new event and its corresponding `skymap_info.json` file using an existing VOEvent with:

```
skymap_info --voevent path/to/voevent.xml
```

or using the event's GraceID (and, implicitly, the most recently uploaded skymap) with:

```
skymap_info --graceid GRACEID
```

Where `GRACEID` can be either a conventional GraceID (starting with "G" for real events, "M" for mock events) or a superevent GraceID (starting with "S" for real superevents, "MS" for mock superevents). In most cases, you will want to run with a superevent GraceID, since this is the official public data product released by LIGO/Virgo.

Whether you use `--voevent` or `-graceid`, you will now be prompted to specify whether this event is a real event or a test event. **Please answer this question carefully, as it determines whether results will be automatically published to collaborators.** If it is a real event, data products will go out to collaborators (via Slack and possibly other methods) when you run `llama run` to update the rest of the contents of the directory. This is the same behavior you'd get from the pipeline when it's automatically triggering off of real data. Of course, if you're looking at a real event, this is a good thing, so don't be shy about saying YES; just make sure you answer *correctly* so as not to bother or confuse people needlessly.

CHAPTER
FOURTEEN

SENSITIVITY AND BACKGROUND STUDIES (O3B)

These instructions apply starting in the second half of O3 (O3B). See the next section for pre-O3B instructions. You might still want to read those instructions for a more detailed overview of how tools work: this section is currently underdeveloped.

Make sure that all data is available on DigitalOcean Spaces and is organized into directories by GraceID and background neutrino list names. For mid-O3 subthreshold runs, this data is located in:

```
s3://llama/llama/artifacts/mid-o3-subthreshold-background/
```

Contents at run time:

```
DIR    s3://llama/llama/artifacts/mid-o3-subthreshold-background/
↳ bklist/
DIR    s3://llama/llama/artifacts/mid-o3-subthreshold-background/
↳ superevents/
2019-11-04 00:45  1088890  s3://llama/llama/artifacts/mid-o3-subthreshold-background/
↳ bklist_names.txt
2019-10-26 20:32  30656   s3://llama/llama/artifacts/mid-o3-subthreshold-background/
↳ missing_event_ids_cwb.npy
2019-11-04 00:30   27      s3://llama/llama/artifacts/mid-o3-subthreshold-background/
↳ placeholder.json
2019-11-03 23:58   244     s3://llama/llama/artifacts/mid-o3-subthreshold-background/
↳ skymap_info.json.template
2019-11-04 00:44   67832   s3://llama/llama/artifacts/mid-o3-subthreshold-background/
↳ superevents.txt
```

In principle, this can be run anywhere that has an internet connection, supports Docker (or has LLAMA dependencies and library installed), and can provide 8GB of memory for peak usage. These instructions assume you are using DigitalOcean droplets provisioned from a seed image using `llama com do`.

Start by getting a running server with `llama` installed under `~/dev/multimessenger-pipeline`. Make sure it doesn't need any extra `$PATH` configuration in order to work. Create a snapshot of this server. Let's say it's `mid-o3-subthreshold` for the sake of provisioning a bunch of servers for the parallel runs.

Follow the instructions in the pre-O3B section below to set things up, then launch with:

```
llama dev dv batch
```

This will run `bin/mid-o3-subthreshold-background-event-runner` on all servers whose names start with `llama-`. Make sure code is running on those servers with:

```
llama dev dv ls-procs
```

Fetch results from S3 using s3cmd

```
s3cmd get --recursive \
s3://llama/llama/artifacts/mid-o3-subthreshold-background/mid-o3-subthreshold-
background/ \
.
```

Finally, collate files into a results table as described below and put them into the background distribution file used for calculating p-values.

SENSITIVITY AND BACKGROUND STUDIES (PRE-O3B)

At time of writing, LLAMA uses DigitalOcean¹⁹ hosting for sensitivity studies. There are a set of scripts used for setting up a large (~100) number of identical DigitalOcean “droplets” (virtual private server instances), getting analyses running on them, collating the outputs, and then spinning down those droplets.

Before you can spin up any servers, you’ll need to have someone on the GECo DigitalOcean²⁰ account invite you to join the team. Visit the [team management page²¹](#) and invite the new user to the team. You will need to create a DigitalOcean²² account using the email provided (or log in if you already have one), at which point you will be able to switch users to the GECo team account by clicking the profile button in the top right corner of the DigitalOcean²³ web interface.

You’ll now need an API token in order to be able to access DigitalOcean using LLAMA’s utility scripts (i.e. without using the web interface). **Your access token is basically like a password, and you should treat it with EXTREME care.** If someone gets access to your token, they’ll be able to rack up charges on the team server (i.e. by mining crypto currency or something) and even delete our data. Ideally, once you create your token, you should only store it on the computer that will be using it, and you should make sure not to put copies anywhere where they can be abused, e.g. a git repository (a **very** common mistake!!). You should also delete old tokens as long as they are not being used any more or if you have the slightest suspicion that any have been compromised.

Go to the [API token page²⁴](#) and create a new token, making sure you check the box for `write` access. (You’ll only have one chance to copy this token down; if you accidentally navigate away from the page, you’ll just need to delete that token and generate a new one.) Now, you can add this token to your command line environment by sticking the following line somewhere in your `.bashrc` (or `.bash_profile` for MacOS):

```
export DIGITALOCEAN=<YOUR-TOKEN-HERE>
```

Where, of course, `<YOUR-TOKEN-HERE>` is replaced with the API token you just generated. Close and reopen your shell window to make sure that this environmental variable is loaded. You can check that it’s loaded by running:

```
echo $DIGITALOCEAN
```

which should print the API token you just created. If it didn’t, make sure you edited the correct file (again, MacOS uses `~/.bash_profile`, Linux uses `~/.bashrc`), that you wrote the `export` command above (with *no spaces around the = sign!*), and that you restarted your bash shell after saving so that the startup script loaded the new variable.

¹⁹ <https://cloud.digitalocean.com/>

²⁰ <https://cloud.digitalocean.com/>

²¹ <https://cloud.digitalocean.com/account/team>

²² <https://cloud.digitalocean.com/>

²³ <https://cloud.digitalocean.com/>

²⁴ <https://cloud.digitalocean.com/account/api>

15.1 Editing your .bashrc

Note: your `.bashrc` or `.bash_profile` (on MacOS) files are scripts that run whenever bash starts up. They let you do things like set environmental variables (what we're doing now) and other useful things to customize your shell and make it more useful. You can edit your `.bashrc` or `.bash_profile` with:

```
# On Linux  
nano ~/.bashrc  
  
# On MacOS  
nano ~/.bash_profile
```

and then add the same `export DIGITALOCEAN=<YOUR-TOKEN-HERE>` code mentioned above.

15.1.1 Using the DigitalOcean API

You can use the command-line utility `llama com` do for interacting with the DigitalOcean API through the command line (our scripts for running sensitivity studies rely on this script). If you have the LLAMA pipeline installed, you can find this script in `<REPODIR>/bin` (where `<REPODIR>` is the location of the LLAMA software directory).

Since the scripts we are using are mostly located in `<REPODIR>/bin`, it's probably easiest to clone the full LLAMA repository (if you haven't already). If you already have this code and have `<REPODIR/bin>` in your `$PATH` (i.e. you can use the commands it contains at the command line), you can skip the next subsection.

15.2 LLAMA Scripts

All `llama` scripts used for background runs are part of the `llama` distribution's command line tools. Use a Docker image or [install the pipeline locally](#) to get access to them. The two main tools you'll be using are `llama com` do to start up DigitalOcean servers/interact with the infrastructure and `llama dev` drop to control the servers.

15.3 Adding Your SSH Keys to the Droplets

You will need to be able to SSH into the droplets where the analysis is being run in order to get the analyses started, check on their statuses, and fetch output data. In order to do this, you will need to give an SSH key (sort of like a password that is unique to your computer) to [DigitalOcean²⁵](#) under the GECo account. By default, `llama com` do will add all available GECo SSH keys to newly provisioned droplets (so that all team members can access analysis servers at will).

For starters, you'll have to [make SSH keys](#) if you haven't already. If you put your key in the default location, you can print it to the terminal with:

```
cat ~/.ssh/id_rsa.pub
```

Next, make sure you're logged in as GECo on [DigitalOcean²⁶](#), then copy the above key and navigate to the [DigitalOcean security page²⁷](#) to add your SSH key. Create a new SSH key and name it something that makes it clear who you are and which device you are using (e.g. Stef-Laptop).

²⁵ <https://cloud.digitalocean.com/>

²⁶ <https://cloud.digitalocean.com/>

²⁷ <https://cloud.digitalocean.com/account/security>

Once you've saved your SSH key, you'll have automatic access to all future droplets created through the `llama com do` interface.

15.4 Installing Requirements

The core CLI tools are part of the LLAMA distribution, though you will also need to have the developer packages installed (defined in `requirements-dev.txt` and as Conda packages in the Dockerfiles in `/docker/llama-dev/`). You can use the developer Docker images to satisfy the non-python dependencies or install them on your local system.

The scripts and examples in this guide were written with bash 4 in mind. If you are on any recent Linux system, you will almost certainly be running bash 4 or later. MacOS systems, however, come with bash 3 installed by default (due to licensing issues, they cannot distribute newer versions). To install the latest version of bash on MacOS:

1. Install MacPorts²⁸ using their instructions.
2. Install bash with `sudo port install bash`.
3. Make a copy of the old list of shells (used by the system to keep track of available shells) with `cp /etc/shells ~/etc-shells`.
4. Edit the list of shell files with `sudo nano /etc/shells` and add `/opt/local/bin/bash` on a new line. Feel free to use a different text editor than nano, e.g. the default graphical text editor with `sudo open /etc/shells`.
5. Run `chsh -s /opt/local/bin/bash` to tell your computer that the MacPorts version of bash is the one you would like to use by default. You will need to enter your password to do this.
6. Without closing your current shell window, open a new window and run `echo $BASH_VERSION`. You should see something starting with “4”.

If things look weird, or if the terminal doesn't start properly, go back to the old shell window you were using, restore the old version of the `/etc/shells` file (in case you broke it) by running `sudo cp ~/etc-shells /etc/shells` followed by `chsh -s /bin/bash` to go back to the default system bash. Open a new window to confirm that you've fixed the damage (everything should now be back to normal), then carefully retry the above steps.

15.5 DigitalOcean Administration Examples

Show full help documentation with:

```
llama com do -h
```

You can list the current running droplets using:

```
llama com do -D
```

If you only want to list droplets whose name start with “llama”, you can add a wildcard to the end of the command (as if you were adding a wildcard to match a bunch of files in a folder):

```
llama com do -D llama*
```

You can also print just the IP addresses of these servers by telling `llama com do` to print only those columns (you can choose any combination of columns besides this example; see the help documentation with the `-h` flag for a full list):

```
llama com do -D llama* -C ip
```

²⁸ <https://www.macports.org>

You can get rid of the header row that labels the columns (useful if you are going to pipe those IP addresses to another script) by adding the `-n` (no headers) flag:

```
llama com do -n -D llama* -C ip
```

You can list the currently saved snapshots on your [DigitalOcean](#)²⁹ account with:

```
llama com do -S
```

Create a single droplet called `llama-provision` from the snapshot called `llama-parallel-early-er14-o2-bg` with:

```
llama com do -c llama-provision -i llama-parallel-early-er14-o2-bg
```

You can create 98 servers numbered `llama-parallel-00` through `llama-parallel-97` (0 included) using the `llama-parallel-early-er14-o2-bg` snapshot with:

```
llama com do \
    -c llama-parallel-{00..97} \
    -i llama-parallel-early-er14-o2-bg
```

You can delete all servers whose names start with “llama” with the `-d` flag; you’ll be asked for final confirmation before the deletion happens, so make sure to double check that the servers listed are the ones you want to actually delete (**deleting droplets is NOT reversible!**):

```
llama com do -d llama*
```

(be sure to type “y” in the prompt to confirm that you want to create the new droplets; note that it will take a while for them to start up. You can monitor their progress by running `llama com do -D` as mentioned above; droplets that will be listed as `new` while they are provisioning, and you will not be able to log into them or otherwise use them until they are finished starting up and are listed as `active`.

15.5.1 Preparing a Server

When you run this analysis in parallel, you’ll be doing it on a bunch of different virtual servers. In order to keep things as simple as possible, you want these servers to be identical in all respects (except, possibly, for input files, which will need to be different if you’re not doing some sort of Monte Carlo sampling like we do in the background studies). In particular, you’ll want to copying large files to and from the individual servers; a 7GB list of skymaps will eat up a whole day on a good connection if you need to copy it from Columbia’s network to 100 separate [DigitalOcean](#)³⁰ servers.

15.6 Server Preparation Overview

The solution (described in detail where necessary in the links below) is to:

1. Create a single droplet, most likely cloning it from the disk snapshot used for the most recent parallel run (see the [code example](#) above)
2. Set everything up on it by moving data files into place and delete old output files; see the [section on moving files into place](#) below.
3. Shut it down at the command line with `sudo shutdown now`.

²⁹ <https://cloud.digitalocean.com/>

³⁰ <https://cloud.digitalocean.com/>

- Take a snapshot of the droplet by opening it on the DigitalOcean³¹ control panel, going to the snapshot page, entering a descriptive name (try to make it clear when this run happened and which set of skymaps it was run on), and clicking the make snapshot button. Making the snapshot will take ~1 hour, so go do something else while you wait.

Once you've done these tasks, you can [spin up servers](#) and [get the analysis started](#). Step 2 is the most complicated and most subject to change; the following section describes it in detail.

15.7 Moving Data into Place

Note: this whole section will be run on some LIGO DataGrid server. You will need to SSH into it with gsissh.

At time of writing, simulated skymaps are located on `ldas-pcdev1.ligo-la.caltech.edu`. Enter your LIGO credentials with:

```
kinit albert.einstein@LIGO.ORG
```

And enter your LIGO password (obviously replace “albert.einstein” with your LIGO username; you’ll need DataGrid credentials for this, so make sure to set those up if you haven’t already). Now create a grid proxy from the Kerberos credentials initialized above with:

```
ligo-proxy-init -k
```

Now, you can SSH into the LIGO DataGrid mentioned above:

```
gsissh ldas-pcdev1.ligo-la.caltech.edu
```

(Alternatively, if you are logging in with `ssh albert.einstein@ssh.ligo.org`, you would select LLO and then pcdev1.) Navigate to the directory where Rainer has been keeping skymaps:

```
$ cd /home/kenneth.corley/GWHEN_sample_skymaps
$ ls
BBH_design_sample_skymaps  BBH_03_sample_skymaps      BNS_03_sample_skymaps
BBH_02_sample_skymaps      BNS_design_sample_skymaps README.txt
```

You should see a list of directories whose names describe the type of skymaps contained. Skymaps should be in a directory called something like `allsky` within one of these directories. Sometimes the skymaps are split up between multiple runs. For example, to get the O3 sensitivity above-threshold BNS skymaps, run:

```
cd BNS_03_sample_skymaps/BNS_03_1/allsky
```

There should be very many (8978) `.fits` skymap files (along with other output files generated by the skymap simulations) in this directory. (We’ll only zip up `BNS_03_1` since most of the skymaps are in this directory; we will ignore `BNS_03_2` for now.) You will need to compress these skymaps into `.fits.gz` files (the format required by the LLAMA pipeline). You can’t write to Rainer’s directories, so you’ll need to make a directory in your own home directory to zip these files to (let’s call it `BNS_03_sample_skymaps` for clarity). Run the following command from the `allsky` folder above:

```
mkdir ~/BNS_03_sample_skymaps
# run this next line from within the directory containing the skymaps
# you're zipping
find . -name '*.fits' | while read fitsfile; do
```

(continues on next page)

³¹ <https://cloud.digitalocean.com/>

(continued from previous page)

```
echo "On ${fitsfile}...";  
gzip \  
  <"${fitsfile}" \  
  >~/BNS_03_sample_skymaps/"$(basename "${fitsfile}")".gz;  
done
```

(This will take a while to complete.) Once you've zipped these files, you can take that output directory and *zip it up again* into a compressed tarball (.tar.gz extension). This will put the whole directory into a single file, which makes it easier to move around (the final file will be ~10GB in size).

```
cd  
tar -czf BNS_03_sample_skymaps.tar.gz BNS_03_sample_skymaps
```

You now have a tar archive of the skymaps you want to use! Congratulations. You'll probably want to download this directly from the server you are provisioning (since the connection between DigitalOcean's servers and LIGO's data centers is probably faster than your home connection). Read on for instructions on where to move the files on the DigitalOcean server you are provisioning.

This section should be run on the DigitalOcean droplet you are provisioning.

SSH into the DigitalOcean³² server you've created for provisioning purposes. All files for background and sensitivity studies are located in ~/pre-er14-background (for historical reasons). Navigate there and list the contents:

```
$ cd ~/pre-er14-background  
$ ls  
bklist    neutrinos          signal-events  
events    scratchevents      signal-neutrinos  
LOCKDIR   signal-doublet-events skymap-lists  
logs      signal-doublet-neutrinos skymaps
```

These directories contain inputs and outputs for the background and sensitivity studies. We are particularly interested in:

- **bklist**: Lists of neutrinos used for background calculations
- **skymaps**: A directory containing subdirectories filled with skymaps (since you might want to lump together a few runs worth of simulated skymaps, each in a separate directory, into a single background run) used by both background and sensitivity runs. For example, you might put a directory called allsky filled with .fits.gz skymaps into skymaps, so that a typical input skymap called 0.fits.gz would be located at ~/pre-er14-background/skymaps/allsky/0.fits.gz.
- **skymap-lists**: Lists of filepaths to skymaps in skymaps that specify subsets of the full set of skymaps. These are used to distinguish between 2- and 3-detector cases; each list will have skymaps that correspond to a specific set of GW detectors that participated in the reconstruction of that skymap. When you *start the analysis*, you can specify one of these lists to only run on skymaps corresponding to that detector configuration. These skymap lists contain one full skymap path per line.

You can make these skymap lists by downloading the coins.dat file from Rainer's directories on ldas-pcdev1.ligo-la.caltech.edu. The fastest way to get these and the skymaps is do download them from our DigitalOcean Spaces directories; simulated skymaps are located in subdirectories of https://1lalma.nyc3.digitaloceanspaces.com/simulations/ligo-la/~kenneth.corley with names like BNS_03_sample_skymaps.tar.gz providing descriptive titles (you will also need the filename hash, so the easiest way to do this is to list files in that subdirectory through a command-line interface or the Spaces web-browser). Within the unzipped tarfile, you'll be able to find the coins.dat files in the specific simulation run you want to use; in the above example, that would be in the BNS_03_1 subdirectory. For reference, on the LLO LDG

³² <https://cloud.digitalocean.com/>

servers (ldas-pcdev1.ligo-la.caltech.edu), coincs.dat would be in the /home/kenneth.corley/GWHEN_sample_skymaps/BNS_03_sample_skymaps/BNS_03_1 directory. Once you have coincs.dat, you can extract the skymap filename lists with something like:

```
for dets in H1,L1 H1,L1,V1 H1,V1 L1,V1; do START="$dets" python -c 'import os; open(../skymap-lists/bns-o3/"${os.environ["START"]}".replace(",","")+" .csv", "w").write(("\n").join(f"/home/vagrant/pre-er14-background/skymaps/{l.split()[1]}.fits.gz" for l in open("participating-detectors-bns-o3.txt").readlines() if l.split()[0] == os.environ["START"])+"\n"); done
```

confirm that the *total* number of filenames matches the number of filenames split between the lists (though this list's length should be one greater because it contains an initial header line):

```
wc -l ~/pre-er14-background//skymap-lists/bns-o3/*
2013 /home/vagrant/pre-er14-background//skymap-lists/bns-o3/H1L1.csv
5507 /home/vagrant/pre-er14-background//skymap-lists/bns-o3/H1L1V1.csv
703 /home/vagrant/pre-er14-background//skymap-lists/bns-o3/H1V1.csv
755 /home/vagrant/pre-er14-background//skymap-lists/bns-o3/L1V1.csv
8978 total
wc -l ~/pre-er14-background/skymaps/coincs-bns-o3.dat
8979 coincs-bns-o3.dat
```

The file contents should look something like this:

```
$ head ~/pre-er14-background/skymap-lists/bns-o3/H1L1V1.csv
/home/vagrant/pre-er14-background/skymaps/bns-o3/1.fits.gz
/home/vagrant/pre-er14-background/skymaps/bns-o3/2.fits.gz
/home/vagrant/pre-er14-background/skymaps/bns-o3/3.fits.gz
/home/vagrant/pre-er14-background/skymaps/bns-o3/4.fits.gz
/home/vagrant/pre-er14-background/skymaps/bns-o3/5.fits.gz
/home/vagrant/pre-er14-background/skymaps/bns-o3/6.fits.gz
/home/vagrant/pre-er14-background/skymaps/bns-o3/8.fits.gz
/home/vagrant/pre-er14-background/skymaps/bns-o3/16.fits.gz
/home/vagrant/pre-er14-background/skymaps/bns-o3/17.fits.gz
```

- **logs:** Contains logfiles for each run; background logfiles are just a timestamp (an integer) followed by .log (with the largest number representing the latest timestamp, and hence the most-recently started run).
- **events:** Output run directory containing one directory for each simulated background trigger (i.e. a GW skymap with a list of neutrinos plus all outputs of the analysis).

15.7.1 Running the Analysis in Parallel

15.8 Starting the Servers

See how many servers you currently have running with:

```
llama com do -D | wc -l
```

List all server names with:

```
llama com do -D
```

You should probably create a single server to begin. You can make sure that parallel operations work on this server before you save a DigitalOcean snapshot of it; create a single server using the command below and *start the analysis* for that single server to see if any bugs pop up. (If you haven't run the analysis in parallel in a while, there might be bugs; if you've made major architectural changes since the last background run, there are almost certainly bugs.)

```
llama com do -c llama-parallel-bns-o3-00 -i llama-parallel-bns-o3
```

Once you've debugged things on this server and gotten the analysis working, you can update the base image (llama-parallel-bns-o3 in this example) through DigitalOcean's web interface³³ from the working copy and spin up many more servers. Let's say you're limited to 100 servers and are already using a few, so you want to spin up 94 more instances named llama-parallel-bns-03-00 through llama-parallel-bns-03-94 (assuming you're keeping the image you created in the previous step, you only need to create the remaining 94 images):

```
llama com do -c llama-parallel-bns-o3-{01..94} -i llama-parallel-bns-o3
```

llama dev dv (and any vestigial pre-er-14-* CLI utilities that have not yet been ported to llama dev dv) are automatically vectorized, so the following commands should apply equally-well regardless of whether you are running one server or 1,000. Outside of edge-case bugs, things should run smoothly with many servers running once you've fixed things in the single-server case.

15.9 Running the Analysis

If everything has been set up properly, you can start a background run. First, add all of the existing DigitalOcean LLAMA droplets to your SSH known hosts list with:

```
llama dev dv init
```

Confirm that no current LLAMA processes are running with (you should see a total of 0 running processes at the end of the command):

```
llama dev dv ls-procs
```

Launch background runs for H1L1 (Hanford and Livingston) combined detectors with:

```
llama dev dv launch bg \
/home/vagrant/pre-er14-background//skymap-lists/bns-o3/H1L1.csv
```

Check to make sure processes are running:

```
llama dev dv ls-procs
```

See how many background run files are complete so far:

```
llama dev dv done bg
```

Print the most-recent lines of the most-recent logs from each server to see if anything interesting (read: bad) is happening (can also be useful to start debugging efforts)

```
llama dev dv tail
```

You might need to execute arbitrary code on one or more remote machines. Let's say you want to peak at the contents of the first background injection on llama-bns-o3-00. You can specify just that droplet and list the contents of a specific directory with:

³³ <https://cloud.digitalocean.com>

```
llama dev dv -p llama-bns-o3-00 eval \
'ls /home/vagrant/pre-er14-background/events/injection-0'
```

You can, of course, use this same method to delete stuff on remote machines, move stuff, or what-have-you. You can use other `llama dev dv` CLI flags to control things with more granularity. When used with the `eval` subcommand, `llama dev dv` is just a vectorized, multi-threaded (read: insanely fast) version of `ssh` that's aware of your DigitalOcean naming semantics; use it to make short work of remote commands that could otherwise be run using `ssh foo@your.server 'some command'`.

Now, assuming you've worked out any bugs and that things are flowing smoothly, sit back and wait until you've collected enough events for some level of statistical significance. Once you've amassed enough successful background runs, kill your LLAMA processes on all droplets with:

```
llama dev dv killall
```

Confirm all processes are dead (after a few moments):

```
llama dev dv ls-procs
```

15.10 Collecting Results

You should run until you have at least a few tens of thousands of events simulated. Check the progress of e.g. a background event simulation with:

```
llama dev dv done bg
```

Once you've simulated a sufficient number of events, you can pull the results down to your local computer. Anticipate around 1GB per 10,000 simulated events based on the pipeline architecture at time of writing. Pull results down and automatically organize them locally with:

```
llama dev dv pull
```

Next, create text tables with collated results from your simulations, **making sure to change the output filename to match the population you are working with**, by running the following command in the directory in which you just ran `llama dev dv pull`:

```
llama dev background table bns-o3-H1L1V1.txt
```

You will now have a table called `bns-o3-H1L1V1.txt` (or something similar based on the population you simulated). Look at the first 10 lines of this file and make sure they are somewhat reasonable looking:

EVENT-ODDS-RATIO	N	SOURCE-DIRECTORY	SKYMAP-FILENAME
+2.2912652264e-36	9	llama-bns-o3-65/events/injection-323	/home/vagrant/pre-er14- background/skymaps/bns-o3/8937.fits.gz
+2.673378655e-25	4	llama-bns-o3-65/events/injection-111	/home/vagrant/pre-er14- background/skymaps/bns-o3/2304.fits.gz
+1.677508305e-15	8	llama-bns-o3-65/events/injection-129	/home/vagrant/pre-er14- background/skymaps/bns-o3/4798.fits.gz
+9.3603728371e-09	7	llama-bns-o3-65/events/injection-116	/home/vagrant/pre-er14- background/skymaps/bns-o3/3400.fits.gz

(continues on next page)

(continued from previous page)

```
+9.233942133e-25 3  llama-bns-o3-65/events/injection-324 /home/vagrant/pre-er14-
˓→background/skymaps/bns-o3/1865.fits.gz
+1.565115763e-09 9  llama-bns-o3-65/events/injection-120 /home/vagrant/pre-er14-
˓→background/skymaps/bns-o3/699.fits.gz
+3.4337601072e-09 7  llama-bns-o3-65/events/injection-312 /home/vagrant/pre-er14-
˓→background/skymaps/bns-o3/8854.fits.gz
+9.4312345753e-09 4  llama-bns-o3-65/events/injection-118 /home/vagrant/pre-er14-
˓→background/skymaps/bns-o3/7545.fits.gz
```

It should look something like the above (the whitespace might be different; ignore that). Most critically, you should see some reasonable distribution of non-zero neutrino trigger list lengths (the N column) and different odds ratio values. The source directories and skymap filenames should also be different and reasonable looking.

Assuming this looks good, you can use these tables to generate the `populations.hdf5` test-statistic file using `llama dev background pvalue`. You can also zip up these simulation results and upload them to our DigitalOcean Spaces storage using `llama dev upload`.

DEVELOPER INSTALLATION

Developers can also use the pre-built LLAMA docker images to develop the pipeline, ensuring that everyone has working versions of the LLAMA developer environment with minimal effort.

16.1 Obtaining the LLAMA Source Code

First, pull the latest version of the code from BitBucket³⁴. You'll need to:

1. Make sure you have `git`³⁵ installed.
2. Make a BitBucket³⁶ user account.
3. *create some SSH keys on your computer*, which you'll use to
4. *authenticate with BitBucket* so that you can actually pull the LLAMA code.
5. If you haven't already, have Stefan or someone else with repository admin privileges add your BitBucket account to the LLAMA project so that you can access the source code.
6. Make a directory where you can store the source code. I recommend having a `~/dev/` directory in your home directory which contains all of your code projects. You can make such a directory with `mkdir ~/dev` if it doesn't already exist and navigate to that directory with `cd ~/dev`.
7. Finally, pull the code. If you've set everything up as described, the following line should work:

```
git clone git@bitbucket.org:stefancountryman/multimessenger-pipeline.git
```

8. You should now have the pipeline source code installed in a directory called `multipipeline`; if you called the last line in `~/dev`, then the pipeline will be stored in `~/dev/multipipeline`. Navigate to your new copy of the LLAMA source code with

```
cd multimessenger-pipeline
```

and confirm that it is a working git repository with

```
git status
```

which should print:

³⁴ <https://bitbucket.org>

³⁵ <https://git-scm.com/>

³⁶ <https://bitbucket.org>

```
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

If you see this, you've successfully installed the LLAMA source code.

16.2 Using the Docker Dev Image

There is a script in the source code (`bin/docker`) that can be used to very quickly and easily set up and use the LLAMA development docker image (`stefco/llama-dev`). `bin/docker` is a very simple wrapper script that just acts as a shortcut for the Docker commands you would need to do pipeline development. (Because of this, you can also use `bin/docker` to teach yourself some Docker syntax if you're interested; if not, just use the `bin/docker` commands listed here). Before you can use the `bin/docker` script, you'll need to install Docker using the [instructions in the Quick Start guide](#).

16.2.1 List `bin/docker` Commands

Once you've done this, you can list the available `bin/docker` commands by running the script with no arguments. Make sure you are in the LLAMA source code directory, and then run:

```
bin/docker
```

You should see:

```
USAGE: bin/docker CMD IMAGE

Shortcut for starting docker containers. Use this for
development or for running code from source. IMAGE is the name
of the credentials you want available on your docker image;
leave it blank to get no credentials, specify "play" to get
credentials for Slack, LVAAlert, and IceCube neutrino data
(readonly), or specify "production" for full production
credentials (Warning: this will allow you to do things like
upload to GraceDB or IceCube Slack!). These credentialled images
require you to be logged in to Docker with "docker login".
```

Valid CMD values and the shortcuts they correspond to:

```
dev
  eval "$DAEMON"
  docker exec -it ${CONTAINER} bash -l
daemon
  mkdir -p "${XDG_DATA_HOME:-$HOME/.local/share}"/llama
  if docker run --rm 2>/dev/null \
    --name ${CONTAINER} \
    -v "`pwd -P`:/root/multimessenger-pipeline \
    -v "${XDG_DATA_HOME:-$HOME/.local/share}"/llama:/root/.local/share/llama \
    -p 8080:8080 \
    -d ${IMAGE} bash -c "while true; do sleep 10; done" \
```

(continues on next page)

(continued from previous page)

```

"$@"
then
    echo "Created container ${CONTAINER}."
    echo "Installing LLAMA in dev mode with pip install -e ."
    docker exec ${CONTAINER} \
        bash -l -c "cd multimessenger-pipeline && pip install -e ."
else
    echo "Container ${CONTAINER} exists"
fi
pull
    docker pull ${IMAGE}
murder
    docker kill --signal=SIGKILL ${CONTAINER}

```

16.2.2 Getting the Latest Development Image

Assuming you have Docker up and running, have received access to the LLAMA Docker images on Docker Hub, and have logged in to Docker Hub using `docker login`, you should now be able to use the script to pull the latest LLAMA development image:

```
bin/docker pull
```

This might take a little while since the development image is a few GB. If you ever need to update to the latest development image, you can use that command again. Otherwise, if you already have it installed, the other commands won't automatically update you to the latest version.

If you're already running a LLAMA dev container and want to update it to the latest version, you can *remove that container* (after recording any changes you've made to the environment outside of LLAMA code updates, which persist automatically), pull the latest with `bin/docker pull`, and then *restart the container*.

16.2.3 Starting a Dev Container

Next, we can start up a developer container from the image we just pulled. It will automatically synchronize the contents of your LLAMA source code folder *as well as your LLAMA outputs folder** with the development container. This means that you can make quick fixes and process data using the development container and have that data available on your machine in the default location (`$XDG_DATA_HOME/llama` if `$XDG_DATA_HOME` is set, otherwise `~/.local/share/llama`). Start up the dev container with:

```
bin/docker dev
```

Or, if you want a container with access to IceCube data and the ability to upload to LLAMA Slack, run:

```
bin/docker dev play
```

Or, if you want full credentials, allowing you to get LIGO private data, upload to GraceDB, and post to IceCube Slack, get the dev container with production credentials using:

```
bin/docker dev production
```

This will start up the container in the background and log you into it; you should now see a promptline looking something like this (though the number after `root@` will be different):

```
root@052c44a5f430:~#
```

You are now logged into a fully functioning LLAMA development container (including, if you specified the play or production images, the necessary authentication credentials for people we communicate with); this is the same Docker image used to build the actual production LLAMA images! By default, you're the root user in the container, which means you can also try installing other software or doing other necessary sysadmin tasks in the container without breaking anything on your host laptop; if you manage to fix something in the development container that was previously broken, you should apply those same changes to the code that generates the LLAMA Docker images.

Now that you're in, you can confirm that the current development version of `llama` is installed and working with:

```
llama --version
```

Of course, this won't work if you've broken your copy of the source code, since the installed version is running directly off the source repository (even within the Docker dev container; this is **not** true of the `stefco/llama` containers, which have a static production copy of the software installed); if the above command doesn't work, you can stash away any changes to your working copy of the LLAMA repo by running `git stash` in the working directory.

Now, navigate to the source code from within the dev container:

```
cd ~/multimessenger-pipeline
```

You are now in a synchronized copy of the working copy of LLAMA on your laptop. This means you can write code on your laptop using whichever text editor you want and then test it in the Docker container; again, the installed version will reflect the current state of the source code.

16.2.4 Sketch of a Bug Fix

Let's say an event, S1234, just came in. You found a critical bug, fixed it in the source code, and then reran it on the dev container, then manually uploaded the data product from `~/.local/share/llama/current_run/S1234/` on your **local** computer (again, the LLAMA output directory is also synced between dev container and your computer) to wherever it needs to go. You should now `git add` your changes to the source code to the index and then `git commit` your changes before `git push`-ing them to the LLAMA repository, `git tag`-ing them with the next appropriate version (only increment the last decimal for a bugfix), and then pushing the tag to trigger automated builds and deployments with `git push --tags`. You can see a list of the latest versions with:

```
git tag -l | grep 'v[0-9]*\.[0-9]*\.[0-9]' | tail
```

16.2.5 Destroying your Dev Container

By default, your dev container will keep running in the background but will be destroyed when you shut it down manually. The LLAMA code is saved on your local computer, so you won't lose changes to that if your container is destroyed. Local changes to the container's *other* installed software will, however, be lost. This is mostly a *good* thing, since it means you can just destroy and restart your container if you accidentally break something. It does, however, mean that if you need to try a new set of software on the dev container, you will need to write down the steps you're following somewhere so that you can reproduce them if you have to destroy and restart the dev container. If the changes you are making to the LLAMA *environment* are bugfixes or necessary updates for the pipeline's functionality, you should also commit your changes to the LLAMA Docker image repositories; this will automatically rebuild the containers to feature your changes.

With that caveat out of the way, if you want to destroy your container (whether to *update to the latest dev image*, free up memory/disk space on your computer, or to start fresh and undo any changes you've made to the OS environment on your current container), you can run:

```
bin/docker murder
```

The somewhat dramatic naming is there to remind you that you are undoing any local changes to the container from the base image (other than the LLAMA software install) when you do this. Again, if you've only analyzed data or changed the LLAMA source code (and not changed the other installed software), you won't lose any work.

CHAPTER
SEVENTEEN

PREVIOUS LLAMA VERSIONS

This section contains deprecated or outdated installation instructions, and can be skipped if you just want to use the current version of the pipeline. It does, however, contain system administration and multi-messenger infrastructure information that can be useful for extending or fixing pipeline functionality, or for basic tasks related to working with multi-messenger data.

17.1 Developer Installation (Pre-O3)

NOTE: These instructions are for older versions of the pipeline (pre-O3). They are kept in place in case you need to work with those versions, or in case you need a starting point for e.g. a full IceCube stack installation. If you are working with the latest versions of the code, please read previous sections of the developer guide.

17.1.1 System Requirements

Make sure you have at least 4GB of memory (physical or virtual; `swap space` is fine) and 15GB of free space on your file system.

17.1.2 Introduction

To get up and running, start up a bash session and run each of the commands below. You should log in as a user other than root (more precisely, you should log in as the user who will run the LLAMA software in production; on a new install, you might need to [create a new user](#)). If you are installing LLAMA on a remote server (i.e. a computer besides the one that you are currently sitting in front of), you can read the [SSH with X11 Forwarding](#) section for instructions on how to connect to a remote server via SSH.

The below instructions will only work on Debian 8 (Jessie). You can adapt them to other platforms if you know what you are doing, but Debian 8 is the only supported platform. It should be possible to finish installation by blindly running commands, but even so, make sure to copy and paste each line one-at-a-time to make sure they are entered properly. Don't worry about the explanatory bits if you are not interested, but make sure to read parts that are in bold. You don't have to click any of the hyperlinks in this guide; they are for reference purposes only. If you encounter trouble, look in the [Appendix](#).

17.1.3 Installing LLAMA Dependencies

There are a few dependencies you will need for LLAMA installation and development. Install these now:

```
sudo apt-get -y update
sudo apt-get -y install python-sphinx rsync curl wget unzip git dtrx \
    msmtcp-mta heirloom-mailx xpdf debconf-utils make \
    apache2 apache2-doc apache2-utils ncdtack-grep silversearcher-ag \
    python3-six python3-pytest \
    python3-pytest-cov ipython3 pandoc libapache2-mod-python \
    latexmk python-profiler python-wxgtk3.0 python-setuptools runsnakerun \
    htop texlive-publishers
```

You will also want to install [git-lfs³⁷](#), an extension to `git` used for large file storage ([git-lfs install instructions taken from here³⁸](#)), which LLAMA uses to store large data files:

```
curl -s \
    https://packagecloud.io/install/repositories/github/git-lfs/script.deb.sh \
    | sudo bash
sudo apt-get -y install git-lfs
```

You must activate `git-lfs` on a per-user basis. **Make sure to run the following command as the user who will run the LLAMA server:**

```
cd ~
git lfs install
```

17.1.4 Obtaining the LLAMA Software (pre-O3)

LLAMA software is stored in a [git³⁹](#) repository with [git-lfs⁴⁰](#) used for large data file storage. You can obtain the entire LLAMA software repository with a quick `git clone` followed by a somewhat slower fetching of the large data files. First, though, you will want to [Generate SSH Keys](#) so that you don't need to enter your password over and over (SSH keys are actually required on [git.ligo.org⁴¹](#) and, at time of writing, on the [Bitbucket⁴²](#) multimessenger pipeline repository). See the [Bitbucket Authentication](#) and [git.ligo.org Authentication](#) sections for instructions on adding your SSH keys to those sites.

Once you've set up SSH keys, it's time to actually clone the LLAMA repository:

```
cd ~
git clone git@bitbucket.org:stefancountryman/multimessenger-pipeline.git
```

You might be prompted to confirm that you want to connect to Bitbucket; if you see something like the following, just type yes and hit enter:

```
The authenticity of host 'bitbucket.org (18.205.93.0)' can't be established.
RSA key fingerprint is 97:8c:1b:f2:6f:14:6b:5c:3b:ec:aa:46:46:74:7c:40.
Are you sure you want to continue connecting (yes/no)?
```

You can (pretty much) confirm that the LLAMA software folder was created by running:

³⁷ <https://git-lfs.github.com/>

³⁸ <https://github.com/git-lfs/git-lfs/wiki/Installation#debian>

³⁹ <https://git-scm.com/>

⁴⁰ <https://git-lfs.github.com/>

⁴¹ <https://git.ligo.org>

⁴² <https://bitbucket.org>

```
cd multimessenger-pipeline && git status; cd ~
```

If you see the following response:

On branch master
 Your branch is up-to-date with ‘origin/master’.
 nothing to commit, working directory clean

then you have (most likely) succeeded in obtaining the LLAMA codebase.

17.1.5 Setting up Configuration Files

You must now move the default configuration files into place (*this affects things like gw-astronomy.org⁴³ access and email sending*):

```
ipython profile create
ln -s \
    ~/multimessenger-pipeline/static/llama-ipython-startup.py \
    ~/.ipython/profile_default/startup/llama-ipython-startup.py
ln -s multimessenger-pipeline/static/.gitattributes ~
ln -s multimessenger-pipeline/static/.inputrc ~
ln -s multimessenger-pipeline/static/.mailrc ~
cp multimessenger-pipeline/static/.msmtprc ~
cp multimessenger-pipeline/static/.netrc ~
cp multimessenger-pipeline/static/.twilio-credentials ~
ln -s multimessenger-pipeline/static/.vimrc ~
mkdir -p ~/.ssh
ln -s multimessenger-pipeline/static/.ssh/config ~/.ssh
chmod 0600 ~/.msmtprc
touch ~/.bashrc
[ -e ~/.bashrc.orig ] || cp ~/.bashrc ~/.bashrc.orig
cat \
    ~/.bashrc.orig \
    ~/multimessenger-pipeline/static/.bashrc-llama-addendum \
    >~/.bashrc
. ~/.bashrc
```

You should see “LLAMA” appear in big letters accross your terminal. You should also now be able to access the main LLAMA executable, `llama`, which contains all user features as subcommands, as well as developer scripts, since they should all now be part of your shell’s `PATH` variable.

Now, you will want to preseed your Kerberos⁴⁴ and HTCondor⁴⁵ preferences to avoid being bugged later by debconf⁴⁶ during your eventual Kerberos installation:

```
sudo debconf-set-selections ~/multimessenger-pipeline/static/preseed.cfg
```

⁴³ <https://gw-astronomy.org>

⁴⁴ <https://web.mit.edu/kerberos/>

⁴⁵ <https://research.cs.wisc.edu/htcondor/description.html>

⁴⁶ <https://wiki.debian.org/debconf>

17.1.6 Install git Hooks

You can have the server automatically regenerate documentation and home page when they are modified by installing the included git hooks. From the repository directory, run:

```
for h in git-hooks/*; do ln -fs ../../$h .git/hooks; done
```

17.1.7 Install LLAMA dependencies

You can install (mostly non-LIGO) LLAMA dependencies from the `requirements.txt` file:

```
curl -O https://raw.githubusercontent.com/stefco/llama-env/master/requirements.txt
pip install -r requirements.txt
rm requirements.txt
```

17.1.8 Install LIGO Software

LIGO supports Scientific Linux, Debian, and (on a best-effort basis) OS X. The below instructions are for Debian 8 (Jessie).

First, we will need to install LIGO dependencies. The first step is adding LIGO's package repositories to Debian's list of sources so that Debian's package manager knows where to find the LIGO software packages we are about to install. This simply requires copying the source list into Debian's source list directory. *Further documentation available on LIGO's Software Downloads page*⁴⁷.

```
sudo cp ~/multimessenger-pipeline/static/lscsoft.list \
/etc/apt/sources.list.d/lscsoft.list
```

The following line should prevent the installation process from asking for user feedback. This is useful if you want to run installation in the background.

```
export DEBIAN_FRONTEND=noninteractive
```

You should always update your package manager's list of repositories before trying to install something new:

```
sudo apt-get -y -qq update
```

You will need to tell the package manager to trust LIGO software. **You might still get warnings about untrusted software packages; this is fine.**

```
sudo apt-get install -y -qq --force-yes lscsoft-archive-keyring
```

Update again.

```
sudo apt-get -y -qq update
```

Finally, install `lscsoft-all`, the comprehensive LIGO software package.

```
sudo apt-get install -y -qq --force-yes \
-o Dpkg::Options::="--force-confdef" \
-o Dpkg::Options::="--force-confold" \
lscsoft-all lalinference
```

⁴⁷ <https://wiki.ligo.org/DASWG/SoftwareDownloads>

Next, install ligo datagrid; *this usually fails the first time and works the second time for some reason. If it fails, the below command will keep retrying until it succeeds, which generally seems to work.*

```
retval=1
until [ $retval -eq 0 ]; do
    echo 'ATTEMPTING TO INSTALL DATAGRID'
    curl https://www.lsc-group.phys.uwm.edu/lscdatagrid/doc/ldg-client.sh \
        >/tmp/ldg-client.sh \
        && sudo bash /tmp/ldg-client.sh
    retval=$?
done
```

Once the script is finished executing, you can confirm that installation was successful by running:

```
type ligo-proxy-init >/dev/null 2>&1 \
    && { echo "LIGO Data Grid Installation succeeded."; } \
    || { echo "LIGO Data Grid Installation failed! Try again."; }
```

to check for one of the installed executables. This command will tell you whether installation was successful (again, if installation failed, just try repeating the previous step).

Finally, install miscellaneous python dependencies using pip. *One of these dependencies, gwpy, is in active development and can be a bit finicky. The below instructions should “just work”, but if not, further documentation is available on GWPy’s install instructions page⁴⁸.*

Tell Debian to use the default locale (otherwise errors come up in the scipy update):

```
export LC_ALL=C
```

Update the python package installer pip:

```
sudo pip install --upgrade pip
```

Update scipy to a version recent enough for GWpy to work (the next line *might* take a while to finish and *might* produce a lot of ugly output depending on your precise version of Debian Jessie; don’t be alarmed if this is the case.):

```
sudo pip install 'scipy>=0.16'
```

Install GWpy:

```
sudo pip install gwpy
```

17.1.9 Install IceCube Offline Software

These packages are used for retrieving neutrino data from IceCube.

⁴⁸ <https://gwpy.github.io/docs/stable/install.html>

17.1.9.1 Installing IceCube Dependencies

First you'll need to make sure you're using a newer version of `cmake` than is available on Debian Jessie; do this by adding more recent backports to the `apt` sources list:

```
sudo cp ~/multimessenger-pipeline/static/backports.list \
    /etc/apt/sources.list.d/backports.list
```

Now you can install the latest `cmake` version by forcing `apt-get` to use the `jessie-backports` repository:

```
sudo apt-get -t jessie-backports install cmake
```

Now, install dependencies⁴⁹ for the IceCube offline software (with all of the recommended extras):

```
apt-get install build-essential cmake libbz2-dev libgl1-mesa-dev \
    freeglut3-dev libxml2-dev subversion libboost-python-dev \
    libboost-system-dev libboost-signals-dev libboost-thread-dev \
    libboost-date-time-dev libboost-serialization-dev \
    libboost-filesystem-dev libboost-program-options-dev \
    libboost-regex-dev libboost-iostreams-dev libgsl0-dev libcdk5-dev \
    libarchive-dev python-scipy ipython-qtconsole libqt4-dev python-urwid \
    libz-dev libqt5opengl5-dev libstarlink-pal-dev python-sphinx \
    libopenblas-dev libcfitsio3-dev libsprng2-dev libmysqlclient-dev \
    libsuitesparse-dev libcfitsio3-dev libmysqlclient-dev \
    libhdf5-serial-dev root-system
```

17.1.9.2 Entering IceCube Credentials

Next, store your IceCube username and password in the `uname` and `pword` variables; we will use this to check out the IceCube repository. You should have received IceCube login credentials as part of the installation procedure. **The next two lines of commands will read your IceCube username and password in and store them as variables. If you make a mistake entering your password, or if something goes wrong while installing IceCube software, you should run these lines again just to make sure that your credentials are being passed to SVN in the next step.**

```
read -p 'Enter your IceCube username: ' uname && \
read -sp 'Enter your IceCube password: ' pword && echo
```

Next, check out IceCube offline software. These instructions are adapted from the [original IceCube offline software installation instructions](#)⁵⁰; you can find all [offline software releases](#) here⁵¹. A full overview of IceCube software is available [here](#)⁵². Make sure to check out the latest version (if it is not the same as the one listed in the following code block). *The while loop is included to catch and handle segmentation faults during checkout; these are, unfortunately, a known bug*⁵³ *in Debian Jessie's SVN version.*

```
sudo mkdir -p /usr/local/icecube/offline
cd /usr/local/icecube/offline
export IceCube_SVN=http://code.icecube.wisc.edu/svn
sudo svn --username "$uname" --password "$pword" co \
    "$IceCube_SVN"/meta-projects/offline-software/releases/V18-06-00 src
```

⁴⁹ http://software.icecube.wisc.edu/documentation/projects/cmake/supported_platforms/debian_variants.html

⁵⁰ <http://software.icecube.wisc.edu/documentation/info/quickstart.html>

⁵¹ <https://code.icecube.wisc.edu/projects/icecube/browser/IceCube/meta-projects/offline-software/releases>

⁵² <http://software.icecube.wisc.edu/documentation/index.html>

⁵³ <https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=736879>

You should run the next code block to make sure that the SVN checkout succeeded in full.

```
retval=$?
while ! [ $retval -eq 0 ]; do
    echo 'QUIT EARLY DUE TO SEGFAULT IN SVN, RESTARTING CHECKOUT'
    sudo svn cleanup src
    sudo svn update src
    retval=$?
done
```

Now it is time to build the IceTray software. *This next step is very slow; let this run for a couple of hours while you go do something else. Fortunately, this step can be resumed at the make command in the event of failure.*

```
sudo mkdir -p /usr/local/icecube/offline/build
cd /usr/local/icecube/offline/build
sudo cmake -DSYSTEM_PACKAGES=True ./src
sudo make
```

If you get an error while running cmake or make, there was probably an error that prevented you from fetching the entire software repository from SVN. You should go back a few steps and start over from when you entered your IceCube credentials a few steps ago. Finally, download some data used by IceCube test and example scripts.

```
sudo make rsync
```

The .bashrc modification you made *at the beginning of this whole procedure* has already put the necessary modifications to your environmental variables in place (e.g. your PYTHONPATH variable has been updated to include the location of the new IceCube python libraries, so that you can import them next time you run python). You can therefore test whether the IceCube python software was installed successfully:

```
python -c 'from icecube import dataclasses; print("Success!")'
```

You should see Success! printed on your console. If so, congratulations! You have successfully installed the IceCube offline software. Now, you will have to check out a set of neutrino querying python tools used by LLAMA. Change to the LLAMA software directory and check out the live software. Again, store your IceCube username and password in variables (you can skip this step if those variables are still initialized from before):

```
read -p 'Enter your IceCube username: ' uname && \
read -sp 'Enter your IceCube password: ' pword && echo
```

Now, check out the live querying software:

```
cd ~/multimessenger-pipeline/icecube
export IceCube SVN=http://code.icecube.wisc.edu/svn
svn --username "$uname" --password "$pword" co \
    "$IceCube SVN"/projects/realtime_tools/releases/V19-02-00/python \
    realtime_tools
svn --username "$uname" --password "$pword" co \
    "$IceCube SVN"/projects/realtime_gfu/releases/V19-02-00/python \
    realtime_gfu
pushd realtime_gfu
svn --username "$uname" --password "$pword" co \
    "$IceCube SVN"/projects/realtime_gfu/releases/V19-02-00/resources \
    resources
popd
```

(continues on next page)

(continued from previous page)

```
# this is a kludge to get the realtime_gfu resources in place
sudo cp -R \
~/multimessenger-pipeline/icecube/realtime_gfu \
/usr/local/icecube/offline/build/
```

17.1.10 Install MATLAB

You will need to download the MATLAB Installer⁵⁴ as well as the JSONLab toolbox⁵⁵. You must install the following three MATLAB Toolboxes at the same time that you install MATLAB, and you must make sure to also install MATLAB itself. Do so by selecting the following four items in the install dialog (note that the MATLAB version might change from 9.1, but it should be at the top of the list):

1. MATLAB 9.1
2. Image Processing Toolbox
3. Mapping Toolbox
4. Statistics and Machine Learning Toolbox

Also make sure to check the box in the install dialogue asking if you would like to make symlinks to MATLAB. This is necessary in order for command line calls to matlab to work (which is necessary for our scripts to run in the current iteration of the pipeline). If you forget to do this, you can manually add the symlink in with:

```
sudo ln -s /usr/local/MATLAB/*/bin/matlab /usr/local/bin/matlab
```

You can confirm that the symlink was made by running:

```
type matlab 2>/dev/null 1>&2 && echo 'success!' || echo 'symlink not found!'
```

You can install the JSONLab toolbox from within the MATLAB interface after it has been successfully installed. See the appendix for some [MATLAB installation details](#). You will usually have to install JSONLab twice before it remains permanently installed. It is not clear why this is, but it works after the second installation. You can confirm that JSONLab is installed by opening a MATLAB session and seeing if loadjson is a valid command; if it is, then JSONLab has been successfully installed. You can also simply run this command to see if JSONLab installed successfully:

```
matcmd="""
matcmd+="if exist('loadjson')"
matcmd+="    disp('JSONLab installed successfully.');
matcmd+="else"
matcmd+="    disp('JSONLab failed to install.');
matcmd+="end;
matcmd+="exit
matlab -nodesktop -nosplash -noFigureWindows -r "${matcmd}"
```

If JSONLab failed to install, just open up MATLAB and reinstall it. This is sometimes necessary for some reason.

Now, install the matlab python module, which will allow python code to call MATLAB scripts via the [MATLAB Engine API](#)⁵⁶:

⁵⁴ https://www.mathworks.com/downloads/web_downloads/download_release?release=R2016b

⁵⁵ <https://www.mathworks.com/matlabcentral/fileexchange/33381-jsonlab--a-toolbox-to-encode-decode-json-files>

⁵⁶ http://www.mathworks.com/help/matlab/matlab_external/get-started-with-matlab-engine-for-python.html

```
cd $(sudo find / -path '\*/extern/engines/python')
sudo python setup.py install
```

Confirm that installation of the Python MATLAB Engine API succeeded:

```
python -c 'import matlab; print("success!")'
```

17.2 Configuration and Authentication

These steps involve entering your credentials for the services used by LLAMA, as well as some basic configuration. These instructions assume that you have LIGO login credentials. *Some of the steps require you to contact one of our partners (e.g. GCN or gw-astronomy.org) to configure authentication. These appear after this section in the external authentication section.*

17.2.1 Generate SSH Keys

SSH keys are used to securely log in to other computers without a password. LLAMA uses them to upload data to gw-astronomy.org⁵⁷. A great and user-friendly guide to setting up SSH keys is available on Digital Ocean's blog⁵⁸. *These instructions are pulled from that article.*

First, check whether you have an SSH key already created. The following command will print “no SSH key found” if it doesn’t find an existing key:

```
[ -e ~/.ssh/id_rsa.pub ] || echo "no SSH key found."
```

If you don’t already have an SSH key, create a new one by running the following command and hitting enter repeatedly until it finishes (**WARNING: this will overwrite your existing SSH key if you do in fact have one already, possibly costing you access to servers you are currently able to SSH into.**):

```
ssh-keygen
```

You should see some funny ASCII art print out once it’s finished. You can now print your SSH key with:

```
cat ~/.ssh/id_rsa.pub
```

You will eventually have to make sure that the public key is on gw-astronomy.org⁵⁹,’s list of authorized keys; this step is discussed below in the external authentication section.

17.2.2 GMail Authentication

1. Create a device (or “app”, as Google calls it) password for gw.hen.analysis@gmail.com (or your personal gmail, if you are just trying to run some tests). This password is different from your login password and is only to be used by a single device. You can generate a device password [here](#)⁶⁰.
2. Use your favorite text editor to open `~/.msmtprc` and replace `<WRITE-PASSWORD-HERE>` with your newly-created device password. *If you are using a personal gmail account (rather than gw.hen.analysis@gmail.com), you will want to find each occurrence of `gw.hen.analysis@gmail.com` in this file and replace it with your personal gmail address.*

⁵⁷ <https://gw-astronomy.org>

⁵⁸ <https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys--2>

⁵⁹ <https://gw-astronomy.org>

⁶⁰ <https://security.google.com/settings/security/app passwords>

17.2.3 LIGO Authentication

1. Run `ligo-proxy-init` `your.username` and enter your LIGO password. This will allow you to download GW event data from GraceDB. *You will need to do this every few days.*

LV_Alert Authentication

1. Make sure you have an LVAAlert username and password set up. You can do this [here⁶¹](#). You can learn more about LVAAlert on the [official documentation pages⁶²](#) and from the GraceDB [LVAAlert guide⁶³](#).
2. Use your favorite text editor to open `~/.netrc` and replace `<WRITE-USERNAME-HERE>` with the LVAAlert username you created in step 1 and `<WRITE-PASSWORD-HERE>` with the password for that LVAAlert username. If you already have a password but forgot it, you can easily reset it at the link provided in step 1.
3. (*Optional*) You can make sure that your credentials work by running

```
lvalert_admin --username albert.einstein --subscriptions
```

where, of course, ``albert.einstein`` is replaced with the LVAAlert username you made [in](#) step 1.

17.2.4 Setting Up SSL Certificates

In order to access the server using the HTTPS protocol (as is generally considered best practice, since this will traffic to/from the server when accessing it via browser), you'll need to get an SSL certificate. The easiest way to do this is by using the [EFF Certbot⁶⁴](#) (linked instructions for Debian Jessie, but just use the correct ones for whatever system is running the server) and using [Let's Encrypt⁶⁵](#).

These instructions will let you install `certbot-auto` from EFF's website. `certbot-auto` can automatically install certificates for an Apache server:

```
sudo /usr/local/bin/certbot-auto --apache
```

You can then automate renewal of the certificate (important because the certs expire after 90 days) by adding the following to your `systemd` timer or root crontab:

```
/usr/local/bin/certbot-auto renew
```

17.2.5 Setting Up Passwords for Run Summary Pages

You can put the username and password for the current run in `~/.llama_run_credentials` in `.netrc` format. This will be used for the run summary pages when basic HTTP authentication is in use with the `flask` dev server.

⁶¹ <https://www.lsc-group.phys.uwm.edu/cgi-bin/jabber-acct.cgi>

⁶² <https://wiki.ligo.org/DASWG/LVAAlertHowto>

⁶³ <https://gracedb.ligo.org/documentation/lvalert.html>

⁶⁴ <https://certbot.eff.org/lets-encrypt/debianjessie-apache>

⁶⁵ <https://letsencrypt.org/getting-started/>

17.3 External Authentication

The services referenced in this section require credentials provided by LLAMA partners and commercial services. Authenticating with these services is not necessary for testing the majority of the LLAMA pipeline's functionality, but it does require either expenditure of funding (for Twilio) or getting help from our busy partners (for gw-astronomy.org and GCN). These are live services that are used to disseminate results, and testing their functionality requires care and coordination. **It is therefore highly recommended that this section be skipped by all users who are not actively deploying the LLAMA code into production.**

17.3.1 Bitbucket Authentication

At time of writing, the main software repository is stored on [Bitbucket](#)⁶⁶.

The first step is to [Generate SSH Keys](#); once you've done that, you can access your SSH key by logging in to your LLAMA server and printing out your **public** key:

```
cat ~/.ssh/id_rsa.pub
```

You can now copy this key straight from your terminal and add it to Bitbucket. First, navigate to [Bitbucket](#)⁶⁷'s website, followed by your user settings page, and then click the *SSH keys* section under *Security*. At time of writing, you can also directly access these settings at the following URL, where **USERNAME** has been replaced with your bitbucket username:

```
https://bitbucket.org/account/user/USERNAME/ssh-keys/
```

Once there, click *Add key* and type the name of your server under *Label* and paste the SSH key into the *Key* section. Hit *Add key* to finish the process. You should now be able to push and pull from your Bitbucket repositories at the command line on your server without needing to enter your username and password, and if you have permission to access the pipeline repository, you will now be able to clone it from its Bitbucket URL.

17.3.2 git.ligo.org Authentication

The first step is to [Generate SSH Keys](#); once you've done that, you can access your SSH key by logging in to your LLAMA server and printing out your **public** key:

```
cat ~/.ssh/id_rsa.pub
```

You can now copy this key straight from your terminal and add it to [git.ligo.org](#). Visit your [git.ligo.org SSH-key settings page](#)⁶⁸. Once there, type the name of your server under *Title* and paste the SSH key into the *Key* section. Hit *Add key* to finish the process. You should now be able to push and pull from your [git.ligo.org](#)⁶⁹ repositories at the command line on your server without needing to enter your username and password, and if you have permission to access the pipeline repository, you will now be able to clone it from its Bitbucket URL.

⁶⁶ <https://bitbucket.org>

⁶⁷ <https://bitbucket.org>

⁶⁸ <https://git.ligo.org/profile/keys>

⁶⁹ <https://git.ligo.org>

17.3.3 Twilio Authentication

We use [Twilio's⁷⁰](#) API to send SMS messages to LLAMA operators for human-in-the-loop parts of the pipeline. To authenticate with Twilio, you will need to provide your credentials as environmental variables. The `.bashrc` file provided by default will try to run `~/.twilio-credentials`, so the easy way to do this is to put your credentials in that file. That file is pre-populated with the current LLAMA Twilio phone number, so you don't have to provide that.

1. Go to [Twilio's dashboard⁷¹](#) and click "Show API Credentials" in the top right corner of the screen. Take note of the *Account SID* and *Auth Token*.
2. Open up `~/.twilio-credentials` with your preferred text editor
3. Replace `<YOUR-SID-HERE>` with the *Account SID* you just got from Twilio's website. Replace `<YOUR-TOKEN-HERE>` with the *Auth Token* you just got and save the file.
4. You will need to reload your credentials or your `~/.bashrc` file for the changes to take effect. You can reload just the credentials by running:

```
source ~/.twilio-credentials
```

17.3.4 GW Astronomy Authentication

Make sure that the `llama` user on the [gw-astronomy.org⁷²](#) server has your public key saved to its `authorized_keys` file. If you have not set this up yet, you will need to email the webmaster at [gw-astronomy.org⁷³](#) and send your public key, `~/ssh/id_rsa.pub`, so that the webmaster can authorize you to log in to LLAMA's [gw-astronomy.org⁷⁴](#) account. This is necessary for uploading analysis results. *If you don't know how to generate SSH keys, refer to the instructions showing how to Generate SSH Keys above.*

17.3.5 GCN Authentication

Make sure that you have an entry for this server in the [GCN⁷⁵](#) network's [Sites Configuration File⁷⁶](#). This is necessary for receiving LVC events. In particular, you need to make sure there is a site entry containing your server's IP address (which **must** be static) under the `DIST_ADDRESS` as well as a properly set `LVC_Enabled` field. You should also make sure you are set up to receive the proper alerts (including `LVC_INITIAL_POS_MAP` and `GWHEN_COINC` at the time of writing).

17.4 Turning on the Pipeline

17.4.1 Subscribing to LVAAlert Nodes

Make sure you have subscribed to all of the LVAAlert nodes used by the pipeline by running:

```
lvalert_admin -i
```

You should see all of the following (ordering does not matter):

⁷⁰ <https://www.twilio.com>

⁷¹ <https://www.twilio.com/console/sms/dashboard>

⁷² <https://gw-astronomy.org>

⁷³ <https://gw-astronomy.org>

⁷⁴ <https://gw-astronomy.org>

⁷⁵ <http://gcn.gsfc.nasa.gov/>

⁷⁶ http://gcn.gsfc.nasa.gov/sites2_cfg.html

```
Node: superevent [ subscribed subid=k7hjUhAX0qLiLjXL6bCnXDPT1JngsDwFmZvCdGqp]
Node: cbc_gstlal [ subscribed subid=DRPc5g1ivFNn504a5uQk2mzHVdarHWVzxD6S3u8Y]
Node: cbc_lowmass [ subscribed subid=DGYiTkwsl58w3fwsFX72g2pIqERg0jfEY7fW9Bop]
Node: stc-testnode [ subscribed subid=7lzckGSYHH28wswE4R18JnlpncKi9SsL8u0JZzZa]
Node: cbc_pycbc [ subscribed subid=CDQriRVw7IsYU6Uh2Y738oHlwPnXkJJvdD9PWeTQ]
Node: cbc_mbtaonline [ subscribed subid=cfiMONixdKRnl2DmSupPSD0BSArF3PasuoMAPcFU]
Node: burst_cwb [ subscribed subid=Lp0fqpgwdnNNn5XnRW9jNPCn7UZuxmtYJW7j1S7Z]
Node: test_superevent [ subscribed subid=T8W2oLnNMC8TUhwSdBERJEG0My8VG1JEcnGMWFv]
Node: cbc_spiir [ subscribed subid=0H4qi2Fh5Uo7OnLjPaT1aydQLKiNdj0gxzmv5TXB]
```

If any of the above nodes are missing, add them with `llama_lvalert_subscribe`, which just calls `lvalert_admin --subscribe --node ...` on every node we want to listen to. It's really just a shortcut for the following:

```
lvalert_admin --subscribe --node superevent
lvalert_admin --subscribe --node cbc_gstlal
lvalert_admin --subscribe --node cbc_pycbc
lvalert_admin --subscribe --node cbc_mbtaonline
lvalert_admin --subscribe --node cbc_spiir
lvalert_admin --subscribe --node cbc_lowmass
lvalert_admin --subscribe --node stc-testnode
lvalert_admin --subscribe --node burst_cwb
lvalert_admin --subscribe --node test_superevent
```

(Note: you can also always print a full list of available LVAAlert nodes by running `lvalert_admin -m`.)

17.4.2 Starting Pipeline Daemons

Once you've subscribed to all of the above LVAAlert nodes, navigate to the repository directory and run:

```
llama run
llama listen lvalert
llama listen gcn
nohup serve_current_run 1>/dev/null 2>&1 </dev/null &
```

These commands will start:

1. The LVAAlert listener, which listens for new LVC events and uses them to create new event directories when skymaps become available; mark events as `EM_READY` when they have been vetted as ready for EM follow-up; and marks them as having a `superevent`⁷⁷ associated with them, indicating that the results of a specific event can be distributed to the public under the name of that superevent.
2. The GCN Notice listener, which listens for new LVC events both as confirmation that the event is public (and hence LLAMA can send out an alert) and as a fallback to the LVAAlert listener (in case something non-standard happened in creating the superevent—as has happened in the past with unusual events—which was corrected when the public-facing GCN Notice was sent out).
3. The LLAMA pipeline daemon (`llama run`), which checks the default event directory for *non-test* events (hence the filter for GraceIDs starting with “G”). When the GCN listener receives a new LVC VOEvent via a GCN Notice, the LLAMA pipeline daemon will detect the new VOEvent and start running the LLAMA joint analysis.
4. The Current Run status page server (`serve_current_run`), which serves a summary webpage with the current status of the LLAMA server and the events processed as part of the current run over port 5000. Assuming that the

⁷⁷ <https://emfollow.docs.ligo.org/userguide/>

server is associated with the domain name `gwhen.com`, one can view the current status of the server (including memory usage and whether daemon processes are running) by visiting `gwhen.com:5000`⁷⁸.

(The `< /dev/null` part is necessary due to some strange behavior in MATLAB when running in the background using `nohup`.)

17.4.2.1 Testing LVAAlert

Note that you can run a quick test to see if you're connected to LVAAlert by starting up `lvalert_listen` (as described above) and submitting a test event in GraceDB's LVAAlert JSON format using:

```
lvalert_send -v --node stc-testnode \
    --file ~/multimessenger-pipeline/static/mock_event.json
```

You should see the contents of that mock event show up in the LVAAlert handler's logs, and if there was some intended side-effect of that LVAAlert, you should also see some verbose logging and a new/modified event in the default event directory (`~/.local/share/llama/current_run/`). Note that, because the test event was sent on `stc-testnode`, the event will be flagged as a test event, and though the pipeline should run, no alerts will be sent to the public.

17.4.3 Viewing Active Processes

You can see if these processes are running with:

```
llama run -R          # find ``llama run`` processes
llama listen lvalert -R      # show lvalert listener processes (if any)
llama listen gcn -R        # show GCN listener processes (if any)
ps -ax | grep serve_current_run # find serve_current_run process
```

17.4.4 Checking Pipeline Logs

You can follow the log output in real time with:

```
lvalertd -t
tail -f logs/llamad.log
tail -f logs/serve_current_run.log
```

In general, all LLAMA pipeline output will be saved to the `logs/` directory. This is true regardless of where you place the repository directory.

17.4.5 Killing Pipeline Daemons

You can shut down the pipeline processes with:

```
llama run -k
llama listen lvalert -k
llama listen gcn -k
ps -ax | grep serve_current_run | sed /grep/d | awk '{print $1}' | xargs kill
```

⁷⁸ <http://gwhen.com:5000>

17.4.6 Deprecated: MATLAB Logs

When running the pipeline with MATLAB code, all MATLAB-specific logging should be viewable by running:

```
tail -f logs/llama.matlab.log  # matlab process output goes here
```

17.5 Developing for LLAMA

This section is intended for new LLAMA developers.

17.5.1 Introduction to Development

17.5.1.1 Structure of the Pipeline

The LLAMA pipeline is designed to be flexible, robust, reproducible, parallel, and easy to develop. This is accomplished by treating data processing as a bunch of methods for processing input data and producing output data independently of other parts of the analysis. LLAMA defines a python class called a `FileHandler` that specifies at minimum the following information about a data file:

1. The filename
2. The complete list of input files required to generate this file
3. The code used to actually generate this file from its inputs

Thus, each file is connected to the list of files necessary for its own generation, forming a file dependency graph (an instance of a Directed Acyclic Graph, or DAG, as used in other data analysis pipeline tools). When a file's dependencies are available, it will be generated without affecting neighboring parts of the analysis, and each step of the pipeline is reproducible using the saved files (which are the only stateful part of the pipeline).

17.5.1.2 Adding Functionality

This approach makes it very easy to add new data processing capabilities to LLAMA. Just follow these steps:

1. Add a new python file in `/llama/files` that defines a subclass of `abstract.FileHandler`. You will have to implement a few abstract methods and properties (see e.g. `llama/files/lvc_skymap_mat.py` for a simple example of a `FileHandler` implementation):
 - The `filename` property, which gives the name of the new output file
 - The `dependencies` property, which returns a list of `FileHandler` classes for each input file required to generate this file (make sure to import those `FileHandler` classes!)
 - The `_generate` method, which defines the steps needed to generate this file using input data. You can access things like the full path to a dependency file by instantiating a copy of its `FileHandler` and getting the `fullpath` property using e.g. `other_file_handler_class(self).fullpath`.
2. Make sure that your new `FileHandler` is included in the actual pipeline by importing the class in `/llama/files/__init__.py`
3. Test that the pipeline runs and produces desired output by running one of the developer tests, e.g. `make testpipeline7` (or whatever test you feel like using in the makefile). Check that the output data is what you would expect for your new file by looking in the event directory for this test (`/testout/events/testpipeline7` in this example).

4. If your new file looks good, run `nosetests` in the root repository directory to make sure that you didn't somehow break anything.
5. If `nosetests` pass, `git commit` your code and push it to the remote repository.
6. Last but not least, **remember to restart the pipeline** in order to see your updates actually applied; the pipeline does not refresh its code automatically (a good thing for developing). You can do so with the following commands:

```
llama run -k # might need to run this twice  
llama run
```

17.5.2 Developer Conventions and Best Practices

Please follow the following standards with code written for the LLAMA pipeline:

17.5.2.1 Data Format Conventions

- All event files must represent angles in *degrees*. This means that, when querying external APIs, you *must convert radians to degrees* before saving data in an event directory. (You should log any raw data you throw out so that we can check the correctness of the conversion; see point on logging below).

17.5.2.2 Coding Best Practices

- Don't hard code filenames or other magic strings! If you need to get data from a file, make sure that that file has its own FileHandler defining how it can be (*reproducibly!*) generated, then import that FileHandler class, instantiate a copy by feeding the current FileHandler as an argument to the dependency FileHandler class, and get the data you want (like filename) from that instance.
- Log all actions. This can be done very easily by sticking to the FileHandler subclass idiom defined by LLAMA.
- Please verbosely log any incoming raw data before it is thrown out. This way we can go back and see if we accidentally deleted good pulled data.

17.5.3 Testing

Go into the root repository directory (i.e. the top level directory of the git repository, which is assumed to be `~/multimessenger-pipeline` at the time of writing of this documentation) and run:

```
make testgracedb
```

to make sure that you are properly authenticated using your LIGO credentials. You should see the following output (or something like it) printed to console (**note that the following is output; don't type it into the console**):

```
llama/tests/testgracedb.py | tee -a logs/llama.testgracedb.log  
running testgracedb at 2016-09-13 18:32:05.577660  
Attempting to connect to GraceDB.  
Attempting to download the boxin day even event.log.  
Received successfully. Logging contents.  
Pipeline: gstlal  
Search: HighMass  
MChirp: 9.555
```

(continues on next page)

(continued from previous page)

```
MTot: 26.3501410484
End Time: 1135136350.647757924
SNR: 11.710
IFOs: H1,L1
FAR: 3.333e-11
```

If your output does not match the above, then you most likely need to log in using your LIGO credentials (See [LIGO Authentication](#)).

Next, run the full collection of unit tests using:

```
nosestests
```

This will take a few minutes and will not produce much output while it is running. If all tests succeed, you will see the number of tests that were run and the status message OK. You will be alerted at the end if any of the tests fail or if the tests themselves cannot be completed due to errors. Be aware that, if any of the tests fails, the output files will remain in /tmp taking up a fair amount of space. They are hard links by default, so this shouldn't take up much extra space, but it is worth deleting directories of the form /tmp/tmp* after a failed run (after making sure you don't have other data matching that file pattern, of course).

17.6 Appendix

17.6.1 Migrating to Conda

Migrating to Conda should simplify some LIGO dependency issues. It might make it harder to use IceCube software (have not checked whether this supports Conda packaging yet), but this is not currently an issue since we have IceCube stub methods installed.

Note that, at time of writing, LVAlert had not migrated to Conda, and so the old installation method is still required to be able to use LVAlert.

17.6.1.1 LIGO Wiki Documentation

The latest documentation on installing from source should always be available from the [LSCSoft Conda](#)⁷⁹ documentation page. The instructions below are pulled directly therefrom. We can also follow LSC's guide for [making a new Conda package](#)⁸⁰ if we want to make the pipeline distributable by Conda in the future.

17.6.1.2 Installing LIGO Software via Conda

Conda installs are done on a per-user basis, so you won't need to use sudo for any of the below. Start by installing the latest version of Conda:

```
curl -O https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
```

You'll now need to exit your SSH session and log in again for conda to show up as a command; run `exit` to exit and then SSH back into the server. Now, activate your Conda environment and add the `conda-forge` channel (used to distribute custom Conda packages):

⁷⁹ <https://docs.ligo.org/lscsoft/conda/>

⁸⁰ <https://wiki.ligo.org/Computing/CondaPackaging>

```
conda activate  
conda config --add channels conda-forge
```

Next, install all available LIGO software:

```
wget -q https://git.ligo.org/lscsoft/conda/raw/master/environment-py36.yml  
conda env create -f environment-py36.yml
```

Finally, you can activate the LIGO python computing environment with:

```
conda activate ligo-py36
```

You will probably want to put this in your `.bashrc` if you plan on using it all the time.

17.6.2 Troubleshooting Installation

Problem: `cmake` keeps failing while installing IceCube software.

Solution: You probably did not check out the entire IceCube repository when you ran `svn co`. You will need to remove the IceCube source and build directories by running `sudo rm -rf /usr/local/icecube/offline/*` and start over from when you *entered your IceCube credentials*.

17.6.2.1 Creating a new user

You should preferably not run the pipeline as `root`. You can add a new user named, e.g., `vagrant` in Debian and Ubuntu with:

```
adduser vagrant
```

You will then want to give this user `sudo` privileges by adding them to the `sudoers` file (this will allow the user to do administrative tasks using the command `sudo`). You will want to run

```
visudo
```

which will open up the `sudoers` file for editing in the default text editor (for Debian, this is probably `nano`⁸¹). You will need to add the following line anywhere in the file, then save and quit:

```
vagrant ALL=(ALL) NOPASSWD: ALL
```

(Of course, if your username is not `vagrant`, you should use a different name above.)

You can test whether your update to `sudoers` succeeded by running:

```
sudo echo 'Success! You have sudo privileges.'
```

If this succeeds, it will print a success message.

⁸¹ <https://www.nano-editor.org/dist/v2.7/nano.html#Editor-Basics>

17.6.2.2 Out of Memory

Add swap space (virtual memory) using the following commands:

```
sudo fallocate -l 12G /swapfile
sudo chmod 0600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
sudo swapon -s
free -m
sudo bash -c \
    "echo '/swapfile    none    swap    sw    0' >>/etc/fstab"
cat /etc/fstab
```

17.6.2.3 MATLAB Installation Troubleshooting

You will probably want to install MATLAB using the GUI. This is easy if you are already on a desktop environment.

If you are configuring a remote server to run the LLAMA pipeline, you will need to make sure that you have X11 Forwarding enabled so that you can control the MATLAB installer GUI from your computer. See the appendix entry on [SSH with X11 Forwarding](#) for information.

Unzip the downloaded zip file with (note: the filename might change if you install a newer version of MATLAB.):

```
unzip matlab_R2016b_ginxa64.zip -d ~/matlab
```

and then run the `install` executable located in the unzipped directory, which will launch the MATLAB installer GUI:

```
~/matlab/install
```

Make sure to select items 1-3 (listed in the [install MATLAB](#) section) in the installer window in addition to MATLAB itself. You can do without all of the other toolboxes. Once you are done, run `matlab` with

```
matlab
```

which will bring up the MATLAB GUI. Download `jsonlab`, then change your MATLAB working directory to the folder containing `jsonlab` and double click on the `jsonlab` file, which should have a full name along the lines of `jsonlab-1.2.mltbx`. You will be asked if you would like to install `jsonlab`; you should, of course, say yes. Again, bear in mind that you might need to repeat this procedure; for some reason MATLAB doesn't always "remember" that it has `jsonlab` installed when you restart it. I don't know how to remedy this besides installing `jsonlab` for a second time. You can confirm that JSONLab is installed by opening a MATLAB session and seeing if `loadjson` is a valid command; if it is, then JSONLab has been successfully installed.

17.6.3 Install IceCube Offline Software with Root

At the original time of writing, it was not necessary to *install IceCube offline software*. The following instructions are necessary in the event that IceCube software libraries with ROOT⁸² dependencies become needed by LLAMA. For some reason, IceCube offline software will not compile properly when using the version of ROOT installed by the system package manager, so it is necessary to install the IceCube I3_PORTS version, as detailed below. This installation is **extremely time consuming** and should be skipped unless IceCube packages containing root become necessary for LLAMA.

⁸² <https://root.cern.ch/>

First, install dependencies⁸³ for the IceCube offline software (plus a few other requirements for LLAMA):

```
sudo apt-get -y install build-essential cmake libbz2-dev libgl1-mesa-dev \
    freeglut3-dev libxml2-dev subversion libboost-python-dev \
    libboost-system-dev libboost-signals-dev libboost-thread-dev \
    libboost-date-time-dev libboost-serialization-dev \
    libboost-fs-dev libboost-program-options-dev \
    libboost-regex-dev libboost-iostreams-dev libgs10-dev libcdk5-dev \
    libarchive-dev python-scipy ipython-qtconsole libqt4-dev \
    python-urwid python-tables libxft-dev
```

Next, install I3 Ports, a set of executables that provide a consistent environment for IceCube offline software. IceCube documentation⁸⁴ suggests running this as a user other than root. *This next step will take several hours to complete and must be restarted if it fails partway through. You might consider letting this run overnight or during some other unsupervised stretch of time. It will be significantly quicker on a fast system, but still painfully slow.*

```
cd ~
svn co http://code.icecube.wisc.edu/icetray-dist/tools/DarwinPorts/trunk \
    port_source
cd port_source
./i3-install.sh "$I3_PORTS"
```

Next, check out IceCube offline software. *if you encounter a segmentation fault during SVN checkout, see the next set of instructions below the next code block.* These instructions are adapted from the original IceCube offline software installation instructions⁸⁵; you can find all offline software releases here⁸⁶.

```
mkdir ~/offline
cd ~/offline
svn co http://code.icecube.wisc.edu/svn/meta-projects/offline-software/releases/V15-08-
↪ 00 src
```

If you encounter a segmentation fault (a known bug⁸⁷ on Debian Jesse's SVN version), you can run svn cleanup src followed by svn update src and repeat until the source code has been fully checked out.

Now it is time to build the IceTray software. *This step is also very long (if not as long as the previous one); set aside a couple of hours for it. Fortunately, this step can be resumed at the make step in the event of failure.*

```
mkdir ~/offline/build
cd ~/offline/build
"$I3_PORTS/bin/cmake" ../src
make
```

Finally, download some data used by IceCube test and example scripts.

```
make rsync
```

You should now be able to open up an icecube environment using:

```
./env-shell.sh
```

⁸³ http://software.icecube.wisc.edu/offline_trunk/projects/cmake/platforms.html#platforms

⁸⁴ http://software.icecube.wisc.edu/offline_trunk/projects/cmake/installing_ports.html

⁸⁵ http://software.icecube.wisc.edu/offline_trunk/metaproject/quickstart.html

⁸⁶ https://wiki.icecube.wisc.edu/index.php/IceTray_Releases#Release_Version:_V13-07-00

⁸⁷ <https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=736879>

You should see a “Welcome to Ice Tray” message. You can check whether the IceCube python module has been installed by running:

```
python -c 'from icecube import dataclasses; print("Success!")'
```

If this runs without error and prints Success!, then you have successfully installed the IceCube python module. Well done! You can now exit the Ice Tray environment:

```
exit
```

To make sure that IceCube software is available when logging in, you will have to add some new environmental variables to your .bashrc file with the install locations. Since environmental variables depend on where you installed the IceCube offline software, it is vital that you base the environmental variables off of your current installation configuration. **Making sure that you are still in the build directory**, run the following commands:

```
eval "$(grep ROOTSYS= env-shell.sh)"
_I3_BUILD="$(pwd -P)"
pushd ../src
_I3_SRC="$(pwd -P)"
popd
pushd "$I3_PORTS"
_I3_PORTS="$(pwd -P)"
popd
_I3_TESTDATA="$I3_PORTS/test-data"
rm -f ~/.i3env
printf 'ROOTSYS="%s" '$ROOTSYS' >>~/.i3env
printf 'PYTHONPATH="%s"/lib:"%s"/lib:$PYTHONPATH'\n' \
    "$_I3_BUILD" "$ROOTSYS" >>~/.i3env
printf 'I3_BUILD="%s" '$_I3_BUILD' >>~/.i3env
printf 'I3_SRC="%s" '$_I3_SRC' >>~/.i3env
printf 'I3_PORTS="%s" '$_I3_PORTS' >>~/.i3env
printf 'I3_TESTDATA="%s" '$_I3_TESTDATA' >>~/.i3env
```

You can now test whether your environment will run correctly by sourcing your .bashrc file:

```
. ~/.bashrc
```

You should see a message telling you that the IceCube software environment was successfully configured. Quickly test whether the IceCube python module is installed in the same way as before:

```
python -c 'from icecube import dataclasses; print("Success!")'
```

Once again, you should see Success! printed on your console. If so, congratulations! You have successfully installed the IceCube environment.

17.6.4 Installing Ubuntu for Windows

With Windows 10, it is now possible to install an Ubuntu userspace (with Bash included) on a Windows machine. This means that Ubuntu packages can be installed and Ubuntu binaries can be run directly on Windows (thanks to some sort of system call translation wizardry). This provides a handy and straightforward way of connecting to remote servers via SSH, even if you are on Windows, without having to install something like [Putty](#)⁸⁸. This has some other nice advantages, and, for the purpose of this guide, I will assume you have Bash installed on your local computer (even if you are using SSH to connect to a remote server for installing the LLAMA software).

You can follow the instructions in [this guide](#)⁸⁹ to install Ubuntu for Windows.

17.6.5 Logging in to a Remote Server Using SSH

Start by opening up a bash instance.

- If you are on a Mac, you can do this by running Terminal.app, which should be in your /Applications folder, or by hitting Command-Space, typing “Terminal.app”, and hitting Enter.
- If you are on a PC running Windows 10, you can actually *install the Ubuntu userspace* (with Bash included). Once this is installed, you can just open the Bash executable from your programs folder in the Start menu, or just hit the Windows button, type bash, and hit Enter to launch a new Bash session.
- If you are running Linux and don’t know how to start a terminal session, may God help you.

Now, let’s say you are logging in to a server at IP address 1.2.3.4, your username on the remote server is `vagrant`, and you have some password. Just run the following and enter your password when prompted:

```
ssh vagrant@1.2.3.4
```

If you have your server mapped to a web URL, like `example.com`, you can use that instead of the IP address, as shown below:

```
ssh vagrant@example.com
```

Congratulations! You should be logged in to your remote server. You can now proceed with installing the pipeline.

17.6.6 Getting LLAMA Software onto a Remote Server

Let’s assume someone sent you the LLAMA software in a compressed archive (e.g. a zipfile). You should have received a download link for the archive, in which case you can download the file directly from your LLAMA server. If you are running a remote machine, [log in using SSH](#); if you are running LLAMA locally or in a virtual machine, simply open up a terminal session.

Let’s assume the URL for the LLAMA software archive is `example.com/llama.zip`. You can download the archive to your home directory as follows:

```
cd ~  
wget example.com/llama.zip
```

Now, you can unzip the file:

```
unzip llama.zip
```

You should now have a directory called `multimessenger-pipeline` in your home folder.

⁸⁸ <http://www.putty.org>

⁸⁹ <http://www.howtogeek.com/249966/how-to-install-and-use-the-linux-bash-shell-on-windows-10/>

17.6.7 SSH with X11 Forwarding

This will allow you to run MATLAB's installer GUI and the MATLAB GUI itself, both of which are necessary for the installation process, on a remote server or on a local virtual machine being accessed through ssh.

- On Macs, you must install [XQuartz⁹⁰](#). Then, when logging in to the LLAMA server, use the `-o ForwardX11=yes` and `-o ForwardX11Trusted=yes` flags to turn on X Window Forwarding. For example, if your server is `c137.com` and the username is `rick.sanchez`, you can log in with:

```
ssh -o ForwardX11=yes -o ForwardX11Trusted=yes rick.sanchez@c137.com
```

- On Linux, assuming you have a GUI, you probably have an X11 Server installed already. If that is the case, you can just ssh with the `-o ForwardX11=yes` and `-o ForwardX11Trusted=yes` flags, just as in the above example for Macs.
- On Windows machines, you can use [XMing⁹¹](#) as your X Windows server and [Putty⁹²](#) as your SSH client. If you have Windows 10, you can use Ubuntu for Windows (See instructions for [Installing Ubuntu for Windows](#)), and you can use bash instead of Putty. Once again, you can run

```
ssh -o ForwardX11=yes -o ForwardX11Trusted=yes rick.sanchez@c137.com
```

If you use your `~/.ssh/config` file for ssh settings (will work for Linux or Mac; I am not sure about Windows), you can also automatically turn on X11 forwarding for your LLAMA server by adding the lines

```
ForwardX11 yes
ForwardX11Trusted yes
```

to the site entry for your LLAMA server in your `~/.ssh/config` file. For example, if you wanted to call the above server `earth`, you could add the following block to your `~/.ssh/config` file:

```
Host earth
  HostName c137.com
  User rick.sanchez
  ForwardX11 yes
  ForwardX11Trusted yes
```

Which makes connecting to the server with all of your desired settings much quicker (and saves you from having to remember ssh command syntax or the server URL):

```
ssh earth
```

Once you have established an SSH connection with X11 forwarding enabled, you will need to make sure that the root account and the account you are logging into share the same `.Xauthority` file. To do so, run:

```
cd ~
if sudo [ -e /root/.Xauthority ]; then
    sudo mv /root/.Xauthority /root/.Xauthority.orig
fi
sudo cp .Xauthority /root/.Xauthority
```

⁹⁰ <https://www.xquartz.org/>

⁹¹ <https://sourceforge.net/projects/xming/>

⁹² <http://www.putty.org/>

17.6.8 Using LLAMA

The `llama` python module and its executables all have docstring documentation and are designed for interactive use. Read the help functions and play around with things in `ipython` to get a feel for how they work, and ask Stefan Countryman for help if you have any questions. I will put FAQs here if I find that any Qs have become F.

17.6.9 Documenting LLAMA

17.6.9.1 Overview

LLAMA uses `reStructuredText`⁹³ documents to generate nicely-formatted webpages and PDF files (via `Sphinx`⁹⁴). There is even a `Makefile` included to facilitate publishing. The instructions below assume that you are starting from within the LLAMA installation folder (if you followed this installation guide, it will be in `~/multimessenger-pipeline`).

17.6.9.2 Publishing to gwhen.com Website

This repository contains tools for making a nice, reviewer/developer friendly website on the production server (`gwhen.com`⁹⁵ at time of writing). Check out the `README.md` file in `~/multimessenger-pipeline/review-site` for instructions. Also make sure that the contents of `multimessenger-pipeline/git-hooks` has been copied to `multimessenger-pipeline/.git/hooks` (to ensure that documentation is automatically refreshed whenever the documentation source is changed).

You can also enable the status server at `gwhen.com/status`⁹⁶. First, turn on CGI scripts:

```
sudo a2enmod cgi
```

You'll then need to add this alias to your apache configuration file (should be `/etc/apache2/apache2.conf`):

```
ScriptAlias "/status/" "/var/www/html/status/status.py"
```

Finally, restart apache:

```
sudo service apache2 restart
```

17.6.9.3 Publishing Readme to Git Host

If your Git hosting provider (e.g. `GitHub`⁹⁷) supports symbolic links and `reStructuredText`⁹⁸ rendering, you can just commit your changes and push them to the provider's remote repository. The Readme will immediately be rendered and updated.

⁹³ <https://github.com/ralsina/rst-cheatsheet/blob/master/rst-cheatsheet.rst>

⁹⁴ <http://www.sphinx-doc.org/en/1.4.8/>

⁹⁵ <http://gwhen.com>

⁹⁶ <http://gwhen.com/status>

⁹⁷ <https://github.com>

⁹⁸ <https://github.com/ralsina/rst-cheatsheet/blob/master/rst-cheatsheet.rst>

17.6.9.4 Publishing PDFs

From the LLAMA installation folder, navigate into the documentation folder and generate the PDF:

```
cd docs
make latexpdf
```

You can check the quality of your handiwork immediately using `xpdf` to view the output file:

```
xpdf build/latex/*pdf
```

17.6.9.5 Publishing HTML Web Pages

From the LLAMA installation folder, navigate into the documentation folder and generate the HTML files:

```
cd docs
make html
```

You can run a python server from within `docs` to view the output:

```
cd build/html
python -m SimpleHTTPServer
```

While the python server is running, you can view the documentation webpage by opening any web browser and navigating to the URL of your server (you will have to append the port number, which defaults to 8000 for `SimpleHTTPServer`). If your server has the URL `example.com`, then you would want to navigate to `example.com:8000`. You should then be able to see the webpage.

17.6.10 Troubleshooting LLAMA

Problem: `llama run` (*or some other program that updates files*) keeps crashing when trying to generate a skymap or other data generated with MATLAB. The MATLAB log located at `logs/llama.matlab.log` mentions something about `loadjson` or `savejson` (or something else with `json` in it), like in the below error message:

```
Undefined function 'loadjson' for input arguments of type 'char'.
```

Solution: Make sure the JSONLab toolbox is installed in MATLAB. For some reason installation does not persist after the first installation, and a second installation is usually required. You can confirm that JSONLab is installed by opening a MATLAB session and seeing if `loadjson` is a valid command; if it is, then JSONLab has been successfully installed.

Problem: `llama run` (*or some other program that updates files*) keeps crashing when trying to download `lvc_initial.fits.gz` or some other file located on GraceDB.

Solution: Refresh your LIGO authentication. See the section above on [LIGO authentication](#).

Problem: `llama run` (*or some other program that updates files*) keeps crashing when trying to generate a skymap or other data generated with MATLAB; MATLAB claims that files are missing.

Solution: Download the missing `~/multimessenger-pipeline/Neutrinos/Data` folder.

Problem: I am running this on a small throwaway server and am out of space, but I don't want to spend more money on disk space.

Solution: Delete the contents of `/var/cache/apt` and `/usr/share/doc` to clear up (probably) around 3GB of data that are unlikely to be of further benefit.

17.6.11 Setting Up the Review Server

These instructions explain how to set up a server for reviewers to test the pipeline on.

17.6.11.1 Provisioning the review server

Run the following command and enter your admin password (since we're creating a new user named "reviewer") as well as info for the new user:

```
~/multimessenger-pipeline/review-site/provision
```

This will get everything set up for a new reviewer. In particular, if the `reviewer` user account has not been made, a command will be run to create the account for you; in this case, you will have to specify a password, accept all other defaults, and then relaunch the `.provision` script. Now, assuming you have password-based authentication turned on, your reviewers can log in to the server using the username `reviewer` and the password you specified.

17.6.11.2 Running the Review

Run:

```
~/llamatatest
```

This script will run through a few injected test cases (as described on gwhen.com⁹⁹). It will describe what it has done and whether it succeeded; you can confirm the results for yourself by looking in the output directory, `~/.local/share/llama/current_run`. The python library used by `llamatatest` is located in `~/multimessenger-pipeline/llama`.

17.6.12 Ideas for the Future

This section contains exploratory notes and links on ideas, methods, and features that might be useful for the pipeline in the future.

17.6.13 CVMFS

It's worth investigating using the [CernVM-File System](#)¹⁰⁰ (CernVM-FS, or CVMFS) as a software environment provisioning method. Software and data are represented in a virtual file-system mounted to `/cvmfs`, and file therefrom are downloaded and cached locally on an as-needed basis, promising fast provisioning times.

LIGO seems to just be starting to support CVMFS¹⁰¹ at time of writing, while IceCube seems to have mature CVMFS support¹⁰² based on a cursory viewing of their documentation. If software matures in both collaborations to the point where the distributions on CVMFS can dependably provide all of the needed features, it might be possible to switch over to CVMFS for IceCube and LIGO software.

⁹⁹ <http://gwhen.com>

¹⁰⁰ <https://cvmfs.readthedocs.io/en/stable/>

¹⁰¹ <https://docs.ligo.org/lscsoft/conda/environments/#pre-built-environments>

¹⁰² <http://software.icecube.wisc.edu/documentation/info/cvmfs.html#cvmfs>

17.6.13.1 Advantages

- If other projects support CVMFS, it will be faster to incorporate their tooling into the pipeline using CVMFS.
- We can create very lightweight virtual machine images and docker containers with empty CVMFS caches for archiving, development, and deployment purposes; they will be faster to provision, archive, download, and restart thanks to this reduces size (though in a cluster environment, it would of course cause multiple parallel downloads to CVMFS, which might be a bottleneck, or even worse, an unintentional DDoS attack on CVMFS).

17.6.13.2 Disadvantages

- Less flexible than having a user-maintained Conda installation.
- root privileges would be necessary for sysadmin, and CVMFS idiosyncrasies will likely make it much harder to support heterogeneous platforms in a fault-tolerant way (this is pretty important).

CHAPTER
EIGHTEEN

ACADEMIC PAPERS

This page contains links to relevant academic papers describing LLAMA and other related efforts and science.

18.1 Low-Latency Algorithm for Multi-messenger Astrophysics (LLAMA) with Gravitational-Wave and High-Energy Neutrino Candidates

Stefan Countryman, Azadeh Keivani, Imre Bartos, Zsuzsa Marka, Thomas Kintscher, Rainer Corley, Erik Blaufuss, Chad Finley, Szabolcs Marka ([arXiv¹⁰³](#), [NASA ADS¹⁰⁴](#))

We describe in detail the online data analysis pipeline that was used in the multi-messenger search for common sources of gravitational waves (GWs) and high-energy neutrinos (HENs) during the second observing period (O2) of Advanced LIGO and Advanced Virgo. Beyond providing added scientific insight into source events, low-latency coincident HENs can offer better localization than GWs alone, allowing for faster electromagnetic follow-up. Transitioning GW+HEN analyses to low-latency, automated pipelines is therefore mission-critical for future multi-messenger efforts. The O2 Low-Latency Algorithm for Multi-messenger Astrophysics (LLAMA) also served as a proof-of-concept for future online GW+HEN searches and led to a codebase that can handle other messengers as well. During O2, the pipeline was used to take LIGO/Virgo GW candidates as triggers and search in realtime for temporally coincident HEN candidates provided by the IceCube Collaboration that fell within the 90% credible region of the reconstructed GW skymaps. The algorithm used NASA's Gamma-ray Coordinates Network to report coincident alerts to LIGO/Virgo's electromagnetic follow-up partners.

18.2 Bayesian Multi-Messenger Search Method for Common Sources of Gravitational Waves and High-Energy Neutrinos

Imre Bartos, Doga Veske, Azadeh Keivani, Zsuzsa Marka, Stefan Countryman, Erik Blaufuss, Chad Finley, Szabolcs Marka ([arXiv¹⁰⁵](#), [NASA ADS¹⁰⁶](#))

Multi-messenger astrophysics is undergoing a transition towards low-latency searches based on signals that could not individually be established as discoveries. The rapid identification of signals is important in order to initiate timely follow-up observations of transient emission that is only detectable for short time periods. Joint searches for gravitational waves and high-energy neutrinos represent a prime motivation for this strategy. Both gravitational waves and high-energy neutrinos are typically emitted over a short time frame of seconds to minutes during the formation or evolution of compact objects. In addition, detectors searching for both messengers observe the whole sky continuously,

¹⁰³ <https://arxiv.org/abs/1901.05486>

¹⁰⁴ <https://ui.adsabs.harvard.edu/#abs/arXiv:1901.05486>

¹⁰⁵ <https://arxiv.org/abs/1810.11467>

¹⁰⁶ <https://ui.adsabs.harvard.edu/#abs/arXiv:1810.11467>

making observational information on potential transient sources rapidly available to guide follow-up electromagnetic surveys. The direction of high-energy neutrinos can be reconstructed to sub-degree precision, making a joint detection much better localized than a typical gravitational wave signal. Here we present a search strategy for joint gravitational wave and high-energy neutrino events that allows the incorporation of astrophysical priors and detector characteristics following a Bayesian approach. We aim to determine whether a multi-messenger correlated signal is a real event, a chance coincidence of two background events or the chance coincidence of an astrophysical signal and a background event. We use an astrophysical prior that is model agnostic and takes into account mostly geometric factors. Our detector characterization in the search is mainly empirical, enabling detailed realistic accounting for the sensitivity of the detector that can depend on the source properties. By this means, we will calculate the false alarm rate for each multi-messenger event which is required for initiating electromagnetic follow-up campaigns.

LLAMA PACKAGE

The Gravitational Wave/High Energy Neutrino (LLAMA) analysis pipeline.

```
class llama.Event(eventid_or_event, rundir='/Users/s/local/share/llama/current_run', pipeline=None)
Bases: llama.event.EventTuple, llama.flags.FlagsMixin, llama.versioning.GitDirMixin
```

An event object, used to update and access data associated with a specific trigger (which receives its own directory or `eventdir`). FileHandler or another Event (or anything with ‘`eventid`’ and ‘`rundir`’ properties) as an input argument, in which case it will correspond to the same event as the object provided as an argument. One can also provide a module or dictionary containing FileHandlers, which will be used to create a `FileGraph` for the Event (i.e. it will specify which files should be made for this event). Defaults to the files module for now, though eventually this should be refactored out.

Parameters

- **`eventid_or_event`** (`str` or `EventTuple` or `llama.filehandler.FileHandlerTuple`) – This can be a string with the unique ID of this event (which can simply be a filename-friendly descriptive string for tests or manual analyses), in which case the next arguments, `rundir` and `pipeline`, will be used; *OR*, alternatively, it can be an `EventTuple` (e.g. another Event instance), `FileHandlerTuple` (e.g. any `FileHandler` instance), or any object with valid `eventid` and `rundir` attributes. In this case, those attributes from the provided object will be re-used, and the `rundir` argument will be ignored. This makes it easy to get a new Event instance describing the same underlying event but with a different Pipeline specified, or alternatively to get the Event corresponding to a given FileHandler (*though in this case you should take care to manually specify the ‘Pipeline’ you want to use!*).
- **`rundir` (`str`, optional)** – The `rundir`, i.e. the directory where all events for a given run are stored, if it differs from the default and is not specified by `eventid_or_event`.
- **`pipeline` (`llama.pipeline.Pipeline`, optional)** – The `pipeline`, i.e. the set of FileHandlers that we want to generate, if it differs from the default pipeline. If none is provided, use `DEFAULT_PIPELINE`.

Returns `event` – A new Event instance with the given properties.

Return type `Event`

Raises `ValueError` – If the `eventid_or_event` argument does not conform to the above expectations or if the `rundir` directory for the run does not exist, a `ValueError` with a descriptive message will be thrown.

property auxiliary_paths

Names of possible auxiliary paths in the directory that are used to track the state of the Event as a whole.

clone(`commit='HEAD'`, `rundir=None`, `clobber=False`)

Make a clone of this event in a temporary directory for quick manipulations on a specific version of a file.

Parameters

- **commit** (*str, optional*) – The commit hash to check out when cloning this event. If not specified, the most recent commit will be used. Unsaved changes will be discarded.
- **rundir** (*str, optional*) – The run directory in which to store the cloned event. If not specified, a temporary directory will be created and used. The contents of this directory will NOT be deleted automatically.
- **clobber** (*bool, optional*) – Whether this cloned event should overwrite existing state.

Returns

- **clone_event** (*llama.event.Event*) – A clone of this event. The full history is saved, but the specified **commit** is checked out. Any uncommitted changes in the working directory will not be copied over to the **clone_event**. If **clone_event** already seems to be a valid event with the correct **commit** hash, no further action will be taken (thus repeated cloning has little performance penalty).
- *Raises*
- *llama.versioning.GitRepoUninitialized* – If this is called on an Event that has not had its git history initialized.
- *IOError* – If this event already exists in the specified **rundir** and is checked out to a different hash, unless **clobber** is True, in which case that working directory will be deleted and replaced with the desired commit.

compare_contents(*other*)

Compare the file contents of this event to another event using `filecmp.cmpfiles` (though results are given as `FileHandler` instances rather than file paths). Use this to see whether two event directories contain the same contents under a given pipeline.

Parameters other (*Event, str*) – The other `Event` instance to compare this one to, or else a directory containing files that can be compared to this `Event` (though in that case the filenames must still follow the expected format).

Returns

- **match** (*FileGraph*) – A `FileGraph` for this `Event` whose files have the same contents as those corresponding to the *other* event.
- **mismatch** (*FileGraph*) – A `FileGraph` for this `Event` whose files have differing contents as those corresponding to the *other* event.
- **errors** (*FileGraph*) – A `FileGraph` for this `Event` whose corresponding files do not exist or otherwise could not be accessed for comparison (either for the files corresponding to this `Event` or the *other* one).

Raises ValueError – If the `Pipeline` instances of this `Event` and the *other* one are not equal, it does not make sense to compare them, and a `ValueError` will be raised.

creation_time()

The time at which this event directory was created (according to the underlying storage system). Note that you probably are more interested in `modification_time`.

property cruft_files

Return a list of files in the event directory that are not associated with any file handler nor with event state directories.

property eventdir

The full path to the directory containing files related to this event.

exists()

Check whether this event already exists.

property files

Get a FileGraph full of FileHandler instances for the files in this event with this particular pipeline.

classmethod fromdir(eventdir='.', **kwargs)

Initialize an event just by providing a filepath to its event directory. If no directory is specified, default to the current directory and try to treat that like an event. Note that the returned event will eliminate symbolic links when determining paths for rundir and eventid. Useful for quickly making events during interactive work.

Parameters

- **eventdir (str, optional)** – The event directory from which to initialize a new event.
- ****kwargs** – Remaining keyword arguments to pass to Event().

gpstime()

Return the GPS time of this event. Returns -1 if none can be parsed.

init()

Initialize the directory for this event, making sure it is in a proper state for processing data. Make sure the eventdir exists by creating it if necessary. Also initializes version control and set flags to the defaults specified in FlagsMixin.DEFAULT_FLAGS (which Event inherits).

Returns Returns this Event instance to allow command chaining.

Return type self

Raises ValueError – If the eventdir path exists but is not a directory or a link to a directory, we don't want to overwrite it to make an the directory.

modification_time()

The time at which this event directory was modified (according to the underlying storage system).

printstatus(loglevel=None)

Print a user-readable message indicating the current status of this event, or, optionally, log it at some log level. To simply print to STDOUT, call with no arguments. To use the logging system, specify the appropriate log level for the output using ‘debug’, ‘info’, ‘error’, ‘warning’, or ‘critical’. For example, if this is to be debug output, use:

```
>>> event = Event.fromdir()
>>> event.printstatus(loglevel='debug')
```

save_tarball(outfile)

Save this event and all its contents as a gzipped tarball. You should probably use a .tar.gz extension for the outfile name.

update(downselect)**

Generate any files that fit the FileGraph downselection criteria specified in downselect. By default, generate all files that have not been generated and regenerate all files that have been obsoleted because their data dependencies have changed. Returns True if files were updated, False if no files in need of update were found.

class llama.Pipeline(*args, **kwargs)

Bases: [llama.classes.ImmutableDict](#), [llama.classes.NamespaceMappable](#)

A pipeline specifies a specific set of data inputs and the functions that act on them in terms of intermediate data products and the functions used to generate them in a Directed Acyclic Graph (DAG); these products are bundled into FileHandlers. FileHandlers are graph nodes with DEPENDENCIES (edges) specified. A Pipeline DAG can

be built purely by specifying the specific FileHandlers which can be done trivially and clearly at the file-system level by putting the FileHandler code into a single directory for each pipeline.

Parameters

- **kwarg** (*dict*) – Names of FileHandler classes mapped to the classes themselves.
- **args** (*array-like*) – FileHandler classes. The `__name__` of each FileHandler will be used as the key.

Returns pipeline – A new Pipeline instance containing all of the FileHandler classes specified in args and kwarg.

Return type *Pipeline*

Raises TypeError – If there are any name collisions between classes in the input args and kwarg, if any of the FileHandler classes it contains are abstract (non-implemented) classes, or if any of the FileHandler classes it contains have missing `required_attributes`.

`check_consistency(other)`

Check whether two Pipeline instances use the same keys to describe the same FileHandler classes, raising a ValueError if they don't.

`dependency_graph(outfile: Optional[str] = None, title: str = 'Pipeline', url: Optional[function] = None, bgcolor: str = 'black')`

Return a graphviz .dot graph of DEPENDENCIES between file handlers in this pipeline. Optionally plot the graph to an output image file visualizing the graph.

Optional file extensions for outfile:

- *dot*: just save the dotfile in .dot format.
- *png*: save the image in PNG format.
- *pdf*: save the image in PDF format.
- *svg*: save the image in svg format.

Parameters

- **outfile** (*str, optional*) – If not provided, return a string in .dot file format specifying graph relationsIf an output file is specified, infer the filetype and write to that file.
- **title** (*str, optional*) – The title of the pipeline graph plot.
- **url** (*FunctionType, optional*) – A function taking FileHandler classes as input and returning a URL that will be added to each FileHandler class's node in the output graph. Allows you to add links. If not included, URLs will not be included.
- **bgcolor** (*str, optional*) – The background color to use for the generated plot.

Returns dot – The dependency graph in .dot format (can be used as input to dot at the command line). This is returned regardless of whether an outfile is specified.

Return type str

`downselect(invert=False, reducer=<built-in function all>, **kwargs)`

Return a Pipeline instance whose FileHandler classes match ALL the given query parameters.

Parameters

- **invert** (*bool, optional*) – Invert results. (Default: False)
- **reducer** (*function, optional*) – Specify any builtin to match if any check passes. Specify all to match only when every check passes. (Default: all)

- **type** (*type, optional*) – The type of the FileHandler must exactly match the given FileHandler.
- **typename** (*str, optional*) – The FileHandler type's name must match this string.
- **subclass** (*type, optional*) – The FileHandler must be a subclass of this FileHandler.
- **subgraph** (*type, optional*) – The FileHandler must be either this FileHandler or one of its UR_DEPENDENCIES; use this to make a Pipeline that only generates the subgraph leading to this FileHandler.

file_handler_instances(*args, **kwargs)

Return a FileHandlerMap with FileHandler instances sharing the same initialization arguments, e.g. for FileHandler instances that all refer to the same event.

classmethod from_module(module)

Create a pipeline by extracting all FileHandler objects from a given submodule.

```
class llama.Run(rundir='/Users/s/local/share/llama/current_run', pipeline=frozenset('Advok',
    'CoincScatterI3LvcPdf', 'CoincScatterI3LvcPng', 'CoincScatterZtfI3LvcPdf',
    'CoincScatterZtfI3LvcPng', 'CoincScatterZtfLVCPdf', 'CoincScatterZtfLVCPng',
    'CoincSignificanceI3Lvc', 'CoincSignificanceSubthresholdI3Lvc', 'CoincSummaryI3LvcPdf',
    'CoincSummaryI3LvcTex', 'FermiGRBsJSON', 'IceCubeNeutrinoList',
    'IceCubeNeutrinoListCoincTxt', 'IceCubeNeutrinoListTex', 'IceCubeNeutrinoListTxt',
    'LVAlertAdvok', 'LVAlertJSON', 'LVCGraceDbEventData', 'LvcDistancesJson', 'LvcGcnXml',
    'LvcRetractionXml', 'LvcSkymapFits', 'LvcSkymapHdf5', 'PAstro',
    'RctSlkI3CoincSummaryI3LvcPdf', 'RctSlkLmaCoincScatterI3LvcPdf',
    'RctSlkLmaCoincScatterI3LvcPng', 'RctSlkLmaCoincScatterZtfI3LvcPdf',
    'RctSlkLmaCoincScatterZtfI3LvcPng', 'RctSlkLmaCoincScatterZtfLVCPdf',
    'RctSlkLmaCoincScatterZtfLVCPng', 'RctSlkLmaCoincSignificanceI3Lvc',
    'RctSlkLmaCoincSignificanceSubthresholdI3Lvc', 'RctSlkLmaCoincSummaryI3LvcPdf',
    'RctSlkLmaLVAlertJSON', 'RctSlkLmaLVCGraceDbEventData', 'RctSlkLmaLvcDistancesJson',
    'RctSlkLmaLvcGcnXml', 'RctSlkLmaLvcRetractionXml', 'RctSlkLmaSkymapInfo', 'SkymapInfo',
    'ZtfTriggerList')), downselection=())
```

Bases: `llama.run.RunTuple`

A single directory containing multiple event directories combined with a pipeline (i.e. a selection of analysis steps to use) and a set of downselection criteria for picking events:

Run Directory |— Event Directory 1 |— Event Directory 2 |— Event Directory 3

This should ordinarily correspond to a run of some sort (an observing run, engineering run, offline run, test run, etc.) where the events are somehow related. Since this class mostly just provides methods for organizing and selecting Event instances with tailored Pipeline instances, it's up to you to decide how to best organize a run. Run objects are immutable to simplify hashing and uniqueness checks.

These tools allow the user to conveniently check on the status of all events in a given Run. A dictionary of downselection arguments (as fed to `downselect`) can be used to restrict the set of events that will be returned events.

Parameters

- **rundir** (*str*) – The directory where all events are stored. Files for individual events are stored in per-event subdirectories of `rundir`. Will be converted to a canonical path with `os.path.realpath` to help ensure unique Run definitions.
- **pipeline** (`llama.pipeline.Pipeline, optional`) – A Pipeline instance holding FileHandler classes that should be used for this analysis. Defaults to the main pipeline in production use.

- **downselection**(*tuple, optional*) – A tuple of dictionaries of keyword arguments of the type passed to downselect. The events returned by `events` will match these downselection criteria with each downselection dict applied in the order they appear in this argument (to allow more complex chained downselections). You probably don't want to manually specify this; a more pythonic way to provide downselection arguments is to use the `downselect` method to return a downselection from a starting Run.

`downselect(**kwargs)`

Get another Run instance identical to the current one but with the following downselection criteria applied to the Event instances returned by `self.events`. Can also specify a sorting function and a maximum number of returned values:

Parameters

- **invert**(*bool, optional*) – Invert what matches and what doesn't. Default: False
- **eventid_filter**(*str, optional*) – A glob (as taken by `fnmatch`) that the eventid must match.
- **fileexists**(*str, optional*) – The event directory contains a file with this name.
- **fexists**(*llama.filehandler.FileHandler, optional*) – The eventdir contains the file for this FileHandler.
- **fnameexists**(*str, optional*) – The eventdir contains the file for the FileHandler with this name.
- **fhmeta**(*llama.filehandler.FileHandler, optional*) – The eventdir contains a metadata rider for the file for this FileHandler.
- **fnamemeta**(*str, optional*) – The eventdir contains a metadata rider for the file for the FileHandler with this name.
- **vetoed**(*bool, optional*) – Whether the events have been vetoed by the VETOED flag or not.
- **manual**(*bool, optional*) – Whether the events have been marked as manual by the MANUAL flag.
- **modbefore**(*float, optional*) – Select events whose directory modtimes were before this timestamp.
- **modafter**(*float, optional*) – Select events whose directory modtimes were after this timestamp.
- **sec_since_mod_gt**(*float, optional*) – Select events whose directory modtimes are more than this many seconds ago.
- **sec_since_mod_lt**(*float, optional*) – Select events whose directory modtimes are less than this many seconds ago.
- **v0before**(*float, optional*) – Select events whose first event state version was generated before this timestamp. Will **IGNORE** directories that do not have any versioned files.
- **v0after**(*float, optional*) – Select events whose first event state version was generated after this timestamp. Will **IGNORE** directories that do not have any versioned files.
- **sec_since_v0_gt**(*float, optional*) – Select events whose first event state version was generated more than this many seconds ago. Will **IGNORE** directories that do not have any versioned files.

- **sec_since_v0_lt** (*float, optional*) – Select events whose first event state version was generated less than this many seconds ago. Will **IGNORE** directories that do not have any versioned files.
- **sortkey** (*function, optional*) – A function taking Event instances that can be passed to sorted to sort the downselected Event instances. Default: None (i.e. no sorting)
- **reverse** (*bool, optional*) – Whether to reverse the order of sorting (i.e. put the results in descending order) before applying limit. Default: False
- **limit** (*int, optional*) – Return up to this number of events. Most useful if sortkey has also been provided. Default: None (i.e. no limit)

downselect_pipeline(*invert=False, **kwargs*)

Return a Run instance with a pipeline that has been downselected using Pipeline.downselect.

property events

Return a list of events in this run directory with self.downselection criteria applied (see downselect for a list of possible downselection criteria).

Parameters

- **sortkey** (*function, optional*) – A sorting key (as passed to sorted) to use to sort the returned events. If none is provided, the events will be sorted based on astrophysical event time using Event.gpstime; beware that an error will be raised if this quantity is ill-defined for ANY of the returned events.
- **reverse** (*bool, optional*) – Whether to reverse the default sort order, i.e. put in descending order. True by default so that the most-recently-occurring events are first in the list.

update()

Get a list of Event instances matching this Run instance's downselection criteria and update each event directory.

property vis

A collection of visualization methods for this Run instance.

LLAMA.BATCH PACKAGE

Tools for analyzing large-scale batches of LLAMA data, e.g. to run simulations off of injected or randomized data, with inputs and results stored locally or on a cloud storage service.

`class llama.batch.DictAction(option_strings, dest, nargs=None, const=None, default=None, type=None, choices=None, required=False, help=None, metavar=None)`

Bases: `argparse.Action`

Split a list of command line arguments into an `OrderedDict` along the first occurrence of “=” in each one.

`llama.batch.batch_error_alert_maintainers(errdump, event, params)`

Get an error callback function like `llama.cli.traceback_alert_maintainers` to pass to `llama.cli.log_exceptions_and_recover` that provides batch-specific information about the error.

Parameters

- `errdump (array-like)` – A list of locations in which dumped data will go.
- `event (llama.event.Event)` – The event which errored. Will be noted in the Slack alert message.
- `params (dict)` – The dictionary that was used to format `errdump` containing values from `--params`. Contains information on the current event being processed.

`llama.batch.batch_error_dump(errdump, event)`

Get an error callback function like `llama.cli.traceback_alert_maintainers` to pass to `llama.cli.log_exceptions_and_recover` that dumps the results of the failing event to the directories listed in `errdump`.

Parameters

- `errdump (array-like)` – A list of locations in which dumped data will go. These can be S3 “directory” paths compatible with `put_file`.
- `event (llama.event.Event)` – The event which errored. Files will be dumped from here.

`llama.batch.load_params(path: str)`

Use `read_file` to read a local or remote path. Decode the file contents as either a JSON list or newline-delimited text file, and return the decoded list.

`llama.batch.postprocess_param_iterator(self: llama.cli.CliParser, namespace: argparse.Namespace)`

Overrides the `namespace.params` value with an iterator that captures all desired combinations of the specified `params`. Loads from file, HTTP, HTTPS, or S3 for each param list. If the value for `namespace.random` is not `None`, will instead become an infinitely long generator serving random combinations of the parameters.

Parameters

- `self (CliParser)` – The `CliParser` applying this post-processing.
- `namespace (Namespace)` – The processed arguments parsed by `self`.

Returns combinations – A generator returning a namedtuple whose names are the keys of parameters and whose values are drawn from the corresponding value of parameters.

Return type GeneratorType

`llama.batch.put_file(source: str, dest: str, public: bool = False)`

Save a file somewhere. Same as copying the file, but with S3 compatibility. The destination directory and all parent directories will be created if they don't exist.

Parameters

- **source (str)** – The path to the file to be uploaded.
- **dest (str)** – Destination path. Works for local paths or for S3 keys (by prepending `s3:/` /`{bucket}`/ where by `{bucket}` is the name of the S3 bucket to which the file should be uploaded). Unlike `read_file`, does not work for uploading via HTTP/HTTPS. Intermediate directories will be made for local paths.
- **public (bool, optional)** – Whether files uploaded to S3 should be publicly available. This will have no effect on local file destinations.

`llama.batch.read_file(path: str)`

Load data from a file path, either local, via HTTP/HTTPS, or from an S3 bucket.

Parameters path (str) – Path to a JSON or (newline-delimited) text file containing a list of values to load. Path can also be an S3 object, denoted with “`s3://`” as the prefix, or a URL, denoted with “`http://`” or “`https://`” as the prefix. In these cases, the file is not saved locally.

Returns data – The contents of the file.

Return type bytes

CHAPTER
TWENTYONE

LLAMA.CLASSES MODULE

Primitive base classes used throughout the pipeline.

```
class llama.classes.Colors(BLUE, RED, GREEN, MAGENTA, YELLOW, CLEAR, BOLD, UNDERLINE,  
                           blue, red, green, magenta, yellow, bold, underline)
```

Bases: tuple

BLUE

Alias for field number 0

BOLD

Alias for field number 6

CLEAR

Alias for field number 5

GREEN

Alias for field number 2

MAGENTA

Alias for field number 3

RED

Alias for field number 1

UNDERLINE

Alias for field number 7

YELLOW

Alias for field number 4

blue

Alias for field number 8

bold

Alias for field number 13

green

Alias for field number 10

magenta

Alias for field number 11

red

Alias for field number 9

underline

Alias for field number 14

```
yellow
    Alias for field number 12

class llama.classes.FileHandlerTuple(eventid, rundir, clsname)
Bases: tuple

clsname
    Alias for field number 2

eventid
    Alias for field number 0

rundir
    Alias for field number 1

class llama.classes.Hdf5Storage(filename)
Bases: object

A class for reading and writing to an HDF5 cache. Has a dictionary interface and abstracts away file opening and closing.

class llama.classes.ImmutableDict(mappable)
Bases: frozenset

A hashable, immutable namespace inspired by namedtuple. Initialize by passing a dict or an iterable of (key, value) tuples. Attributes are accessible using dot notation or the map interface.

get(k[, d]) → D[k] if k in D, else d. d defaults to None.

items()
D.iteritems() -> an iterator over the (key, value) items of D

keys()
D.iterkeys() -> an iterator over the keys of D

values()
D.itervalues() -> an iterator over the values of D

class llama.classes.JsonRiderMixin
Bases: object

Class for reading and writing JSON to multiple rider files in a manifest.

read_json(err=False)
Returns None if none of the fullpaths exists, unless err is True, in which case a FileNotFoundError will be raised.

write_json(outdict)
Write outdict to all the files in self.fullpaths.

class llama.classes.ManifestTuple(eventdir, manifest_filehandlers)
Bases: tuple

eventdir
    Alias for field number 0

manifest_filehandlers
    Alias for field number 1

llama.classes.MetaClassFactory(function, meth_names=None)
Create a MetaClass that wraps all methods in function. Use it by setting __metaclass__ = <your new class when declaring a class whose methods should be wrapped thus.
```

Parameters

- **function** (*func*) – The function that should wrap each method of a class.
- **meth_names** (*list, optional*) – If provided, only wrap these methods. Otherwise, wrap all methods.

Raises `TypeError` – If `meth_names` is provided, all of the specified strings must refer to functions in the class to be created, otherwise a `TypeError` will be raised.

`class llama.classes.NamespaceMappable`

Bases: `abc.ABC`

A mappable class implementing a dot-notation for accessing members.

`exception llama.classes.OptionalFeatureWarning`

Bases: `UserWarning`

A warning indicating that an optional feature will not be available. You will usually want to suppress this (except when running tests, evaluating the status of a fresh install/upgrade, or debugging).

`class llama.classes.RequiredAttributeMixin`

Bases: `object`

Class with a `required_attributes` classmethod that finds all `_REQUIRED` attributes for itself and all superclasses. Use this feature to check whether subclasses have implemented all required attributes from their abstract superclasses.

To add new required attributes, specify a `_REQUIRED` tuple listing the names of those attributes as `str` instances. If your superclasses already define other required attributes thus, there is no need to reinclude them in the new class's `_REQUIRED` tuple.

`classmethod required_attributes()`

Recursively fetch a set of required attributes for this class based on its `_REQUIRED` attribute as well as the `required_attributes` class method of all of superclasses implementing it.

`class llama.classes.RiderFile(eventdir, manifest_filehandlers)`

Bases: `llama.classes.ManifestTuple, abc.ABC`

A rider class that specifies `filenames` and `fullpaths` for a given manifest and a given `rider_fmt`.

`delete()`

Delete all rider files associated with this manifest.

`exists()`

Check whether any of the rider files in this manifest exist.

`property filenames`

Get the name of the file indicating the cooldown status of this file.

`property fullpaths`

Get the full path of the file indicating the cooldown status of this file.

`abstract property rider_fmt`

The format string for this rider file. Must be implemented by subclasses as a property or attribute.

`llama.classes.optional_env_var(varnames: List[str], errmsg: str = "", defaults: Optional[List[str]] = None)`

→ `List[str]`

Get environmental variables `varnames` from `os.environ`. Log and warn the user with an `OptionalFeatureWarning` if the environmental variable is not set and return `None`; otherwise, return the value of the environmental variable.

Parameters

- **varnames** (`List[str]`) – A list of environmental variables to import.

- **errmsg** (*str, optional*) – A descriptive message to display in logs and warnings if the user has not configured the specified environmental variable. This will be printed in addition to a default message explaining which environmental variables were not available and which module tried to define them.
- **defaults** (*List[str]*) – A list of default values corresponding to varnames. If any of the variables are not defined, raise the warning and use defaults. If defaults are not defined, return None for each variable.

Returns **values** – The values of the environmental variables specified in varnames, with each replaced by None if it is not set.

Return type List[str]

Raises **ValueError** – If defaults is not None but has a different length than varnames.

`llama.classes.placeholderclass(name, modulename, bases=<class 'object'>)`

Define and return a placeholder class that raises a `NotImplementedError` on instantiation.

Parameters

- **name** (*str*) – The name of the new class.
- **modulename** (*str*) – The name of the module you would like to import from.
- **bases** (*tuple*) – The base classes of the new placeholder class.

Returns **newclass** – A placeholder class that immediately raises a `NotImplementedError` when you try to instantiate it.

Return type type

`llama.classes.registerstub(name, modulename, stub)`

Register a stub object. Just a way to keep track of when partial implementations of objects have been defined as replacements for modules that cannot be imported for whatever reason (since these stub objects might not provide perfect/full functionality of the missing modules).

Parameters

- **name** (*str*) – The variable name of the stub object.
- **modulename** (*str*) – The name of the module where it is defined; you should probably set this to `__name__` to get the name of the defining scope for the stub.
- **stub** (*function or type*) – The stub object itself.

`llama.classes.rider_mixin_factory(classname, **kwargs)`

Get a Mixin with name `classname` for `FileHandler` classes that provides properties with the whose names are the `kwargs` keys and whose return values are `RiderFile` instances initialized to that specific `FileHandler`. This implies that the values of `kwargs` are `RiderFile` subclasses.

CHAPTER
TWENTYTWO

LLAMA.CLI MODULE

Command-Line Interface (CLI) primitives to be used by scripts throughout the pipeline. Base your scripts off of these classes as much as possible to shorten development time and provide a unified “look-and-feel” to the entire library’s CLI.

```
class llama.cli.CanonicalPathAction(option_strings, dest, nargs=None, const=None, default=None,
                                     type=None, choices=None, required=False, help=None,
                                     metavar=None)
```

Bases: `argparse.Action`

Canonicalize a collection of paths with `os.path.realpath`, remove duplicates, and sort the canonicalized paths so that the full list is an unambiguous representation of the specified values.

```
class llama.cli.CliParser(*args, parents=(), **kwargs)
```

Bases: `argparse.ArgumentParser`

Extend `ArgumentParser` with postprocessing functions that run on the parsed arguments when `parse_args` or `parse_known_args` are called to adjust their values or raise errors as necessary.

`POSTPROCESSORS = ()`

`error(message)`

Same as `ArgumentParser.error` but with a bright red error message.

```
parse_known_args(args: Optional[List[str]] = None, namespace=None)
```

Parse known arguments and apply all functions in `POSTPROCESSORS` to the returned `namespace`. Also return unrecognized arguments.

Parameters

- `args (List[str], optional)` – The arguments to parse. Will parse from `sys.argv` using `ArgumentParser.parse_args` if not provided.
- `namespace (optional)` – Namespace to pass to `ArgumentParser.parse_args`.

Returns

- `parsed (argparse.Namespace)` – Arguments with `self.postprocess` applied.
- `unrecognized (List[str])` – Unrecognized arguments.

```
postprocess(namespace: argparse.Namespace)
```

A method that acts on the `argparse.Namespace` returned by `ArgumentParser.parse_args` and returns the same `namespace` with any necessary modifications. A good place to raise exceptions or execute actions based on the full combination of parsed arguments. Works by calling `self.POSTPROCESSORS` in order (a tuple of functions with the same signature as the unbound `self.postprocess` method).

`Parameters namespace (argparse.Namespace)` – The return value of `ArgumentParser.parse_args`.

Returns `namespace` – The input with any necessary transformations applied.

Return type `argparse.Namespace`

```
print_help(file=None)
```

Print the help string for command line consumption. Same as `ArgumentParser.print_help`, but cleans up ReST directives in default output of the parser for improved legibility.

```
class llama.cli.Parsers
```

Bases: `object`

Use a `Parsers` instance to access `ArgumentParser` classes that can be passed as a list in any combination to a new `ArgumentParser` instance as the `parents` keyword argument. This prevents you from having to write the same help documentation repeatedly. You can override any keyword arguments

```
clobber = CliParser(prog='sphinx-build', usage=None, description=None,
formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error',
add_help=False)

dev_mode = CliParser(prog='sphinx-build', usage=None, description=None,
formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error',
add_help=False)

outdir = CliParser(prog='sphinx-build', usage=None, description=None,
formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error',
add_help=False)

outfile = CliParser(prog='sphinx-build', usage=None, description=None,
formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error',
add_help=False)

version = CliParser(prog='sphinx-build', usage=None, description=None,
formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error',
add_help=False)
```

```
class llama.cli.RecursiveCli(prog: Optional[str] = None, description: Optional[str] = None,
                                subcommands: Optional[Dict[str, module]] = None, localparser:
                                Optional[argparse.ArgumentParser] = None, preprocessor:
                                Optional[function] = None)
```

Bases: `object`

A recursive command line interface that allows the user to access the `__main__.py:main()` functions of submodules using a `CMD SUBCMD` notation with clever recursive helpstring documentation to enable straightforward subcommand discovery by the user and to avoid cluttering a namespace with hyphen-separated subcommands.

Examples

Using `RecursiveCli` to implement `main` in `llama.__main__` lets you access `llama.files.__main__:main()` in a convenient way from the command line. The commands below are equivalent:

```
python -m llama.files
python -m llama files
```

This becomes useful when you realize that you now only need a single script/alias/entry point for your script's submodules. So if you add an entry point or launcher script for `llama` to your distribution, you can replace `python -m llama` with `llama`, and you get the `llama files` subcommand without any extra work. With the addition of a single entry point, the above command becomes:

```
llama files
```

Better still, this can be applied recursively to every subpackage to access its mixture of package-level CLI commands and submodule CLIs (with no changes to your single point mentioned above). So, for example, the following two can be equivalent:

```
python -m llama.files.i3
llama files i3
```

Most importantly, this automatic feature discovery enables users to hierarchically find commands and features without necessitating changes to higher-level packages docstrings or CLIs. The same code that enables the subcommand syntax shown above allows those subcommands to be listed with a help string, so that the following command will *tell you* that `llama files` is a valid command and *summarize* what it does:

```
llama --help
```

You can then, of course, run `llama files --help` to learn details about that module (and see which submodules it offers). This is similar to `git` and other programs, but it can be recursed ad-infinitum for rich libraries.

description: str

classmethod from_module(modulename: str, **kwargs)

Autogenerate the submodules dictionary by finding available CLIs in `module`.

Parameters

- **modulename** (str) – fully-qualified module name in which to search for `subcommands` to pass to `__init__`. The keys of `subcommands` will simply be the module names of the submodules of `modulename`. `prog` will be the `modulename` with periods replaced by spaces, e.g. '`llama.files`' will turn into `prog='llama files'`, since this reflects the way the command is used (assuming the top level module, `llama` in the given example, implements a `RecursiveCli` and is callable from the command line using the top-level module name, again, `llama` in this example).
- ****kwargs** – Remaining arguments (besides `subcommands` and `prog`) will be passed to `__init__` along with the subcommands described above.

Returns `cli` – A new instance with `subcommands` automatically discovered from `module`.

Return type `RecursiveCli`

Raises `TypeError` – If `subcommands` or `prog` is in `**kwargs` or if any other arguments not recognized by `__init__` are passed.

get_epilog()

Get an epilog to pass to `argparse.ArgumentParser` that contains information on available subcommands.

get_parser()

Get a command-line argument parser for this CLI.

Returns `parser` – An `CliParser` instance (see `CliParser`, a subclass of `argparse.ArgumentParser` implementing post parsing hooks) containing all specified `subcommands`. If `self.localparser` was passed at initialization time, then `parser` will be initialized therefrom.

Return type `CliParser`

localparser: `argparse.ArgumentParser`

main()

The `main` function that you should run if this module is called as a script. Parses the command line options using `self.get_parser()`, prints the help documentation if no `SUBCOMMAND` is specified at the command line, runs `self.preprocessor(self.get_parser().parse_args())` and then, if it completes without exiting, runs a `SUBCOMMAND` if specified.

preprocessor: function**print_help_if_no_cli_args(_args)**

If no command-line arguments are parsed, prints the help documentation and exits. This is the default preprocessor (since, in the absence of another preprocessor, a complete absence of CLI arguments results in a no-op likely indicating a lack of user understanding).

Uses an `CliParser` to parse arguments (since it knows how to deal with variations in executable names, e.g. `python -m llama` vs. `llama/__main__.py` vs. `python3 llama/__main__.py`) and, if no arguments whatsoever are discernable, runs `self.get_parser().print_help()` and quits.

prog: str**subcommands: Dict[str, module]****llama.cli.close_stdout_stderr(outfile='/dev/null')**

Redirect stdout and stderr to `outfile` at the file descriptor level so that, when the process is forked to the background, no new data is written to stdout/stderr. Since everything should be getting logged anyway, this should not be necessary, but unfortunately we need to use production code that prints to these file descriptors instead of using logging.

Parameters `outfile(str, optional)` – The path to the logfile that should collect all of the output printed to STDOUT and STDERR. Defaults to `os.devnull` (i.e. delete all outputs).

llama.cli.get_logging_cli(default_logfile, default_loglevel='error')

Create a `CliParser` that automatically turns on logging to the specified output file (`--logfile`, always at maximum verbosity) as well as the terminal at the specified verbosity level (`--verbosity`) with `default_logfile` and `default_loglevel` as defaults. `verbosity` should be `none` if no terminal output is to be printed or else one of the logging log levels in lower case form (see: `LOGLEVELS`). Again, output will be logged to the `logfile` at the maximum verbosity (`DEBUG`) to make sure nothing is lost; suppress this behavior by setting `/dev/null` as the logfile.

Parameters

- `default_logfile(str)` – Path to which the script should log by default.
- `default_loglevel(int or NoneType, optional)` – How verbose to be by default. `none` means to print nothing; other values are typical log levels. Must be a value specified in `LOGLEVELS`.

Returns `parser` – A parser to use as one of the parents to a new `CliParser` instance, which will inherit the logging CLI options and automatic logging setup behavior.

Return type `CliParser`

Raises `ValueError` – If `default_loglevel` is not a value in `LOGLEVELS`.

llama.cli.get_postprocess_required_arg(*arg_names)

Return a postprocessor that prints help if the required argument `arg_names` are not specified at the command line by checking whether they're set to a value that evaluates as `False`. Use this if you want to defer checking for a required argument until postprocessing.

llama.cli.log_exceptions_and_recover(callbacks=(<function traceback_alert_maintainers>,))

Decorator to run `func` with no arguments. Log stack trace and run callbacks to clean up (default: send the traceback to maintainers) when any `Exception` is raised, returning the value of the wrapped function or else the exception that was caught (note that this will break functionality of any function that is supposed to return an

`Exception`, and that you should only apply this sort of behavior in a command line script that needs to recover from all exceptions). Error tracebacks are syntax-highlighted (for 256-color terminals) for readability.

Optionally provide an iterable of callbacks to run to override the default, `traceback_alert_maintainers`. Callbacks passed in this list must have the same signature as that function. Use this to perform other cleanup tasks or to avoid sending an alert on error.

Signature below is for the returned decorator.

Parameters `func` (*function*) – A function that is supposed to recover from **all** possible exceptions.

Exceptions with tracebacks will be logged and sent to maintainers using `alert_maintainers`.

You should only wrap functions that **cannot** be allowed to crash, e.g. the main function of a long-running CLI script.

Returns `func` – The wrapped function. Has the same inputs and return values as the original function

unless the original function raises an `Exception` while executing. In this case, the return value will be the exception instance that was raised. Note that you probably don't care about this value in normal use patterns, and also note that you should therefore *not* wrap a function that would ever nominally return an exception instance since there will no longer be any way to distinguish an expected return value from a caught exception.

Return type `function`

`llama.cli.loglevel_from_cli_count(loglevel)`

Get the label, e.g. `DEBUG`, corresponding to the number of times the user typed `-v` at the command line.

`llama.cli.parse_atom(*args, postprocessors=(), **kwargs)`

Create a new `CliParser` class with no help documentation added and add a single argument to it.

Parameters

- `*args` – Positional arguments to pass to the new parser's `add_argument` method.
- `postprocessors` (*list-like*) – A list of functions to set `POSTPROCESSORS` to in the returned `CliParser`
- `**kwargs` – Keyword arguments to pass to the new parser's `add_argument` method.

Returns `parser` – A new parser with a single argument. Pass this to other new `ArgumentParser` instances as one of the `parents`.

Return type `ArgumentParser`

`llama.cli.pidfile(lockdir)`

Get the path to the file containing the process ID for the `llama run` instance running on this run directory with this `eventidfilter`.

`llama.cli.postprocess_dev_mode(self: llama.cli.CliParser, namespace: argparse.Namespace)`

If we're not running on a clean git repository, quit (unless `namespace.dev_mode` is `True`, indicating that the developer knows the repository is in an unclean state). Intended to help reproducibility and correctness.

`llama.cli.postprocess_logging(self: llama.cli.CliParser, namespace: argparse.Namespace)`

Run `setup_logger(namespace.logfile, loglevel)` to set up a logger for this script based on user input.

`llama.cli.postprocess_version(self: llama.cli.CliParser, namespace: argparse.Namespace)`

If `namespace.version` is `True`, print the LLAMA version and exit.

`llama.cli.print_running_procs_action(running_lock_dir: str, command_nicknames: Dict[tuple, str])`

Get a `PrintRunningProcsAction` class that can be used in an `ArgumentParser` argument as the action to show which processes whose lockfiles are stored in `running_lock_dir` are currently running.

`llama.cli.printprocs(pids, command_nicknames=frozenset({}))`

Print a nicely-formatted list of processes and their subprocesses using `proc_printer`.

Parameters

- **pids** (*list-like*,) – An iterable of process ids.
- **command_nicknames** (*Dict[tuple, str]*, *optional*) – Specify nicknames for commands as a dictionary mapping from the arguments associated with a command (e.g. ['llama', 'run']) to the replacement nicknames for each (each of which will be wrapped in square brackets like [llama run]) when the processes are printed. The python version printed before these arguments is omitted to save space. This command shortening is intended to highlight and shorten interesting commands.

`llama.cli.proc_formatter(pid)`

Get a dictionary that can be used to format PROC_FMT via proc_printer with information about the process with id pid.

`llama.cli.proc_printer(procs, command_nicknames: Dict[tuple, str] = frozenset({}), indent: list = [])`

Print a bunch of processes in a nice tree-like way to STDOUT.

`llama.cli.register_exit_handler()`

Make sure we perform exit actions by calling `exit` explicitly on interrupts and SIGTERMs.

`llama.cli.running_pids(running_lock_dir: str)`

Find all running instances of whatever program is using `running_lock_dir` to store its lock directories (each of which should contain a pidfile with the process ID stored in it).

`llama.cli.safe_launch_daemon(lockdir: str, post_fork: Optional[function] = None)`

Fork this process twice, exiting the parent and grandparent, to put this script into the background and creating a lock directory (atomic on most filesystems) specific to this process, then run a `post_fork` function to do any extra initialization or conflict checking (e.g. to check whether this process is conflicting with other processes in a way not accounted for by the initial lock acquisition check). Continues execution in the new grandchild process.

`llama.cli.spawn_daemon()`

Do the UNIX double-fork magic, see Stevens' "Advanced Programming in the UNIX Environment" for details (ISBN 0201563177). Taken from <https://stackoverflow.com/questions/6011235>.

`llama.cli.traceback_alert_maintainers(func: function, err: Exception, tb: str, self, *args, **kwargs)`

An `error_callback` function for `log_exceptions_and_recover`. Runs `alert_maintainers` with the current traceback.

Parameters

- **func** (*FunctionType*) – The function that caused the error.
- **err** (*Exception*) – The exception that was caught.
- **tb** (*str*) – The traceback to send as a message.
- **self** (*object or None*) – If `func` is a method, this will be the `__self__` value bound to it, otherwise `None`.
- ***args** – The positional arguments passed to `func` that caused `err`.
- ****kwargs** – The keyword arguments passed to `func` that caused `err`.

CHAPTER
TWENTYTHREE

LLAMA.COM PACKAGE

Utilities for communicating with remote resources.

23.1 llama.com.dl package

Module for downloading publicly-accessible files.

```
llama.com.dl.download(url: str, filename: str, sha256sum: Optional[str] = None, tries: int = 10, mkdirs: bool = False)
```

Download the file located at `url` and save it to path `filename`. If `sha256sum` is specified, check the downloaded file's sha256 hex digest against that value and raise an `IOError` if they do not match. If `mkdirs` is `True`, make any intermediate directories as required. Give up after unsuccessfully trying `tries` times.

23.2 llama.com.do package

Provision or destroy a bunch of digitalocean servers. User will be asked for confirmation before creation or destruction of droplets proceeds.

```
llama.com.do.create_droplets(names, image='gwhen.com-post-er13', keys=None, tags=())
```

Create droplets with the given names from the given snapshot. Asks for user confirmation before creating. If `keys` is specified as a list, use ssh keys whose name or ID exactly equal one of the given keys. Specify tags to apply to droplets by name in `tags`.

```
llama.com.do.destroy_droplets(names, tags=())
```

Destroy droplets matching the given names. If `names` is an empty list, destroy all droplets with at least one of the tags contained in `tags`. Asks for user confirmation before destroying.

```
llama.com.do.droplet_header(columns)
```

Return a header row with column names.

```
llama.com.do.droplet_row_fmt(columns)
```

Get a format string for a given list of columns. Those columns will be inserted into the format string in order.

```
llama.com.do.get_ssh_keys()
```

Get a list of all SSH keys.

```
llama.com.do.get_tags()
```

Get a list of all available Droplet tags.

```
llama.com.do.new_droplet(name, image='gwhen.com-post-er13', ssh_keys=None)
```

Return a Droplet object with the given name from the given snapshot. Use the returned Droplet object to actually create a corresponding droplet on DigitalOcean. Specify `ssh_keys` to add to the droplet (default: no keys).

```
llama.com.do.print_droplets(droplets, columns, header_rows=True)
```

Print out a list of droplets in a nice format.

```
llama.com.do.print_ssh_keys(ssh_keys, header_rows=True)
```

Print out a list of ssh_keys in a nice tabular format.

23.3 llama.com.email package

Utility functions and abstract `FileHandler` bases for communicating via email.

```
class llama.com.EmailReceipt(eventid_or_fh, rundir=None)
```

Bases: `llama.com.utils.UploadReceipt`

A log file created as a side effect of emailing some text (e.g. for submission of GCN notices or circulars). This is sufficiently formulaic that only an uploaded file, recipient list, and (optional) veto condition must be specified in each subclass.

abstract property recipients

A list of email addresses that should receive this email.

property send_as_attachment

If this returns True, the UPLOAD will be submitted as an email attachment rather than as the body of the email, e.g. for PDF or image files. Returns False unless overridden by a subclass.

abstract property subject

A subject line for this email.

```
llama.com.email.utcnow()
```

Return a new datetime representing UTC day and time.

23.4 llama.com.gracedb package

Safely import GraceDB, registering a warning if the import fails.

```
class llama.com.gracedbGraceDb(*args, **kwargs)
```

Bases: `ligo.gracedb.rest.GraceDb`

A subclass of `ligo.gracedb.rest.GraceDb` that refreshes LLAMA GraceDb credentials in the event of an authentication error before retrying requests. Also checks for installed, non-expired credentials before initializing and automatically installs LLAMA keytab and refreshes Kerberos principal if necessary. Only does this if environmental variable `LLAMA_GRACEDB_AUTH` is set to an S3 key for a valid LIGO robot keytab; otherwise, behaves just like `ligo.gracedb.rest.GraceDb` (with some extra logging on authentication errors).

`ligo.gracedb.rest.GraceDb` docstring below:

GraceDb REST client class.

Provides a client object for accessing the GraceDB server API. Various methods are provided for retrieving information about different objects and uploading information.

Lookup of user credentials is done in the following order:

1. If provided, import X.509 credentials from the certificate-key pair or combined proxy file provided in the `cred` keyword arg.
2. If provided, use the username and password provided in the keyword arguments.
3. If the `X509_USER_CERT` and `X509_USER_KEY` environment variables are set, load the corresponding certificate and key.

4. If the X509_USER_PROXY environment variable is set, load the corresponding proxy file.
5. Look for a X.509 proxy from ligo-proxy-init in the default location (/tmp/x509up_u\${UID}).
6. Look for a certificate and key file in \$HOME/.globus/usercert.pem and \$HOME/.globus/userkey.pem.
7. Look for a username and password for the server in \$HOME/.netrc.
8. Continue with no authentication credentials.

Parameters

- **url** (str, optional) – URL of server API root.
- **cred** (tuple or str, optional) – a tuple or list of (/path/to/cert/file, /path/to/key/file) or a single path to a combined proxy file. Used for X.509 authentication only.
- **username** (str, optional) – username for basic auth.
- **password** (str, optional) – password for basic auth.
- **force_noauth** (bool, optional) – set to True if you want to skip credential lookup and use this client without authenticating to the server.
- **fail_if_noauth** (bool, optional) – set to True if you want the client constructor to fail if no authentication credentials are provided or found.
- **api_version** (str, optional) – choose the version of the server API to use. At present, there is only one version, but this argument is provided with the expectation that this will change in the future.
- **reload_certificate** (bool, optional) – if True, your certificate will be checked before each request whether it is within reload_buffer seconds of expiration, and if so, it will be reloaded. Useful for processes which may live longer than the certificate lifetime and have an automated method for certificate renewal. The path to the new/renewed certificate **must** be the same as for the old certificate.
- **reload_buffer** (int, optional) – buffer (in seconds) for reloading a certificate in advance of its expiration. Only used if reload_certificate is True.

Examples

Instantiate a client to use the production GraceDB server:

```
>>> g = GraceDb()
```

/se another GraceDB server:

```
>>> g = GraceDb(service_url='https://gracedb-playground.ligo.org/api/')
```

Use a certificate and key in the non-default location:

```
>>> g = GraceDb(cred=('/path/to/cert/file', '/path/to/key/file'))
```

event(graceid)

This method will automatically try to install LIGO proxy credentials if a `RuntimeError` or `ligo.gracedb.exceptions.HTTPError` is caught (if `GRACEDB_AUTH` is set to the S3 key of a valid LIGO robot keytab, set by environmental variable `LLAMA_GRACEDB_AUTH`).

Original docstring:

Get information about an individual event.

Args: graceid (str): GraceDB ID of the event

Returns: requests.models.Response

Raises:

ligo.gracedb.exceptions.HTTPError: if the response has a status code >= 400.

Example:

```
>>> g = GraceDb()  
>>> event_dict = g.event('T101383').json()
```

events(query=None, orderby=None, max_results=None, **kwargs)

This method will automatically try to install LIGO proxy credentials if a RuntimeError or ligo.gracedb.exceptions.HTTPError is caught (if GRACEDB_AUTH is set to the S3 key of a valid LIGO robot keytab, set by environmental variable LLAMA_GRACEDB_AUTH).

Original docstring:

Search for events which match a query.

Information on forming queries is available here: <https://gracedb.ligo.org/documentation/queries.html>

Args: query (str, optional): query string. orderby (str, optional): field to order the results by. max_results (int, optional): maximum number of results to

return (default: all).

Returns: Iterator[dict]

An iterator which yields individual event dictionaries.

Raises:

ligo.gracedb.exceptions.HTTPError: if the response has a status code >= 400.

Example:

```
>>> g = GraceDb()  
>>> for event in g.events('ER5 submitter: "gstlalcbc"'):  
...     print(event['graceid'], event['far'], event['gpstime'])
```

files(object_id, filename='', *args, **kwargs)

This method will automatically try to install LIGO proxy credentials if a RuntimeError or ligo.gracedb.exceptions.HTTPError is caught (if GRACEDB_AUTH is set to the S3 key of a valid LIGO robot keytab, set by environmental variable LLAMA_GRACEDB_AUTH).

Original docstring:

Get a list of files or download a file associated with an event or superevent.

If filename is not provided, get a list of available files associated with the event or superevent. If filename is provided, download the contents of that file.

Args: object_id (str): event graceid or superevent ID. filename (str, optional): name of file to download.

Returns: requests.models.Response

When `filename` is not specified, `response.json()` contains a dict with file basename keys and full file URL values. When `filename` is specified, use `response.read()` to get the contents of the file.

Raises:

`ligo.gracedb.exceptions.HTTPError: if the response has a status code >= 400.`

Examples: Get a list of files:

```
>>> g = GraceDb()
>>> event_files = g.files('T101383').json()
>>> for filename in list(event_files):
...     # do something
...     pass
```

Get a file's content:

```
>>> outfile = open('my_skymap.png', 'w')
>>> r = g.files('T101383', 'skymap.png')
>>> outfile.write(r.content)
>>> outfile.close()
```

`logs(object_id, log_number=None, *args, **kwargs)`

This method will automatically try to install LIGO proxy credentials if a `RuntimeError` or `ligo.gracedb.exceptions.HTTPError` is caught (if `GRACEDB_AUTH` is set to the S3 key of a valid LIGO robot keytab, set by environmental variable `LLAMA_GRACEDB_AUTH`).

Original docstring:

Get log entries associated with an event or superevent.

If `log_number` is specified, only a single log message is returned. Otherwise, all log messages attached to the event or superevent in question are returned.

Args: `object_id` (str): event graceid or superevent ID. `log_number` (int, optional): ID number (N) of the log entry to retrieve.

Returns: `requests.models.Response`

Raises:

`ligo.gracedb.exceptions.HTTPError: if the response has a status code >= 400.`

Examples:

Get all log messages:

```
>>> g = GraceDb()
>>> response_dict = g.logs('T101383').json()
>>> print "Num logs = %d" % response_dict['numRows']
>>> log_list = response_dict['log']
>>> for log in log_list:
...     print log['comment']
```

Get a single log message:

```
>>> g = GraceDb()
>>> log_info = g.logs('T101383', 10).json()
```

superevent(*superevent_id*)

This method will automatically try to install LIGO proxy credentials if a `RuntimeError` or `ligo.gracedb.exceptions.HTTPError` is caught (if `GRACEDB_AUTH` is set to the S3 key of a valid LIGO robot keytab, set by environmental variable `LLAMA_GRACEDB_AUTH`).

Original docstring:

Get information about an individual superevent.

Args: `superevent_id` (`str`): GraceDB ID of the superevent.

Returns: `requests.models.Response`

Raises:

`ligo.gracedb.exceptions.HTTPError`: if the response has a status code ≥ 400 .

Example:

```
>>> g = GraceDb()  
>>> superevent = g.superevent('S181224a').json()
```

superevents(*query*=", *orderby*=`None`, *count*=`None`, *columns*=`None`, *max_results*=`None`)

This method will automatically try to install LIGO proxy credentials if a `RuntimeError` or `ligo.gracedb.exceptions.HTTPError` is caught (if `GRACEDB_AUTH` is set to the S3 key of a valid LIGO robot keytab, set by environmental variable `LLAMA_GRACEDB_AUTH`).

Original docstring:

Search for superevents which match a query.

Information on forming queries is available here: <https://gracedb.ligo.org/documentation/queries.html>

Args: `query` (`str`, optional): query string. `orderby` (`str`, optional): field to order the results by. `count` (`int`, optional): each generator iteration will yield

this many objects (default determined by the server).

columns (`list[str]`, optional): list of attributes to return for each superevent (default: all).

max_results (`int`, optional): maximum number of results to return (default: all).

Returns: `Iterator[dict]`

An iterator which yields individual superevent dictionaries.

Raises:

`ligo.gracedb.exceptions.HTTPError`: if the response has a status code ≥ 400 .

Example:

```
>>> g = GraceDb()  
>>> for s in g.superevents(query='is_gw: True', orderby=  
... ['-preferred_event'], columns=['superevent_id', 'events']):  
...     print(s['superevent_id'])
```

```
writeLog(object_id, message, filename=None, filecontents=None, tag_name=[], displayName=[], *args, **kwargs)
```

This method will automatically try to install LIGO proxy credentials if a `RuntimeError` or `ligo.gracedb.exceptions.HTTPError` is caught (if `GRACEDB_AUTH` is set to the S3 key of a valid LIGO robot keytab, set by environmental variable `LLAMA_GRACEDB_AUTH`).

Original docstring:

Create a new log entry associated with an event or superevent.

Args: `object_id` (str): event graceid or superevent ID. `message` (str): comment to post. `filename` (str, optional): path to file to be uploaded.

Use '`-`' to read from stdin.

filecontents (file, optional): handler pointing to a file to be read and uploaded. If this argument is specified, the contents will be saved as `filename` on the server.

tag_name (str or list[str], optional): tag name or list of tag names to be applied to the log message.

displayName (str or list[str], optional): tag display string or list of display strings for the tag(s) in `tag_name`. If provided, there should be one for each tag. Not applicable for tags which already exist on the server.

Returns: `requests.models.Response`

Raises:

`ligo.gracedb.exceptions.HTTPError`: if the response has a status code ≥ 400 .

Example:

```
>>> g = GraceDb()
>>> r = g.writeLog('T101383', 'Good stuff.', '/path/to/plot.png',
...     tag_name='analyst_comments')
>>> print r.status_code
201
```

llama.com.gracedb.bashlines()

Get the lines in `~/.bashrc`

llama.com.gracedb.gracedb_auth_wrapper(func)

A wrapper for methods connecting to GraceDb. If `GRACEDB_AUTH` is set to the S3 key for a LIGO robot keytab (controlled by setting environmental variable `LLAMA_GRACEDB_AUTH` to that key), this method will catch `RunTime` and `ligo.gracedb.exceptions.HTTPError` errors and then try to refresh authentication credentials before attempting to call the wrapped function a second time.

llama.com.gracedb.install_keytab()

Install the keytab, granting access to GraceDB from this device.

llama.com.gracedb.keytab()

Download the Kerberos keytab if it does not exist locally and return the Path to that keytab. Raises a `KeyError` if `KEYTAB` is None (due to `LLAMA_GRACEDB_AUTH` not being set to a LLAMA S3 key for a valid LIGO robot keytab).

llama.com.gracedb.uninstall()

Uninstall keytab, run `kdestroy`, and remove env variable declarations from `.bashrc`.

23.5 llama.com.s3 package

Tools for uploading to/downloading from AWS S3 APIs (including DigitalOcean's S3-interface clone for DigitalOcean Spaces).

If using DigitalOcean spaces, for example, You will need to [configure an access token¹⁰⁷](#) for DigitalOcean spaces and set the key and secret as environmental variables DIGITALOCEAN_SPACES_KEY and DIGITALOCEAN_SPACES_SECRET, respectively (this should work for AWS S3 as well, though of course the link for generating the tokens will be different).

```
class llama.com.s3.PrivateFileCacher(key, bucket='llama', localpath=None)
Bases: llama.com.s3.PrivateFileCacherTuple
```

Like `llama.utils.RemoteFileCacher` but for private files stored behind an AWS S3 interface. If the file is not present locally, it will be automatically downloaded to the `pathlib.Path` returned by `get()` (provided that you have API credentials with access permissions for that file).

Parameters

- **key (str)** – The key of the remote file object.
- **bucket (str, optional)** – The S3 bucket in which the file is stored.
- **localpath (str, optional)** – The (optional) local path at which to cache this resource. By default, will just be `/Users/s/.cache/llama/objects/filename` where `filename` is actually taken from the remote URL filename.

get()

If the file is not available locally, download it and store it at `localpath` (do nothing if present). Return `localpath`.

```
class llama.com.s3.PrivateFileCacherTuple(key, bucket, localpath)
```

Bases: tuple

bucket

Alias for field number 1

key

Alias for field number 0

localpath

Alias for field number 2

```
llama.com.s3.get_client(region_name='nyc3', endpoint_url='https://nyc3.digitaloceanspaces.com',
                        aws_access_key_id='RDTWT7GEBWMB23CYZDDR',
                        aws_secret_access_key='2qg7K3qVeNnnQ3nzflsQLnIuzTTEww3FIMv5mZr0/Z0',
                        **kwargs)
```

Get a boto3 client connecting to the given DigitalOcean Spaces/AWS S3 region and endpoint.

Parameters

- **region_name (str, optional)** – The server region. This is the geographical region in which your servers reside. Check your DigitalOcean or AWS account to find this.
- **endpoint_url (str, optional)** – The endpoint URL for your specific Spaces/S3 instance. Again, check your account to find this.
- **aws_access_key_id (str, optional)** – Your access key, generated on your account website. You can only view this when you create it, so if you lost track of an old version, just

¹⁰⁷ <https://cloud.digitalocean.com/settings/api/tokens>

delete it and make new credentials. For DigitalOcean, you can do this [here](#)¹⁰⁸. If not provided, will default to the value of the DIGITALOCEAN_SPACES_KEY environmental variable, or None if it doesn't exist (which will result in an authentication error).

- **aws_secret_access_key** (*str, optional*) – The secret corresponding to your aws_access_key_id. Create this at the same time you create your aws_access_key_id (see notes above). If not provided, will default to the value of the DIGITALOCEAN_SPACES_SECRET environmental variable, or None if it doesn't exist (which will result in an authentication error).
- ****kwargs** – Extra keyword arguments to pass to `boto3.session.Session.client`.

Returns client – A client for interacting with the specified Spaces/S3 instance using the specified credentials. You can use this client to interact with the S3 API for file storage, retrieval, permissions modifications, etc. See `boto3.session.Session.client` for more details on the interface.

Return type `boto3.session.Session.Client`

```
llama.com.s3.upload_file(filename, key, bucket='llama', public=False, tries=5, **kwargs)
Upload file to a DigitalOcean Spaces/AWS S3 bucket.
```

Parameters

- **filename** (*str*) – Local path to the file you wish to upload.
- **key** (*str*) – The object key, analogous to a remote file path; the remote file will be available at /<bucket>/<key>. You can put slashes in the key, which will be treated as subdirectories on the DigitalOcean web file browser.
- **bucket** (*str, optional*) – Name of the target bucket. For DigitalOcean Spaces, this is the name of the directory in the root Spaces directory, e.g. `bucket=llama` will put everything under /llama/ remotely.
- **public** (*bool, optional*) – is True, the file will be publicly-accessible.
- **tries** (*int, optional*) – How many times to try the upload before giving up due to errors.
- ****kwargs** – Keyword arguments will be passed to `get_client` to initialize it, overriding its defaults. Use this to specify access credentials and upload target.

Returns url – If `public` is True, the remote URL at which the resource can be publicly accessed; otherwise, None.

Return type str or None

23.6 `llama.com.slack package`

Utilities for interacting with slack. Used for slack upload filehandlers as well as various slack-based logging utilities.

```
llama.com.slack.alert_maintainers(msg, desc=None, recover=True)
```

Shortcut to send a message to the LLAMA maintainer of LLAMA functionality on the default LLAMA channel, tagging MAINTAINERS using `tag_user` and sending the message with `send_message`. Import this in other parts of the code to have a Slack-based logging/alert tool. Optionally provide the name of the calling module (or some other description) as `desc`. Since this is meant to be used elsewhere in the code (including parts of the code that may run on non-production servers), it will fail if no `SLACK_TOKENS['LLAMA']` is defined. If `recover=True`, suppress and log errors due to message sending failures; otherwise, raise errors as usual.

¹⁰⁸ <https://cloud.digitalocean.com/account/api>

llama.com.slack.client(*token*)

Return a SlackClient instance for interacting with Slack's API. Will default to the LLAMA slack organization (which will fail if you have not configured an auth token via environmental variables).

llama.com.slack.get_users(*organization*)

Get list of users for an organization. Particularly useful because each user's id value can be used to tag users, as is done in `tag_user`. Returns the API dict straight from `slack.WebClient`.

llama.com.slack.search_users(*organization, queries*)

Fuzzy search for Slack users.

Parameters

- **organization (str)** – The Slack organization name to search for users.
- **queries (list)** – A list of string queries, each of which should match one unique user. Matches existing users for `organization` based on display names, real names, or Slack IDs that match the values of `queries`. Case-insensitive for every field except the ID. Will only return users who are not deleted.

Returns `users` – A list of Slack API user dictionaries matching `queries`, which can be used to tag them in messages using the values corresponding to their `id` keys.

Return type list**Raises**

- **slack.errors.SlackApiError** – If you are not authenticated or else there is some other error while calling `get_users`.
- **ValueError** – If your query returns multiple users or no users, or if some of the queries correspond to the same user.

llama.com.slack.send_message(*organization, message, channel=None, recover=False*)

Send a simple text message to `channel` (default: first channel registered for `organization`) in `organization`. Returns the response dict from `slack`. If `recover=True`, suppress and log errors due to message sending failures; otherwise, raise errors as usual.

llama.com.slack.tag_users(*organization, queries, recover=True*)

Get a string of text that can be used to tag users for a specific Slack organization. Stick this text somewhere in your message body to tag people and alert them. Same interface as `search_users`, but you can optionally recover from a failure to find matching users by specifying `recover=True`.

23.7 llama.com.utils module

Classes and utilities for communicating with external organizations (e.g. downloading inputs and uploading results, sending alerts, etc.). Pretty much everything having to do with data transfer should be inheriting from primitives in this package.

class llama.com.utils.UploadReceipt(*eventid_or_fh, rundir=None*)

Bases: `llama.filehandler.FileHandler`, `llama.filehandler.mixins.OnlineVetoMixin`, `llama.filehandler.mixins.ObservingVetoMixin`

A log file created as a side effect of uploading some other original file to some remote server (like gw-astronomy.org or gracedb). The `generate()` function should contain instructions on how to upload the original file and should log helpful information to the `UploadReceipt` file. The presence of an `UploadReceipt` file should be taken as evidence that an upload has succeeded.

Define the file that is to be uploaded as `UPLOAD` when subclassing `UploadReceipt`.

```
CLASSNAME_FMT = None
FILENAME_FMT = None
UPLOAD = None
class_vetoes = ((<function upload_flag_false>, None),)
classmethod decorator_dict(upload)
```

See `UploadReceipt.upload_this`. Takes the decorated `FileHandler` class as its first argument. By default takes no additional arguments.

Parameters `upload` – The decorated `FileHandler` class that is being registered for upload.

Returns `newclassdict` – A dictionary that can be passed to `type` to specify the attributes of a new class.

Return type dict

```
classmethod set_class_attributes(subclass)
```

See `FileHandler.set_class_attributes`; this method additionally sets the `FILENAME` and `DEPENDENCIES` attributes based on `subclass.UPLOAD`.

```
classmethod upload_this(*args, namespace=None, **kwargs)
```

Return a decorator that automatically uploads `FileHandler` classes by implementing this class with that `FileHandler` class as the `UPLOAD` attribute. Additional attributes are set by `cls.decorator_dict` using the `FileHandler` class decorated by the returned decorator, `*args`, and `**kwargs`.

Parameters

- `*args` – Positional arguments to pass to `cls.decorator_dict` (after the `FileHandler` class decorated by the returned decorator, which takes the first positional argument position).
- `namespace(dict, optional)` – The namespace to which the new `UploadReceipt` created by decorator will be added. The default behavior is to define `UploadReceipt` in the global namespace of the calling frame (so that using the decorator is equivalent to manually defining the `UploadReceipt` class with the decorated `FileHandler` as its `upload` immediately after defining the `FileHandler` to be decorated).
- `**kwargs` – Keyword arguments to pass to `cls.decorator_dict`.

Returns `decorator` – A decorator that takes a `FileHandler` and returns an implementation of this class with the decorated `FileHandler` as its `UPLOAD` file (along with any other implementation attributes calculated by this class's `decorator_dict` function using the new `FileHandler` class, `*args`, and `**kwargs` as input). The generated classname will be named `cls.CLASSNAME_FMT.format(upload)` (where `upload` is the decorated `FileHandler`) and will be added as a variable with that name to `namespace`.

Return type func

Raises

- `TypeError` – If there is only one positional argument provided which is itself a `type` (since this usually indicates that the developer forgot to `call` this method to get the *actual* decorator and instead applied this function as if it were a decorator to the `FileHandler` class they intend to decorate).
- `Exception` – `decorator_dict` can, in principle, throw other exceptions as well. See the specific implementation for details.

```
llama.com.utils.upload_flag_false(eventdir)
```

Return whether a trigger directory has its “`UPLOAD`” flag set to “`false`”.

LLAMA.DETECTORS MODULE

A registry of detectors providing uniform naming conventions.

24.1 Available Detectors

name	abbrev	fullname
IceCube	i3	IceCube
LVC	lvc	LVC
Fermi	frm	Fermi
ZTF	ztf	Zwicky Transient Facility
LLAMA	lma	Low-Latency Algorithm for Multimessenger Astrophysics

```
class llama.detectors.Detector(name, abbrev, fullname=None, url=None, summary=None, description='',
                               citations=frozenset({}))
Bases: llama.detectors.TableRowRepresentable
```

An object describing some sort of detector (or, more generally, a source of data). Use these instances to provide uniform naming conventions, data source descriptions, and other information across the pipeline. You shouldn't create `Detector` instances outside of the main detectors file since they are meant to be stored in the same namespace for shared use throughout the pipeline. On instantiation, an assertion runs to ensure that there are no name collisions or inconsistencies and to register the new detector name in the `detectors` namespace. This allows the `detectors` submodule to provide a convenient, unified interface for accessing all defined `Detector` instances and avoid runtime complications. This name disambiguation is important to keep automatically-generated code and documentation uniform and unambiguous with respect to detector references.

Parameters

- **name (str)** – The name of the detector. Must be a valid name for a python variable, and should correspond to the name of the `Detector` in the `detectors` submodule.
- **abbrev (str)** – A **very** short abbreviation (think 3 characters) for this detector. Must be a valid name for a python variable. This will be used for dynamically-generated subclasses and filenames.
- **fullname (str, optional)** – The full name of the detector (useful for longer detector names). If `name` is an acronym or other abbreviation, then `fullname` should spell out what `name` stands for. If not provided, this will take the same value as `name`.
- **url (str, optional)** – A URL pointing to this detector's homepage (or whatever page best summarizes this detector/links to other good information about the detector/offers data access). (default: `None`)

- **summary** (*str, optional*) – A short (think: one or two lines) sentence that describes the detector. If not provided, this will take the same value as `name`.
- **description** (*str, optional*) – An extended (think: up to a few paragraphs) description of the detector from which a new developer could gain cursory understanding of the data source.
- **citations** (*ImmutableDict, optional*) – A dictionary of citations which a developer or user can use to learn more about the detector. Map article titles/names (`str`) to URLs (`str`) at which those articles can be found, e.g. NASA ADS or arXiv links.

Raises `TypeError` – If this detector clashes in `name` or `abbrev` with another existing detector.

```
class llama.detectors.DetectorTuple(name, abbrev, fullname, url, summary, description, citations)
```

Bases: tuple

abbrev

Alias for field number 1

citations

Alias for field number 6

description

Alias for field number 5

fullname

Alias for field number 2

name

Alias for field number 0

summary

Alias for field number 4

url

Alias for field number 3

```
class llama.detectors.TableRowRepresentable(name, abbrev, fullname, url, summary, description, citations)
```

Bases: `llama.detectors.DetectorTuple`

A class for printing `DetectorTuple` rows in docstring format for a table. *This superclass of `Detector` implements table printing for ‘`DetectorTuple`’ instances in such a way that it can be reused to generate the header rows of the ‘`llama.detectors`’ module docstring.*

```
TABLE_COLUMN_WIDTHS = DetectorTuple(name=13, abbrev=13, fullname=65, url=None, summary=None, description=None, citations=None)
```

doctable_row()

Get a row describing this `DetectorTuple` for inclusion in a table of all detectors. *This method is defined in a separate class from `Detector` because it is also used (in a hacky way) to generate the table header.*

```
llama.detectors.doc_table_header()
```

Get the header rows for the table of `Detector` instances in `llama.detectors.__doc__`.

CHAPTER
TWENTYFIVE

LLAMA.DEV PACKAGE

Developer tools and scripts.

25.1 `llama.dev.background` package

Tools and scripts for running background sensitivity studies.

25.1.1 `llama.dev.background.pvalue` package

Take the output files made by `llama dev background table` for each population you are interested in and collate them into a single HDF5 file for p-value calculations. All files are expected to be in the current directory, and the output will be in this directory as well.

Both input and output files correspond to background significance values for various source populations. The HDF5 file simply contains a dictionary of ascending-order arrays whose values are the test statistics of background simulations and whose keys are the name of the background population in a specific hierarchical order. These keys are automatically selected when `populations-hdf5-collater` is run in this directory by splitting the source `.txt` table filenames on hyphens (-). For example, if the populations folder containg `bns-design.txt` `bns-o2.txt`, the output HDF5 populations file will have the following dictionary in it:

```
{  
    "bns": {  
        "design": array([...]),  
        "o2": array([...])  
    }  
}
```

The output HDF5 file is saved to the current directory by default, but if you want to use it for LLAMA p-value calculations, make sure to upload both the tarballed populations directory and the compressed HDF5 file to cloud storage using `llama dev upload` and then store the new URLs in the `RemoteFileCacher` instances used for `llama.files.coinc_significance.NEUTRINO_POPULATIONS` and `llama.files.coinc_significance.NEUTRINO_POPULATIONS_TARBALL`.

To summarize: you want to download the existing population tarball, decompress it with `tar -xzf populations.tar.gz` (assuming it's named `populations.tar.gz`, of course), move your `llama dev background table` output text files to that directory, name the new tables in the expected format of `{SEARCH_TYPE}-{POPULATION}-{LVCRUN}-{GWDETECTORS}.txt` (e.g. `opa-bbh-o3-H1V1.txt`), run this script in that directory, put the text tables in a compressed tarball called e.g. `populations.tar.gz` by running `tar -czf populations.tar.gz populations` in the containing directory, upload both that and the HDF5 file to the cloud with `llama dev upload`, and then use the URLs in the manifest printed by `llama dev upload` to update the values

of NEUTRINO_POPULATIONS and NEUTRINO_POPULATIONS_TARBALL in the `llama.files.coinc_significance` source code.

25.1.2 `llama.dev.background.table` package

Make a table with the coincident significance values as rows (one EVENT per row). Works on doublet neutrinos too (as long as the directory structure is as expected).

25.1.3 `llama.dev.background.table_singles` package

Script for collating results of background sensitivity runs into usable formats.

25.2 `llama.dev.clean` package

Clean up the LLAMA output directory, rotating out-of-date auxilliary files out of the way and archiving them on remote storage and setting event flag values for matching events to the values specified in `--flags`. You will need to specify a pattern for matching events that need to be cleaned using the `run` positional argument and the `--downselect` flag (see `llama run -h` help documentation for details); for example, you can match events created more than 86400 seconds ago with `--downselect sec_since_v0_gt=86400` to clean up test events older than a day.

```
llama.dev.clean.cleanup_testevents(run: llama.run.Run, archive_dir: Optional[str] = None, dryrun: bool = False)
```

Clean up all events returned by `run.events` and move them to a dated subdirectory of `archive_dir` (defaults to `TEST_ARCHIVE`). You should apply any downselections to `run` so that `run.events` only returns test events older than the age you specify. If `dryrun` is `True`, no action will be taken, but the intended actions will be logged.

```
llama.dev.clean.set_flags_oldevents(run: llama.run.Run, dryrun: bool = False, flags: dict = frozenset({}))
```

Set event flags for all events returned by `run.events`. Apply any downselections you need to `run` before passing it. Use this to e.g. mark old events as no longer being able to upload files automatically.

25.3 `llama.dev.data` package

`llama data` tools are used to securely fetch, move, and store privileged data. Use these tools to cache private detector data in a safe way on LLAMA servers. You will not be able to use these scripts unless you have your authentication credentials for relevant services set properly.

25.3.1 `llama.dev.data.i3` package

Tools for fetching remote archival neutrino data from IceCube's servers. You will need IceCube login credentials to be able to run these scripts.

```
llama.dev.data.i3.available_versions(root: str = '/data/ana/analyses/gfu/')
```

Get a list of available GFU archival neutrino versions from the IceCube servers.

Parameters `root` (`str, optional`) – The root directory in which to search for neutrino archive versions.

Returns `version_dirs` – A list of relative subdirectories to `root` on the remote IceCube server. If you picked `root` correctly, this should be a list of neutrino archive version snapshots in ascending temporal order.

Return type List[str]

Raises ValueError – If I3_USERNAME is None (probably happened because you didn't set the I3_USERNAME to your IceCube server username).

Examples

You should be able to get archival neutrinos used in the early-o3 era:

```
>>> 'version-002-p04' in available_versions()
True
```

```
llama.dev.data.i3.fetch_archival_neutrinos(version_dir: str, dest: str, root: str =
    '/data/ana/analyses/gfu/')
```

Same as secure_transfer_to_s3, but store the files in the dest directory instead of uploading them to S3. Mainly used as an implementation tool for secure_transfer_to_s3. Raises a FileNotFoundError if dest is not an existing directory.

```
llama.dev.data.i3.secure_transfer_to_s3(version_dir: str, root: str = '/data/ana/analyses/gfu/')
```

Fetch data files containing archival IceCube neutrino track events from IceCube servers and upload it to non-public AWS S3/DigitalOcean Spaces storage. Returns information on the file locations that can be used for automatic authenticated data retrieval.

Parameters

- **version_dir (str)** – The name of the directory (as a relative path from root) in which the archival neutrinos you want are stored as .npy or .root files. Should be one of the return values of available_versions(root=root).
- **root (str, optional)** – The root directory in which to search for neutrino archive versions on the IceCube server.

Returns private_file_cachers – A code string that's a dict of llama.com.s3.PrivateFileCacher declarations pointing to the relevant files; you should paste this file into the llama.files.i3.json.ARCHIVAL_NEUTRINOS dictionary to enable access to these files.

Return type str

25.4 llama.dev.docs package

Tools for automating documentation workflows.

25.4.1 llama.dev.docs.cli package

(Semi-)automatically collect command-line interface tools with --help documentation and dump them into an rst file.

25.5 llama.dev.dv package

Execute actions across a bunch of DigitalOcean servers based on their names. Useful for background and sensitivity runs. Executes a shell command on each server using SSH. You can either specify shell scripts to run (in non-interactive mode, as if you were calling “ssh foo@bar ‘my shell command’”) or can provide your own shell command as an argument to the script.

class `llama.dev.dv.Command(commandname, local, args)`

Bases: `object`

A command that can be used on many Droplets. Can have arguments passed to it.

cleanup(env)

Run a cleanup command AFTER the parallel processes run for each server. Specified as a standard bash command in ‘CLEANUP=…’ anywhere in the file (including in a comment, which is useful for non-bash scripts with different variable setting syntax). This command is *ALWAYS* run once locally, regardless of how the vectorized commands run. STDOUT and STDERR are not piped.

cmdline(ip)

The command line arguments in a list format (as expected by the first argument of `subprocess.Popen`).

property default_str

Get the raw defaults spec string for this command from the original script (parsed into argparse arguments using `Command.defaults`).

property defaults

Get the default argparse args for this command. Specified as ‘DROPVEC=…’ anywhere in the file (including in a comment, which is useful for non-bash scripts with different variable setting syntax).

property help

Return the help string for this command. Specified as ‘HELP=…’ anywhere in the file (including in a comment, which is useful for non-bash scripts with different variable setting syntax).

run(droplet_name, ip, timeout, i, num_drops)

Run a command synchronously using `subprocess.Popen`.

Parameters

- **droplet_name** (`str`) – The name of the droplet to run on.
- **ip** (`str`) – The IP address of the droplet to run on.
- **timeout** (`float`) – The maximum execution time in seconds, after which the process will be killed.
- **i** (`int`) – The number of the droplet in the full list of droplets.
- **num_drops** (`int`) – The total number of droplets run.

Returns

- **stdout** (`str`) – STDOUT from the process.
- **stderr** (`str`) – STDERR from the process.
- **retval** (`str`) – The return value of the process.

setup(env)

Run a setup command BEFORE the parallel processes run for each server. Specified as a standard bash command in ‘SETUP=…’ anywhere in the file (including in a comment, which is useful for non-bash scripts with different variable setting syntax). This command is *ALWAYS* run once locally, regardless of how the vectorized commands run. STDOUT and STDERR are not piped.

`llama.dev.dv.apply_default_cli_args(args)`

Modify the command line arguments with the args provided in args. Returns the provided command line arguments modified using defaults from `get_parser` when neither CLI args nor args overrides them.

`llama.dev.dv argparse_to_dict(args)`

Get a dictionary representing an `argparse.Namespace`.

`llama.dev.dv.argset_string(args)`

Set a bunch of arguments as if they were passed through the command line of a bash script.

`llama.dev.dv.get_droplets(pattern)`

Get droplets whose names match the fnmatch pattern `pattern`.

`llama.dev.dv.get_parser()`

Get an ArgumentParser with this tool's command line interface.

`llama.dev.dv.print_defaults()`

Print default values of command line arguments for `dropvec` for a given command.

`llama.dev.dv.print_descriptions()`

Print descriptions of the available commands.

`llama.dev.dv.run_workers(cmd, droplets, fmt, prelaunch, connections, timeout)`

Run the given bash command string on the specified droplets, printing the given `fmt` string after each one completes, using up to `connections` concurrent connections while spending up to `timeout` seconds on each connection.

`llama.dev.dv.scripts()`

Print a list of available scripts.

25.6 `llama.dev.log` package

Tools for parsing log files to extract logged information.

25.6.1 `llama.dev.log.lvalert` package

Extracts LVAAlert messages from the `gwhen listen lvalert` log files.

25.7 `llama.dev.upload` package

Tools for uploading manifests of data files to a DigitalOcean Spaces/Amazon S3 object storage solution (for later user installation using `llama install`).

```
llama.dev.upload.upload_and_get_manifest(root: str = '.', glob: str = '**/*', key_prefix: str =
                                         'llama/objects/', key_uses_relpah: bool = False, bucket: str =
                                         'llama', public: bool = True, endpoint_url: str =
                                         'https://nyc3.digitaloceanspaces.com', dry_run: bool = False,
                                         **kwargs)
```

Upload files from the specified path to DigitalOcean Spaces/AWS S3 and return a manifest mapping the stored object URLs to local relative paths. The sha256 sum of the uploaded file will be the actual filename, allowing for versioning of files and avoiding redundant file uploads and downloads, with `key_prefix` prepended to aid in organization. Use this to offload large data files onto separate file storage and generate the `MANIFEST` constant (and related constants) for installation.

Parameters

- **root** (*str, optional*) – The path to the root directory that should be uploaded to cloud storage. All local paths in the returned manifest will be relative to this path as well. Can be relative or absolute.
- **glob** (*str, optional*) – The glob specifying which files to match from the provided **root**. By default, recursively matches all files in all subdirectories.
- **key_prefix** (*str, optional*) – A prefix to prepend to the uploaded files' sha256 sums in order to create their object keys (i.e. remote filenames). Note that this is just a prefix, so if you want it to act/look like a containing directory for uploaded files, you will need to make sure it ends with /.
- **key_uses_relpah** (*bool, optional*) – If True, put the relative filepath from **root** of each file as a prefix in front of the sha256 sum when generating the key. In the filesystem analogy, this would put your remote files (on DigitalOcean, at least) at /<bucket>/<key_prefix>/<relative-path>/<sha256sum>. Use this if you want it to be easier to find the file at a glance/want to organize things by filename on the object store (e.g. for one-off uploads); don't use this if you're planning on organizing things with the returned manifest.
- **bucket** (*str, optional*) – The DigitalOcean Spaces/AWS S3 bucket to upload files to. For DigitalOcean this is just the name of the directory in your root Spaces directory.
- **public** (*str, optional*) – Whether to make files public. If you specify **public=False**, the uploaded files will have None as their remote URLs in the returned manifest (which should not be surprising, since the returned manifest is intended for unauthenticated downloads). You want this to be True if you are uploading files for the purpose of public distribution.
- **endpoint_url** (*str, optional*) – The **endpoint_url** argument for `llama.com.s3.get_client`. Specifies which S3 service you are using.
- **dry_run** (*bool, optional*) – If provided, don't upload the file. Instead, print the manifest that would be generated and quit. Use this to see where your files will be uploaded before actually doing it.
- ****kwargs** – Keyword arguments to pass to `llama.com.s3.get_client` that set authentication parameters and choose the target space for uploads; see documentation for that function for details.

Returns manifest – A dictionary whose keys are local paths of uploaded files relative to the **root** argument and whose values are tuples of the remote upload URL and sha256 sum of the file described by the key. Use this manifest to later download and install the correct versions of the uploaded files with the correct directory structure. Looks like {filename: (url, sha256sum)}.

Return type Dict[str, Tuple[str, str]]

Examples

Try uploading some dummy files with known contents to a remote test directory to confirm that you have access rights.

```
>>> # coding: utf-8
>>> import os
>>> from llama.dev.upload import upload_and_get_manifest
>>> from tempfile import TemporaryDirectory
>>> from pathlib import Path
```

(continues on next page)

(continued from previous page)

```
>>> from requests import get
>>> from hashlib import sha256
>>> with TemporaryDirectory() as tmpdirpath:
...     tmpdir = Path(tmpdirpath)
...     with open(tmpdir/'foo', 'w') as foo:
...         _ = foo.write('bar')
...     with open(tmpdir/'baz', 'w') as baz:
...         _ = baz.write('quux')
...     manifest = upload_and_get_manifest(root=tmpdirpath, bucket='test',
...                                         key_prefix='llama/dev/upload/',
...                                         public=True)
>>> sha256(get(manifest['foo'][0]).content).hexdigest()
'fcde2b2edba56bf408601fb721fe9b5c338d10ee429ea04fae5511b68fbf8fb9'
>>> sha256(get(manifest['baz'][0]).content).hexdigest()
'053057fda9a935f2d4fa8c7bc62a411a26926e00b491c07c1b2ec1909078a0a2'
```


LLAMA.EVENT PACKAGE

A class for interacting with the files associated with a given event. This class contains methods for working with these files individually (with FileHandlers) or as a group. In particular, the update method can be used to abstractly update the files in an event directory with the latest available information. This approach is robust and allows for trivial modifications and additions to the types of data files associated with events.

```
class llama.event.Event(eventid_or_event, rundir='/Users/s/.local/share/llama/current_run', pipeline=None)
Bases: llama.event.EventTuple, llama.flags.FlagsMixin, llama.versioning.GitDirMixin
```

An event object, used to update and access data associated with a specific trigger (which receives its own directory or eventdir). FileHandler or another Event (or anything with ‘eventid’ and ‘rundir’ properties) as an input argument, in which case it will correspond to the same event as the object provided as an argument. One can also provide a module or dictionary containing FileHandlers, which will be used to create a FileGraph for the Event (i.e. it will specify which files should be made for this event). Defaults to the files module for now, though eventually this should be refactored out.

Parameters

- **eventid_or_event** (*str or EventTuple or llama.filehandler.FileHandlerTuple*) – This can be a string with the unique ID of this event (which can simply be a filename-friendly descriptive string for tests or manual analyses), in which case the next arguments, rundir and pipeline, will be used; OR, alternatively, it can be an EventTuple (e.g. another Event instance), FileHandlerTuple (e.g. any FileHandler instance), or any object with valid eventid and rundir attributes. In this case, those attributes from the provided object will be re-used, and the rundir argument will be ignored. This makes it easy to get a new Event instance describing the same underlying event but with a different Pipeline specified, or alternatively to get the Event corresponding to a given FileHandler (*though in this case you should take care to manually specify the ‘Pipeline’ you want to use!*).
- **rundir** (*str, optional*) – The rundir, i.e. the directory where all events for a given run are stored, if it differs from the default and is not specified by eventid_or_event.
- **pipeline** (*llama.pipeline.Pipeline, optional*) – The pipeline, i.e. the set of FileHandlers that we want to generate, if it differs from the default pipeline. If none is provided, use DEFAULT_PIPELINE.

Returns `event` – A new Event instance with the given properties.

Return type `Event`

Raises `ValueError` – If the eventid_or_event argument does not conform to the above expectations or if the rundir directory for the run does not exist, a ValueError with a descriptive message will be thrown.

property auxiliary_paths

Names of possible auxiliary paths in the directory that are used to track the state of the Event as a whole.

clone(*commit='HEAD'*, *rundir=None*, *clobber=False*)

Make a clone of this event in a temporary directory for quick manipulations on a specific version of a file.

Parameters

- **commit** (*str, optional*) – The commit hash to check out when cloning this event. If not specified, the most recent commit will be used. Unsaved changes will be discarded.
- **rundir** (*str, optional*) – The run directory in which to store the cloned event. If not specified, a temporary directory will be created and used. The contents of this directory will NOT be deleted automatically.
- **clobber** (*bool, optional*) – Whether this cloned event should overwrite existing state.

Returns

- **clone_event** (*llama.event.Event*) – A clone of this event. The full history is saved, but the specified **commit** is checked out. Any uncommitted changes in the working directory will not be copied over to the **clone_event**. If **clone_event** already seems to be a valid event with the correct **commit** hash, no further action will be taken (thus repeated cloning has little performance penalty).
- *Raises*
- *llama.versioning.GitRepoUninitialized* – If this is called on an Event that has not had its git history initialized.
- *IOError* – If this event already exists in the specified **rundir** and is checked out to a different hash, unless **clobber** is True, in which case that working directory will be deleted and replaced with the desired commit.

compare_contents(*other*)

Compare the file contents of this event to another event using `filecmp.cmpfiles` (though results are given as `FileHandler` instances rather than file paths). Use this to see whether two event directories contain the same contents under a given pipeline.

Parameters other (*Event, str*) – The other Event instance to compare this one to, or else a directory containing files that can be compared to this Event (though in that case the filenames must still follow the expected format).

Returns

- **match** (*FileGraph*) – A FileGraph for this Event whose files have the same contents as those corresponding to the *other* event.
- **mismatch** (*FileGraph*) – A FileGraph for this Event whose files have differing contents as those corresponding to the *other* event.
- **errors** (*FileGraph*) – A FileGraph for this Event whose corresponding files do not exist or otherwise could not be accessed for comparison (either for the files corresponding to this Event or the *other* one).

Raises ValueError – If the Pipeline instances of this Event and the *other* one are not equal, it does not make sense to compare them, and a `ValueError` will be raised.

creation_time()

The time at which this event directory was created (according to the underlying storage system). Note that you probably are more interested in `modification_time`.

property cruff_files

Return a list of files in the event directory that are not associated with any file handler nor with event state directories.

property eventdir

The full path to the directory containing files related to this event.

exists()

Check whether this event already exists.

property files

Get a FileGraph full of FileHandler instances for the files in this event with this particular pipeline.

classmethod fromdir(eventdir='.', **kwargs)

Initialize an event just by providing a filepath to its event directory. If no directory is specified, default to the current directory and try to treat that like an event. Note that the returned event will eliminate symbolic links when determining paths for rundir and eventid. Useful for quickly making events during interactive work.

Parameters

- **eventdir (str, optional)** – The event directory from which to initialize a new event.
- ****kwargs** – Remaining keyword arguments to pass to Event().

gpstime()

Return the GPS time of this event. Returns -1 if none can be parsed.

init()

Initialize the directory for this event, making sure it is in a proper state for processing data. Make sure the eventdir exists by creating it if necessary. Also initializes version control and set flags to the defaults specified in FlagsMixin.DEFAULT_FLAGS (which Event inherits).

Returns Returns this Event instance to allow command chaining.

Return type self

Raises ValueError – If the eventdir path exists but is not a directory or a link to a directory, we don't want to overwrite it to make an the directory.

modification_time()

The time at which this event directory was modified (according to the underlying storage system).

printstatus(loglevel=None)

Print a user-readable message indicating the current status of this event, or, optionally, log it at some log level. To simply print to STDOUT, call with no arguments. To use the logging system, specify the appropriate log level for the output using ‘debug’, ‘info’, ‘error’, ‘warning’, or ‘critical’. For example, if this is to be debug output, use:

```
>>> event = Event.fromdir()
>>> event.printstatus(loglevel='debug')
```

save_tarball(outfile)

Save this event and all its contents as a gzipped tarball. You should probably use a .tar.gz extension for the outfile name.

update(downselect)**

Generate any files that fit the FileGraph downselection criteria specified in downselect. By default, generate all files that have not been generated and regenerate all files that have been obsoleted because their data dependencies have changed. Returns True if files were updated, False if no files in need of update were found.

class llama.event.EventTuple(eventid, rundir, pipeline)

Bases: tuple

eventid

Alias for field number 0

pipeline

Alias for field number 2

rundir

Alias for field number 1

`llama.event.NOW(tz=None)`

Returns new datetime object representing current time local to tz.

tz Timezone object.

If no tz is specified, uses local timezone.

CHAPTER
TWENTYSEVEN

LLAMA.FILEHANDLER PACKAGE

Abstract definitions of FileHandler classes. FileHandlers provide methods for defining and working with data associated with a trigger.

```
class llama.filehandler.EventTriggeredFileHandler(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler
```

A data file that is downloaded or received as the result of an event notification from some external source, e.g. a post request or a GCN notification. The generate() method should be able to take any arguments necessary for the creation of this file, since this method is likely to be invoked by an event-handler script. These FileHandler classes should not have DEPENDENCIES because they are created by events external to the dependency graph.

DEPENDENCIES = ()

```
class llama.filehandler.FileGraph(mappable)
Bases: llama.classes.ImmutableDict, llama.classes.NamespaceMappable
```

Used to store a list of FileHandler instances, e.g. those associated with a particular GraceDB event. In that example, one might access the LvcSkymapHdf5 file handler associated with some event using

```
> event.files.LvcSkymapHdf5
```

Has the nice feature of being able to take a dictionary as an initialization argument and create a dot-notation accessible map from that dictionary's key-value pairs.

```
dependency_graph(outfile: Optional[str] = None, title: str = 'FileGraph', urls: Optional[str] = None,
bgcolor: str = 'black')
```

Return a graphviz .dot graph of the FileHandler instances in this FileGraph and their DEPENDENCIES on each other. Plot their status (whether the file exists) as well as their metadata. Optionally plot the graph to an output image file visualizing the graph.

Parameters

- **outfile (str, optional)** – If not provided, return a string in .dot file format specifying graph relationsIf an output file is specified, infer the filetype and write to that file.
- **title (str, optional)** – The name of the graph to use. If not provided, “FileGraph” will be used.
- **urls (str, optional)** – A format string for including URLs attributes for each FileHandler node. If provided, include a URL attribute pointing to each FILENAME for use on the summary page website, where the FILENAME of each FileHandler instance will be provided to the urls format string as the only format argument. The URL attribute is not compatible with all graphviz output formats (<http://www.graphviz.org/doc/info/attrs.html#d:URL>) so make sure to turn it off if publishing to an incompatible format.
- **bgcolor (str, optional)** – The background color to use for the generated plot.

Returns `dot` – The dependency graph in .dot format (can be used as input to dot at the command line). This is returned regardless of whether an outfile is specified.

Return type str

Raises

- **ValueError** – If the FileHandler instances in this FileGraph do not all refer to the same event.
- **Optional file extensions for outfile:** –
 - `dot` – just save the dotfile in .dot format.:
 - `png` – save the image in PNG format.:
 - `pdf` – save the image in PDF format.:
 - `svg` – save the image in svg format.:

downselect(*invert=False*, *reducer=<built-in function all>*, ***kwargs*)

Get a FileGraph of file handlers that match *all* of the provided criteria. Checks in this docstring are defined in DOWNSELECT_CHECKS.

Parameters

- **invert** (*bool*) – Invert results. (Default: False)
- **reducer** (*function*) – Specify the any builtin to match if any check passes. Specify `all` to match only when every check passes. (Default: `all`)
- **instanceof** (*type*) – The FileHandler must be an instance of this class.
- **type** (*type*) – The type of the FileHandler must exactly match.
- **typename** (*str or list*) – The FileHandler type's name must match this string (or one of these strings if given an iterable of strings).
- **extension** (*str or list*) – The file extension for the file name (or one of these strings if given an iterable of strings).
- **startswith** (*str or list*) – The file name starts with this string (or one of these strings if given an iterable of strings).
- **endswith** (*str or list*) – The file name ends with this string (or one of these strings if given an iterable of strings).
- **inname** (*str or list*) – The file name contains this substring (or one of these strings if given an iterable of strings).
- **nameis** (*str or list*) – The filename equals this string (or one of these strings if given an iterable of strings).
- **depends** (*type or list*) – Has this FileHandler class (or one of these FileHandler classes) as a dependency.
- **ancestor** (*type or list*) – Has this FileHandler class (or one of these FileHandler classes) as an ur dependency, i.e. steps AFTER this FileHandler.
- **descendent** (*type or list*) – Has this FileHandler class (or one of these FileHandler classes) as a descendent, i.e. steps BEFORE this FileHandler.
- **subgraph** (*type or list*) – Same as descendent, but also include FileHandlers that are instances of the query arguments. Defines a subgraph of the original FileGraph graph containing only the given FileHandler instances and their DEPENDENCIES.

- **dependsname** (*str or list*) – Has the `FileHandler` with this name (or one of these names if given an iterable of strings) as a dependency.
- **dependsfile** (*str or list*) – Has the `FileHandler` with this filename (or one of these names if given an iterable of strings) as a dependency.
- **exists** (*bool*) – Whether the returned `FileHandler` instances' output files exist.
- **cooldown** (*bool*) – Whether the returned `FileHandler` instances are currently cooling down.
- **intent** (*bool*) – Whether the returned `FileHandler` instances are currently being generated.
- **vetoed** (*bool*) – Whether the returned `FileHandler` instances have been permanently vetoed.
- **urvetoed** (*bool*) – Whether the returned `FileHandler` instances have had any of their ancestors vetoed. Does not check the instances themselves (combine with `vetoed` for that).
- **obsolete** (*bool*) – Whether the returned `FileHandler` instances' output files exist but are obsoleted due to newer input data being available. **NB** Obsolescence *implies* that the file exists! Will not include `FileHandler` results whose files don't exist.
- **selfobsolete** (*bool*) – Same as `obsolete`, but don't run obsolescence checks on each file's ancestors; only run checks relevant to each `FileHandler` instance.
- **depsmet** (*bool*) – Whether the returned `FileHandler` instances' `DEPENDENCIES` have been met. These can be generated immediately if they don't exist.
- **needregen** (*bool*) – Whether the returned `FileHandler` instances that have their `DEPENDENCIES` met and either exist (but are `obsolete`) or have not been generated yet.

Returns `matches` – A new `FileGraph` instance containing a subset of the `FileHandler` instances for this `FileGraph` that match the given downselection criteria.

Return type `FileGraph`

`status()`

Return a status description of this graph for use in status-checking scripts and webpages. Values are taken from `NODELEGEND_COLOR_FMT` so that they can be colored appropriately.

`update(**downselect)`

Generate any files that fit the `FileGraph` downselection criteria specified in `downselect`. By default, generate all files that have not been generated and regenerate all files that have been obsoleted because their data `DEPENDENCIES` have changed. Will only generate files that are *immediately* generateable; if some of the files you want to generate depend on files that haven't been generated yet, you'll need to keep running this method until you've generated each successive layer of dependencies (and, finally, your target files). You can do this by running the method repeatedly until it returns `False`.

Parameters `**downselect` – Keyword arguments that can be passed to `Event.downselect` to narrow down the set of `FileHandler` instances that should be generated.

Returns `files_were_generated_this_cycle` – A boolean indicating whether any files were generated. If you want to generate all possible files in this `FileGraph`, keep running this method in a loop until it returns `False`.

Return type `bool`

`class llama.filehandler.FileGraphTuple(eventid, rundir, pipeline)`
Bases: `tuple`

eventid

Alias for field number 0

pipeline

Alias for field number 2

rundir

Alias for field number 1

class `llama.filehandler.FileHandler(eventid_or_fh, rundir=None)`

Bases: `llama.classes.FileHandlerTuple`, `llama.classes.RequiredAttributeMixin`, `abc.ABC`, `llama.flags.FlagsMixin`, `llama.vetoes.VetoMixin`, `llama.versioning.GitDirMixin`, `llama.lock.LockMixin`, `llama.classes.RiderMixin`

A class for generating, opening, and checking existence of data files associated with these events. Specify the maximum amount of time that should be spent generating each file by setting the TIMEOUT attribute of the relevant implementation class. *Instances are immutable* other than their parent and graph attributes.

`FileHandler` instances can check whether their corresponding output data has been generated and stored. If the data in their input DEPENDENCIES have changed, they can dynamically check whether their corresponding outputs need to be regenerated. Use `DEP_CHECKSUM_KWARGS` to specify which subsets of input data are relevant to each `FileHandler`; these keyword arguments will be fed to the `checksum` methods of each dependency to see whether relevant subsets of input data have changed (causing the current version of the `FileHandler` instance to become obsolete and triggering automatic file regeneration on the next update).

The following class attributes must be defined in subclasses, either manually or programmatically (see: `FileHandler.set_class_attributes` and its implementations in subclasses).

FILENAME [str] The base filename for this filehandler as it will appear in an event directory.

DEPENDENCIES [Tuple[FileHandler]] A tuple of other `FileHandler` subclasses whose data this `FileHandler` uses in order to generate its own output.

MANIFEST_TYPES [Tuple[FileHandler]] A tuple of other `FileHandler` subclasses that are generated at the same time as this one. In other words, running `self.generate` for any of the subclasses in `MANIFEST_TYPES` will produce all of the files in that tuple.

UR_DEPENDENCIES [Tuple[FileHandler]] Return a list of ur-dependencies, i.e. `DEPENDENCIES` (of `DEPENDENCIES` etc.) of this `FileHandler`, i.e. `FileHandler` classes whose data is ultimately used (after some number of steps in the DAG) to generate this `FileHandler`. The list is ordered such that its files can be generated in order without encountering missing dependencies (i.e. items deepest in `cls.UR_DEPENDENCY_TREE` come first).

UR_DEPENDENCY_TREE [ImmutableDict] A dict of all `DEPENDENCIES` of `DEPENDENCIES` going back to the original input files that are ultimately required to generate this file. Maps `FileHandler` classes to dictionaries of their own ancestry trees recursively starting at the current `FileHandler`. The deepest items in the dictionary are the furthest `DEPENDENCIES` up the dependency graph (and consequently the files that must be generated first in order for the shallower files to be generated). See `FileHandler.UR_DEPENDENCIES` for a flattened, ordered version.

Parameters

- **eventid_or_fh** (str, `llama.Event`, or `llama.FileHandler`) – the `eventid` of the file handler or else another `FileHandler` or `Event` instance (though anything with `eventid` and `rundir` properties will work). If such an object is provided, the `rundir` will be inferred therefrom.
- **rundir** (str, optional) – The directory in which events from this run are being stored. If `eventid_or_fh` is an object with a `rundir` attribute, then that value of `rundir` will be used and this argument will be ignored. (See `DEFAULT_RUN_DIR` for default value.)

Raises `AssertionError` – If class constants in `cls.required_attributes` are not defined or if `cls.MANIFEST_TYPES` has any inconsistencies.

`COOLDOWN_PARAMS = CoolDownParams(base=60, increment=60, maximum=14400)`

`DEPENDENCIES = None`

`DEP_CHECKSUM_KWARGS = frozenset({})`

`FILENAME = None`

`MANIFEST_TYPES = None`

`TIMEOUT = 20`

`UR_DEPENDENCIES = None`

`UR_DEPENDENCY_TREE = None`

`are_dependencies_met()`

Check whether the data needed to generate this filetype exists. If there are no `DEPENDENCIES` (i.e. no input), this file cannot possibly be made (since outputs are considered to be pseudo-functional mappings from inputs, having no inputs means that no meaningful output can exist).

`are_ur_dependencies_met()`

Recursively check whether `DEPENDENCIES` for this file can be generated (by in turn running this same check on their `DEPENDENCIES`). In short, this is a check as to whether we can eventually get to generating this file or whether the required data to get to this node in the FileHandler DAG is simply not in the eventdir. If there are no `DEPENDENCIES` (i.e. no input), this file cannot possibly be made (since outputs are considered to be pseudo-functional mappings from inputs, having no inputs means that no meaningful output can exist).

`property auxiliary_paths`

Return all names of *possible* auxiliary (rider) files associated with this `FileHandler` instance. These are things like metadata, cooldown, veto, and locking files.

`checksum(**kwargs)`

Get the sha256 checksum of this file's contents. Use this to version files and check whether their contents have changed.

Parameters `kwargs (dict)` – (Ignored in the base `FileHandler` implementation). Subclass implementations can optionally use input arguments to identify relevant subsets of the file's contents for versioning or diff-checking purposes. In this way, checks can be made for changes on only specific subsets of file contents. This is useful for ignoring changes to unused input data when checking for obsolescence.

`compare_contents(other)`

Check whether the contents of this `FileHandler` instance's file are the same as the contents of the other `FileHandler` instance's file.

Parameters

- **other (`FileHandler`, `str`)** – The `FileHandler` to compare to this one or else a path to a file to compare to this one.
- **Returns –**
- **same (`bool`)** – Returns True if both files exist and have the same contents. Otherwise, returns False.

`delete()`

Delete this file if it exists, along with all of its rider files, and commit that change to version control.

dep_checksums()

Recalculate the sha256 sums of the contents of the input files (i.e. DEPENDENCIES) for this FileHandler instance's manifest_filehandlers (either to store them or to check whether they have changed from the stored values).

Returns

- **dep_checksums (dict)** – Keys are FileHandler.clsname values for each of this FileHandler class's DEPENDENCIES and values are the corresponding checksums of each file. These checksums uniquely determine the exact input files used and are suitable for creating snapshots of pipeline state.
- **dep_subset_checksums (dict)** – Same as checksums but with self.DEP_CHECKSUM_KWARGS applied to each dependency's checksum method to **only** check whether the data used by self.generate has changed. These checksums determine whether the DEPENDENCIES have changed *in a way that is meaningful to this specific filehandler* (since unused fields are ignored) and are suitable for determining whether a FileHandler needs to be regenerated based on the availability of new data. (To avoid unnecessary computation, these checksums are only calculated separately from checksums when a dependency has a set of DEP_CHECKSUM_KWARGS defined for it.)

diff_contents(other, force=False)

Return a diff of two text files. If the files are not text files (i.e. if their file extensions are not in TEXT_FILE_EXTENSIONS), prints “Binary files self and other differ” (with self and other replaced with their full file paths).

Parameters

- **other (FileHandler, str)** – The FileHandler to diff to this one or a path to a file to diff with this one.
- **force (bool, optional)** – Whether to force the diff (as if self and other are both text files) even if the files are not recognized as a form of text file.
- **Returns** –
- **diff (str)** – A textual diff of the two files' contents, provided they are both recognized as text files (or force is True); or else a message saying that they differ. Prints nothing if the files are the same.

property eventdir

The directory where data for the event this FileHandler corresponds to is stored.

exists()

Check whether the file associated with this handler has yet been generated.

property filename_for_download

Get a filename that includes the eventid, revision number, and version hash for this file (i.e. what version number this is in the version history; e.g. if three versions of *this* file exist in the version history, then this is version 3). If this file does not appear in the git history, it will be marked ‘v0’ and the hash will be ‘UNVERSIONED’. The output format is eventid, version, first 7 digits of commit hash, and filename, split by hyphens, so that the third version of skymap_info.json for event S1234a with git hash dedb33f would be called S1234a-v3-dedb33f-skymap_info.json. Use this for file downloads or files sent to other services in order to facilitate data product tracking outside the highly-organized confines of a pipeline run directory.

property fullpath

The full path to the file referred to by this FileHandler.

generate(*args, **kwargs)

Make the next version of a file, assuming it does not exist or is in need of updating. In the event that some sort of error causes file generation to fail, delete the output file (if it exists). You should ALWAYS use this instead of `_generate()` in order to generate and version files safely and atomically without risk of disrupting parallel file manipulations.

property graph

Get the `FileGraph` to which this `FileHandler` instance belongs. This can be set dynamically to associate a `FileHandler` with a given `FileGraph` (and hence a given `Pipeline`). If `graph` is not manually set, it defaults to the subgraph of this `FileHandler`.

is_obsolete(checked=None, ancestors=True)

Check whether this file exists but needs to be regenerated by seeing whether any of its `DEPENDENCIES` have updated their file contents since this file was made. This is a somewhat conservative check to see whether any files need to be regenerated automatically; it will return `False` if there is any ambiguity (in which case you will need to manually delete and regenerate child files). You can extend this definition with extra obsolescence criteria, but make sure to call `super` to keep this automatic regeneration functionality in response to regenerated `DEPENDENCIES`.

If the file is marked as “locked” (see: `llama.lock.LockHandler`), the file will **never** be marked obsolete. This provides a way to manually prevent file obsolescence in situations in which it does not apply.

If any of this file’s ancestors are obsolete, then this file will be marked obsolete. You can skip this check with `ancestors=False`.

Failing that, if the output file does not exist, or if its input `DEPENDENCIES` do not exist, it is not considered obsolete, and this method will return `False`. Likewise, if the file has no `DEPENDENCIES`, it cannot be naively obsoleted, and this method will return `False`.

Failing that, it will be marked obsolete if its `dep_subset_checksums()` [1] (see: `FileHandler.dep_checksums` second return value) have changed from their previously recorded values. If those checksums are not recorded, it will not be marked as obsolete.

Parameters

- **checked** (`dict, optional`) – A dictionary mapping `FileHandler` instances that have previously had their obsolescences checked mapped to whether they are yet obsolete; used internally to track whether this `FileHandler` instance’s `DEPENDENCIES` are obsolete without recomputing them.
- **ancestors** (`bool, optional`) – If `False`, don’t check whether ancestors are obsolete; only run this `FileHandler` instance’s obsolescence checks.

property manifest

A set of filenames generated by this `FileHandler` (not including temp files that should be deleted after generation). By default, this set only includes the filename for this `FileHandler`. If `generate` fails, these files will be removed to enforce atomicity.

In general, you should not need to modify this, since it will return filenames from the `FileHandler` instances in `self.manifest_filehandlers`, which should contain an exhaustive list of `FileHandler` instances associated with the `_generate` method that creates them.

property manifest_filehandlers

A set of `FileHandler` instances generated by this `FileHandler` (not including temp files that should be deleted after generation). By default, this set only includes the filename for this `FileHandler`. If `generate` fails, these files will be removed to enforce atomicity.

If you are making a bunch of `FileHandler` subclasses that are all generated with the same method, you should define an abstract base class for those `FileHandler` classes with a suitable `_generate` method and a `manifest_filehandlers` property that returns all of their filenames. The subclasses then only need

to return those `FileHandler` instances (and any other relevant methods and properties unique to each subclass `FileHandler`).

modtime(*ts=None*)

Get a `datetime` object with the modification time of this file, assuming it exists.

Parameters `ts` (*int or float, optional*) – If the file does not exist, return a `datetime` parsed from the UNIX timestamp given by `ts` if provided; otherwise, return `None`. Use this to provide a fallback modification time in cases when modification times might need to be compared between existing files and files whose existence is uncertain.

open(*mode='r'*)

Open this file in readonly mode and return the resulting object.

property parent

Get the `FileHandler` instance that created this `FileHandler`. Returns `None` if this `FileHandler` was created from scratch. Useful for things like running on clusters or temporary directories.

classmethod set_class_attributes(*subclass*)

Decorater for a new subclass that sets its `MANIFEST_TYPES` class attribute to the default `FileHandler` value without requiring them to be manually specified. Also determines the `UR_DEPENDENCIES` and `UR_DEPENDENCY_TREE` based on `subclass.DEPENDENCIES`. **NB: manually set class attributes will be overwritten by this decorator.**

Parameters `subclass` (*type*) – The subclass whose class attributes need to be set.

Returns `subclass` – The same decorated subclass.

Return type `type`

size()

Get the size in bytes of this file. Raises a `FileNotFoundException` if this file does not exist.

status(*obscheck=None*)

Return the status string (see `NODELEGEND_COLORS`) for a given `FileHandler` instance.

property subgraph

Get all ur `DEPENDENCIES` along with this `FileHandler` instance in a single `FileGraph` instance.

classmethod sync_shared_manifests(**filehandlers*)

When implementing a `FileHandler` with its own subclasses in its manifest (the easiest pattern for ensuring that the `FileHandler` classes in the manifest share a common `generate` method and `DEPENDENCIES`), use this decorator before each class definition to make sure that the base `FileHandler` (with the shared `generate` function) as well as its subclasses (which distinguish between its outputs by each having their own `FILENAME` attributes).

Parameters *`filehandlers` (`FileHandler` or `str`) – `FileHandler` subclasses that are generated together.

Returns The first `FileHandler` subclass from `filehandlers` (this allows the function to operate as a class decorator).

Return type `first_filehandler`

Raises

- **ValueError** – If provided `filehandlers` are not subclasses of this class or if no `filehandlers` are provided.
- **TypeError** – If provided `filehandlers` do not share the same `DEPENDENCIES` and `_generate` methods as this class, or if any of them lack a `FILENAME` attribute.

class `llama.filehandler.GenerateOnceMixin`Bases: `object`

An object that, once generated, never becomes obsolete; must be manually regenerated.

isObsolete(*checked=None*, ***kwargs*)

Because this class inherits from `GenerateOnceMixin`, it is never marked obsolete automatically; it must be manually regenerated. This function always returns False. See `FileHandler.isObsolete` for the meaning of `checked`.

class `llama.filehandler.JSONFile(eventid_or_fh, rundir=None)`Bases: `llama.filehandler.FileHandler`

A FileHandler abstract subclass providing tools to work with JSON dictionaries.

checksum(*fields=None*)

Get the sha256 checksum of this file's contents. Use this to version files and check whether their contents have changed. Optionally only use specific fields in generating the checksum to ignore irrelevant changes (e.g. when determining file obsolescence).

Parameters

- **fields** (*tuple or list, optional*) – A tuple of tuples of strings, with each sub-tuple containing strings or integers indexing into this JSON-file's contents (strings for `dict` fields, integers for `list` fields) to specify only relevant fields. See example below.
- **kwargs** (*dict*) – Remaining keyword arguments are ignored (see `FileHandler.checksum` note on `kwargs`).

Raises

- **KeyError** – If a field that is expected in a sub-dictionary is not found.
- **IndexError** – If an entry that is expected in a sub-list is not found.
- **TypeError** – If you attempt to index into a sub field that is not a dictionary using a string or if `fields` cannot be indexed into.

Examples

For a JSON dictionary like: `>>> foo = { ... "names": ["stef", "countryman"], ... "age": 27, ... "eyes": { ... "left": "brown", ... "right": "brown", ... } ... }`

You can calculate the checksum using *only* the first name and left eye-color values by using these `fields`:
`>>> fields = (... ('names', 0), ... ('eyes', 'left') ...)`

property `html_table`

Generate an HTML table representation of the JSON data contained in this `FileHandler`.

read_json()

Read in this JSON file as a dictionary.

llama.filehandler.NOW(*tz=None*)

Returns new datetime object representing current time local to `tz`.

tz Timezone object.

If no `tz` is specified, uses local timezone.

class `llama.filehandler.Status(status, stats, color)`Bases: `tuple`

```
color
    Alias for field number 2

stats
    Alias for field number 1

status
    Alias for field number 0

class llama.filehandler.TriggerList(eventid_or_fh, rundir=None)
    Bases: llama.filehandler.FileHandler
        A file that contains lists of triggers of some sort.

DETECTORS = None

abstract property num_triggers
    The number of triggers described by this file. Useful mostly for quickly determining if this trigger list is empty.

llama.filehandler.fromtimestamp()
    timestamp[, tz] -> tz's local time from POSIX timestamp.

llama.filehandler.recursive_obsolescence(func)
    Store is_obsolete values for repeated calls to make sure that we don't recompute them while recursively checking is_obsolete values.

llama.filehandler.utcnow()
    Return a new datetime representing UTC day and time.
```

27.1 `llama.filehandler.mixins` module

Mixins for FileHandler classes that add extra functionality.

```
class llama.filehandler.mixins.FileExtensionMixin
    Bases: abc.ABC
        For FileHandler abstract subclasses that have multiple possible output formats (and programmatically-generated filenames calculated therefrom in their implementation subclasses).

        The following class attributes must be defined in subclasses, either manually or programmatically (see: FileHandler.set_class_attributes and its implementations in subclasses).

        The file extension for this file as a string.

FILEEXT = None

class llama.filehandler.mixins.ObservingVetoMixin
    Bases: abc.ABC
        These files will not be generated for mock data, test events, nor anything that doesn't look like an online LVC trigger. Veto these files if 'test' or 'injection' appear anywhere in the event directory path, or if the eventid does not follow the real GraceID regex.

class_vetoes = ((<function role_flag_not_observation>, None), (<function
eventdir_path_contains_string_scratch>, None), (<function
eventdir_path_contains_string_test>, None), (<function
eventdir_path_contains_string_injection>, None))

class llama.filehandler.mixins.OnlineVetoMixin
    Bases: abc.ABC
```

Mixin for `FileHandler` classes.

Automatically veto on anything that doesn't look like a GraceDB online event. Use this to prevent accidentally querying GraceDB or other services for events that are meant to run on a local computer.

```
class_veto = ((<function online_flag_false>, None), (<function  
eventdir_path_contains_string_manual>, None))
```

```
llama.filehandler.mixins.eventid_is_not_real_event_graceid(eventdir)
```

Check whether the eventid BREAKS from the pattern used for the GraceIDs of real LVC events. Returns True if this eventid looks like a fake or injected GraceID (or just a manually-generated non-GraceDB ID).

```
llama.filehandler.mixins.eventid_is_not_valid_graceid(eventdir)
```

Check whether the eventid BREAKS from the pattern used for the GraceIDs of ANY (test OR observation) LVC events. Returns True if this eventid looks like a manually-generated non-GraceDB ID.

```
llama.filehandler.mixins.online_flag_false(eventdir)
```

Return whether a trigger directory has its “ONLINE” flag set to “false”.

```
llama.filehandler.mixins.role_flag_not_observation(eventdir)
```

Return whether a trigger directory has it's “ROLE” flag set to something other than “observation”.

CHAPTER
TWENTYEIGHT

LLAMA.FILES PACKAGE

A collection of FileHandler classes for a given pipeline.

```
class llama.files.Advok(eventid_or_fh, rundir=None)
Bases: llama.filehandler.EventTriggeredFileHandler, llama.filehandler.JSONFile
```

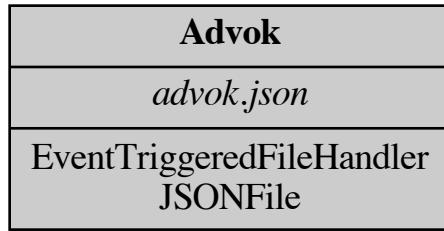


Fig. 1: `llama.files.advok`.`Advok` is created from external triggers. It therefore has no LLAMA-representable input dependencies, but instead acts as initial input for other `FileHandler` classes.

The existence of this file should be taken as an indication that the event has been marked as ADVOK. This may have occurred either through the actual application of the ADVOK label to the event on GraceDB, or through the sending of a GCN notice for this event (which has happened even when ADVOK was not applied, as in G298048), or when a LLAMA operator chooses to manually mark the event as EM followup ready by creating this file. Once this file exists, dependent files will assume that the event is legitimate and will carry on with any communication efforts they are programmed to perform.

Should be created either simultaneously with a GCN notice or simultaneously with an LVAAlert adding the ADVOK label to an event (these are the use cases at time of writing, anyway).

```
FILENAME = 'advok.json'
MANIFEST_TYPES = (<class 'llama.files.advok.Advok'>,)
UR_DEPENDENCIES = ()
UR_DEPENDENCY_TREE = frozenset({})
property alert_file_handler
    Get the file handler which tells us that this event is ready for EM followup.
property alert_type
    A string describing the provenance of this trigger, i.e. what "alert" caused us to create this event (and with it, this SkymapInfo file).
property time
    The time at which this became ADVOK.
```

```
class llama.files.CoincScatterI3LvcPdf(eventid_or_fh, rundir=None)
Bases: llama.files.coinc_plots.CoincScatterI3Lvc
```

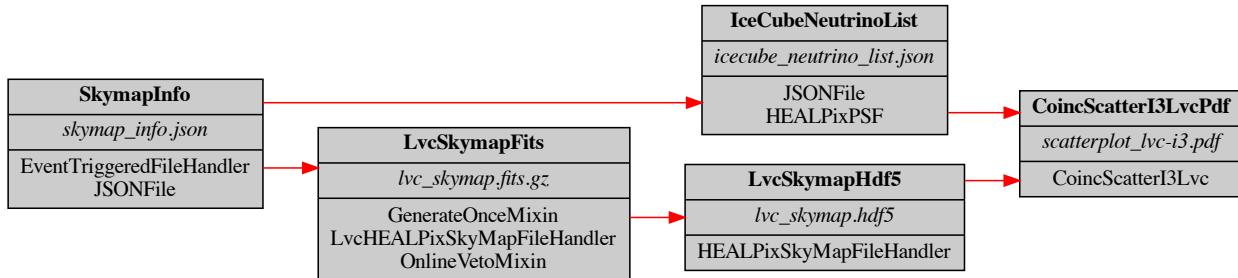


Fig. 2: Required input files for `llama.files.coinc_plots.CoincScatterI3LvcPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincScatterI3LvcPdf` to be generated.

Human-readable joint Neutrino/Gravitational Wave skymap plot in PDF format.

```
DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'	llama.files.i3.json.IceCubeNeutrinoList'>)

FILEEXT = 'pdf'

FILENAME = 'scatterplot_lvc-i3.pdf'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincScatterI3LvcPdf'>)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'	llama.files.lvc_skymap.LvcSkymapFits'>, <class
'	llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'	llama.files.i3.json.IceCubeNeutrinoList'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class
'	llama.files.i3.json.IceCubeNeutrinoList'>: ImmutableDict({<class
'	llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}), <class
'	llama.files.lvc_skymap.LvcSkymapHdf5'>: ImmutableDict({<class
'	llama.files.lvc_skymap.LvcSkymapFits'>: ImmutableDict({<class
'	llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})})})})

class llama.files.CoincScatterI3LvcPng(eventid_or_fh, rundir=None)
Bases: llama.files.coinc_plots.CoincScatterI3Lvc
```

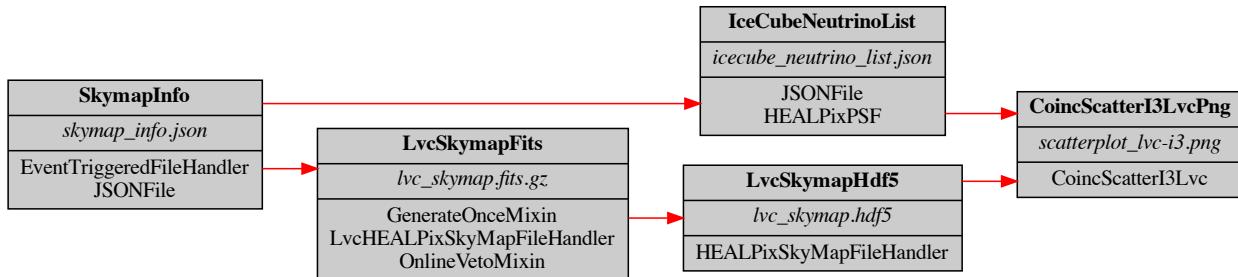


Fig. 3: Required input files for `llama.files.coinc_plots.CoincScatterI3LvcPng` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincScatterI3LvcPng` to be generated.

Human-readable joint Neutrino/Gravitational Wave skymap plot in PNG format.

```

DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)

FILEEXT = 'png'

FILENAME = 'scatterplot_lvc-i3.png'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincScatterI3LvcPng'>)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}), <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>: ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})})})})

class llama.files.CoincScatterZtfI3LvcPdf(eventid_or_fh, rundir=None)
Bases: llama.files.coinc_plots.CoincScatterZtfI3Lvc
    
```

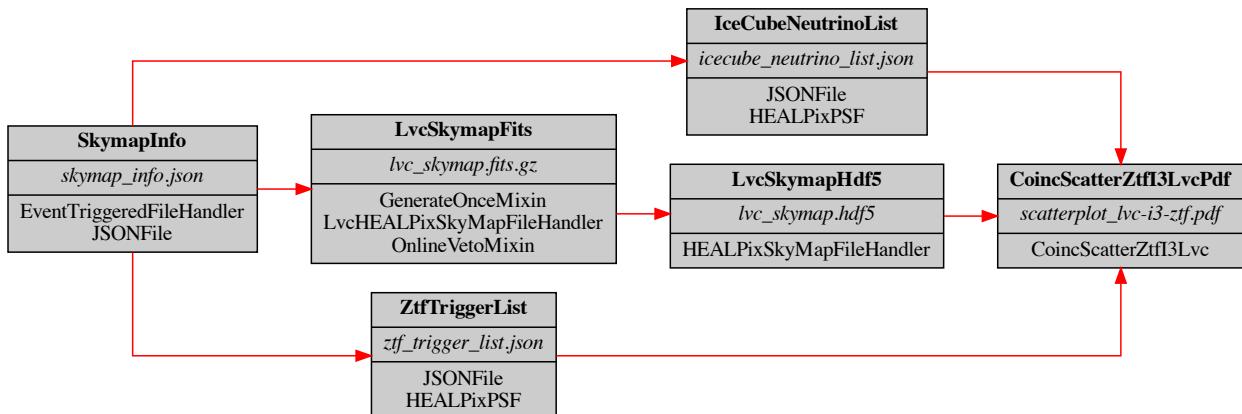


Fig. 4: Required input files for `llama.files.coinc_plots.CoincScatterZtfI3LvcPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincScatterZtfI3LvcPdf` to be generated.

Human-readable joint ZTF/Gravitational Wave/High Energy Neutrino skymap plot in PDF format.

```

DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>)

FILEEXT = 'pdf'

FILENAME = 'scatterplot_lvc-i3-ztf.pdf'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincScatterZtfI3LvcPdf'>)
    
```

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{})), <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>: ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}))}), <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}))})})

class llama.files.CoincScatterZtfI3LvcPng(eventid_or_fh, rundir=None)
Bases: llama.files.coinc_plots.CoincScatterZtfI3Lvc
    
```

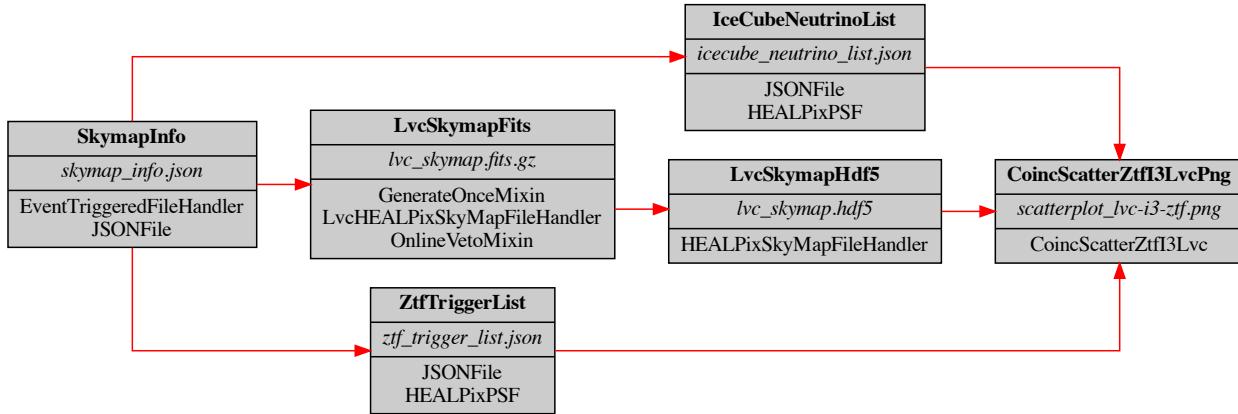


Fig. 5: Required input files for `llama.files.coinc_plots.CoincScatterZtfI3LvcPng` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincScatterZtfI3LvcPng` to be generated.

Human-readable joint ZTF/Gravitational Wave/High Energy Neutrino skymap plot in PNG format.

```

DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>)

FILEEXT = 'png'

FILENAME = 'scatterplot_lvc-i3-ztf.png'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincScatterZtfI3LvcPng'>)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)
    
```

```

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}})}), <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}})}), <class
'llama.files.ztf_trigger_list.ZtfTriggerList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}})})})
Bases: llama.files.coinc_plots.CoincScatterZtfLVC
    
```

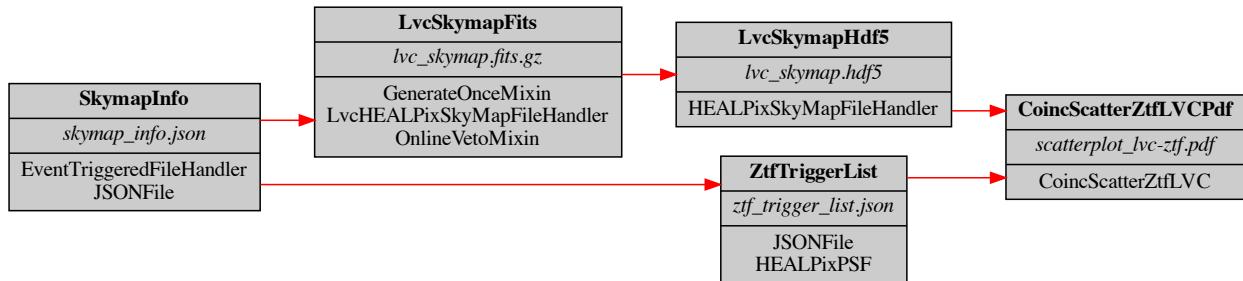


Fig. 6: Required input files for `llama.files.coinc_plots.CoincScatterZtfLVCPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincScatterZtfLVCPdf` to be generated.

Human-readable joint ZTF/Gravitational Wave skymap plot in PDF format.

```

DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>
FILEEXT = 'pdf'
FILENAME = 'scatterplot_lvc-ztf.pdf'
MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincScatterZtfLVCPdf'>,
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>)
UR_DEPENDENCY_TREE = ImmutableDict({<class 'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class 'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class 'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}})}), <class 'llama.files.ztf_trigger_list.ZtfTriggerList': ImmutableDict({<class 'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}})})})})
Bases: llama.files.coinc_plots.CoincScatterZtfLVC
Human-readable joint ZTF/Gravitational Wave skymap plot in PNG format.
DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>
FILEEXT = 'png'
FILENAME = 'scatterplot_lvc-ztf.png'
    
```

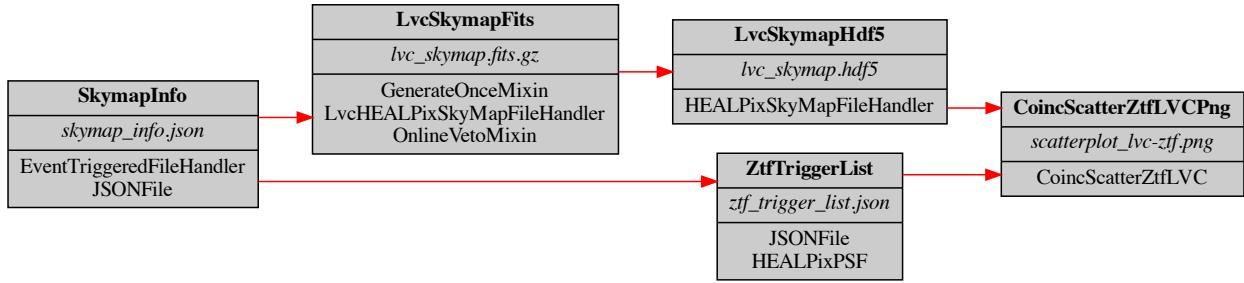


Fig. 7: Required input files for `llama.files.coinc_plots.CoincScatterZtfLVCpng` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincScatterZtfLVCpng` to be generated.

```

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincScatterZtfLVCpng'>)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>:
ImmutableDict({<class 'llama.files.lvc_skymap.LvcSkymapFits'>:
ImmutableDict({<class 'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}),
<class 'llama.files.ztf_trigger_list.ZtfTriggerList'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})})})

class llama.files.CoincSignificanceI3Lvc(eventid_or_fh, rundir=None)
Bases: llama.filehandler.JSONFile
  
```

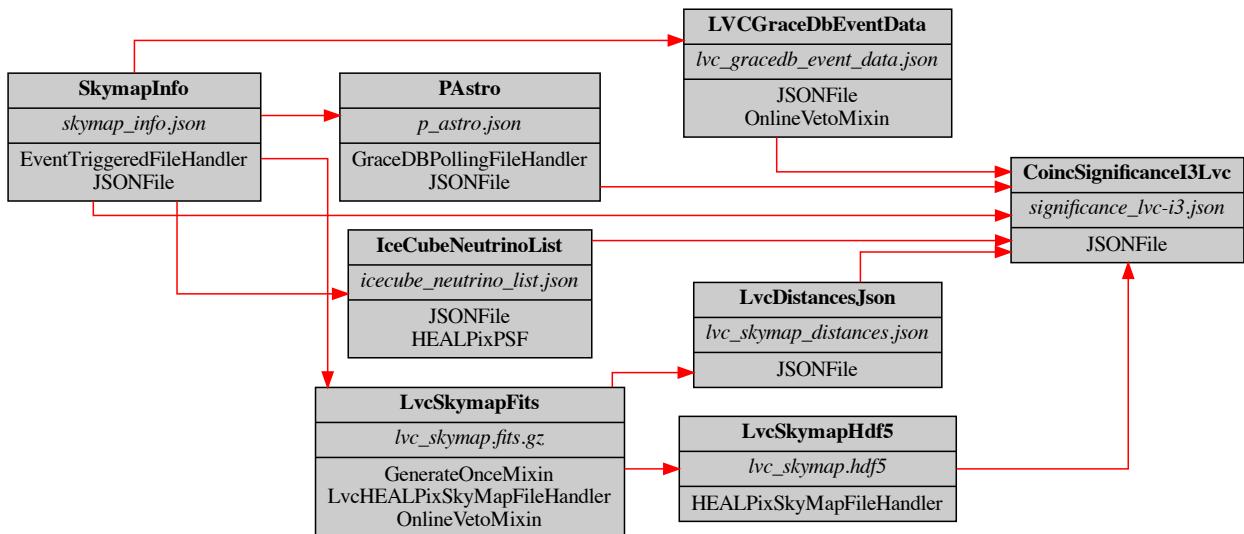


Fig. 8: Required input files for `llama.files.coinc_significance.opa.CoincSignificanceI3Lvc` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_significance.opa.CoincSignificanceI3Lvc` to be generated.

Calculates the significance (Bayes Factor) of a joint GW+HEN event.

```

DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.lvc_skymap.LvcDistancesJson'>, <class 'llama.files.gracedb.PAstro'>)

DETECTORS = (Detector(name='IceCube', abbrev='i3', fullname='IceCube',
url='http://wiki.icecube.wisc.edu', summary='IceCube', description='',
citations=ImmutableDict({})), Detector(name='LVC', abbrev='lvc', fullname='LVC',
url='http://wiki.ligo.org', summary='LVC', description='',
citations=ImmutableDict({})))

FILENAME = 'significance_lvc-i3.json'

MANIFEST_TYPES = (<class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>,)

TIMEOUT = 600

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.gracedb.LVCGraceDbEventData'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}})}), <class
'llama.files.i3.json.IceCubeNeutrinoList'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}})}), <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>: ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}})})}), <class
'llama.files.lvc_skymap.LvcDistancesJson'>: ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}})})}), <class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}}, <class
'llama.files.gracedb.PAstro'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}})}))}

property combined_p_value
    Return the combined p-value for the whole event.

odds()
    Return a list of the odds ratios calculated. Ordering corresponds to the ordering of neutrinos in the
    IceCubeNeutrinoList file used to calculate the odds ratios.

property p_values
    Return a list of the p-values calculated. Ordering corresponds to the ordering of neutrinos in the
    IceCubeNeutrinoList file used to calculate the odds ratios.

class llama.files.CoincSummaryI3LvcPdf(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler

Human-viewable joint Neutrino/Gravitational Wave skymap plot complete with a list of neutrinos and their prop-
erties in PDF form. Useful for quick human checks of the output of the pipeline on an event.

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincSummaryI3LvcTex'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>)

```

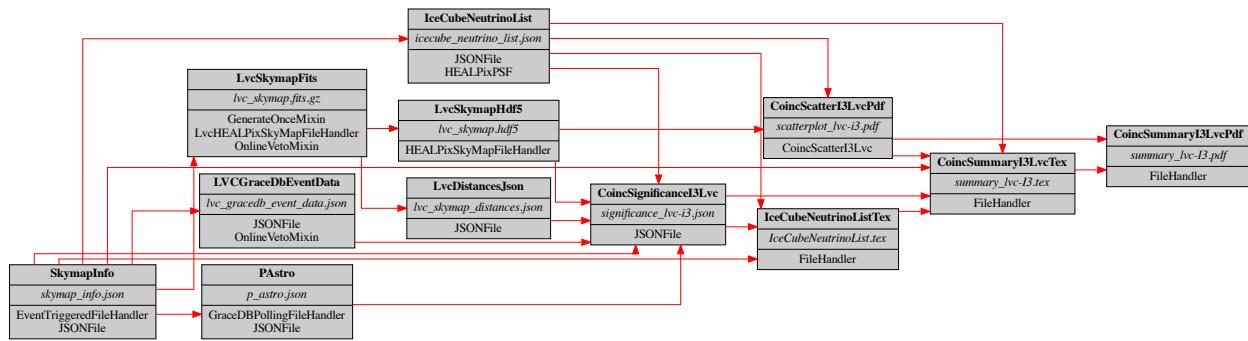


Fig. 9: Required input files for `llama.files.coinc_plots.CoincSummaryI3LvcPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincSummaryI3LvcPdf` to be generated.

```

FILENAME = 'summary_lvc-I3.pdf'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincSummaryI3LvcPdf'>,)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>, <class
'llama.files.i3.tex.IceCubeNeutrinoListTex'>, <class
'llama.files.coinc_plots.CoincSummaryI3LvcTex'>)

```

```
UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.coinc_plots.CoincSummaryI3LvcTex': ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}), <class
'llama.files.i3.tex.IceCubeNeutrinoListTex': ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}), <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc': ImmutableDict({<class
'llama.files.gracedb.LVCGraceDbEventData': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})), <class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})), <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})), <class
'llama.files.lvc_skymap.LvcDistancesJson': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}), <class
'llama.files.gracedb.PAstro': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}))), <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf': ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})), <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}))), <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc': ImmutableDict({<class
'llama.files.gracedb.LVCGraceDbEventData': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})), <class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})), <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}))), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})), <class
'llama.files.lvc_skymap.LvcDistancesJson': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}))), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}), <class
'llama.files.gracedb.PAstro': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}))), <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf': ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})), <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}))), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}))), <class
'llama.files.CoincSummaryI3LvcTex(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler
```

```
class llama.files.CoincSummaryI3LvcTex(eventid_or_fh, rundir=None)  
    Bases: llama.filehandler.FileHandler
```

LaTeX file for a human-viewable joint Neutrino/Gravitational Wave skymap plot complete with a list of neutrinos and their properties. Gets compiled to a PDF file.

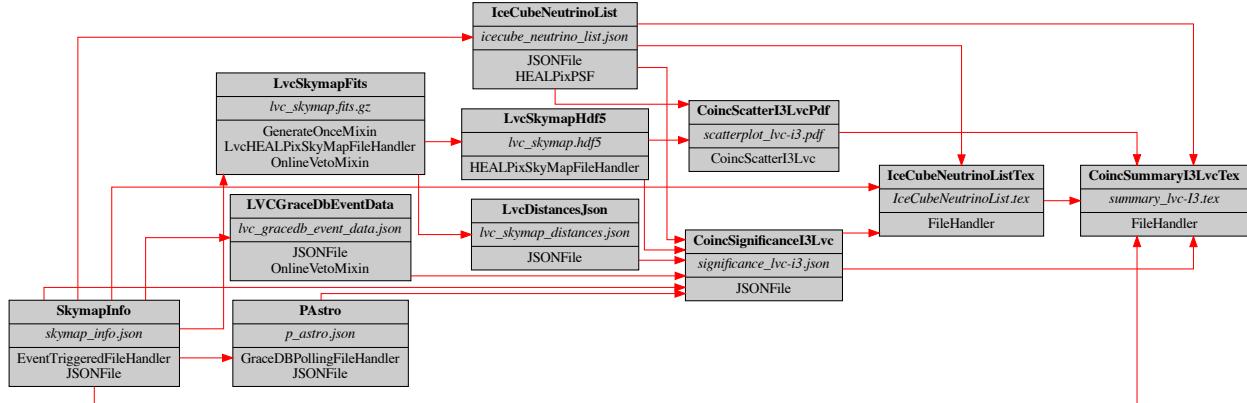


Fig. 10: Required input files for `llama.files.coinc_plots.CoincSummaryI3LvcTex` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincSummaryI3LvcTex` to be generated.

```

DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>, <class
'llama.files.i3.tex.IceCubeNeutrinoListTex'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)

FILENAME = 'summary_lvc-I3.tex'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincSummaryI3LvcTex'>)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>, <class
'llama.files.i3.tex.IceCubeNeutrinoListTex'>)
  
```

```

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.i3.tex.IceCubeNeutrinoListTex': ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc': ImmutableDict({<class
'llama.files.gracedb.LVCGraceDbEventData': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}), <class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}), <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.lvc_skymap.LvcDistancesJson': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.gracedb.PAstro': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf': ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc': ImmutableDict({<class
'llama.files.gracedb.LVCGraceDbEventData': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.lvc_skymap.LvcDistancesJson': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.gracedb.PAstro': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class

```

class `llama.files.FermiGRBsJSON(eventid_or_fh, rundir=None)`

Bases: `llama.filehandler.JSONFile`, `llama.files.healpix.skymap.HEALPixPSF`

New and historical Fermi GRB triggers saved in a JSON file. Triggers are in a list of JSON objects, where each trigger will have (at least) the following properties:

ra [float] Right ascension in degrees.

dec [float] Declination in degrees.

sigma [float] Angular uncertainty in degrees.

gps_time [float] GPS time of the trigger.

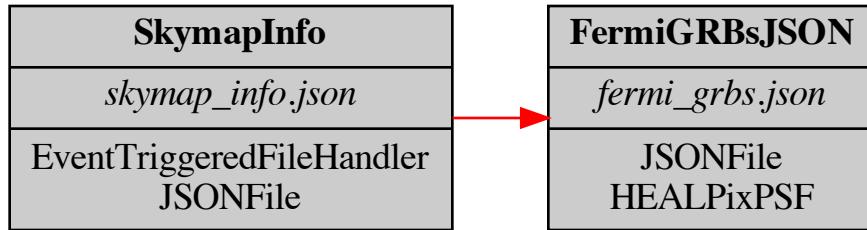


Fig. 11: Required input files for `llama.files.fermi_grb.FermiGRBsJSON` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.fermi_grb.FermiGRBsJSON` to be generated.

Additional properties might include:

`graceid` [str] GraceID of the event, if available.

`skymap_url` [str] URL of a FITS skymap, if available.

`DEPENDENCIES` = (<class '`llama.files.skymap_info.SkymapInfo`'>,)

`DETECTORS` = (`Detector(name='Fermi', abbrev='frm', fullname='Fermi', url=None, summary='Fermi', description='', citations=ImmutableDict({}))`,)

`FILENAME` = '`fermi_grbs.json`'

`MANIFEST_TYPES` = (<class '`llama.files.fermi_grb.FermiGRBsJSON`'>,)

`UR_DEPENDENCIES` = (<class '`llama.files.skymap_info.SkymapInfo`'>,)

`UR_DEPENDENCY_TREE` = `frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})`

property num_triggers

The number of triggers described by this file. Useful mostly for quickly determining if this trigger list is empty.

source_locations()

Returns a list of tuples of (RA, Dec, sigma) for all point-like sources, where (RA, Dec) is the central sky position and sigma is the standard deviation of the Gaussian PSF. Since this depends on the structure of data in the relevant file handler, it must be specified for each subclass.

property template_skymap_filehandler

The HEALPixSkyMapFileHandler whose HEALPix parameters should be used as a template for the HEALPixSkyMap output by this FileHandler. It is assumed that this HEALPixSkyMapFileHandler only returns one skymap in its list; consequently, only the first skymap loaded by this file handler will be used. Note that the template skymap file handler is likely not a dependency of this file handler; it is only used as a default for formatting generated skymaps from this file handler's data.

class `llama.files.IceCubeNeutrinoList`(`eventid_or_fh`, `rundir=None`)

Bases: `llama.filehandler.JSONFile`, `llama.files.healpix.skymap.HEALPixPSF`

Takes a list of dictionaries describing neutrinos and saves it to a text file. Validates the neutrino data to make sure each neutrino contains (minimally) the following properties:

- mjd
- zenith
- azimuth
- sigma

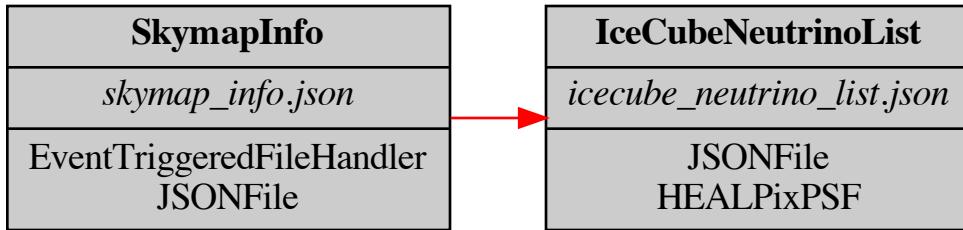


Fig. 12: Required input files for `llama.files.i3.json.IceCubeNeutrinoList` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.json.IceCubeNeutrinoList` to be generated.

- energy
- type (one the values found in `NEUTRINO_TYPES`)

If this file is generated within the temporal search window for neutrinos, it is possible not all neutrinos needed for the final result will be available. In this case, the file will automatically become `obsolete` and will be regenerated after the window has elapsed.

Neutrinos from IceCube can be delayed due to communications issues; to combat this, when neutrinos are pulled from the online data stream after the temporal search window has elapsed, a check will be run to see if the next neutrino after the end of the search window is yet available. If it is not available, it is assumed that not all neutrinos needed for the search have arrived from south pole, and a `GenerationError` will be raised, allowing the pipeline to cooldown and retry the neutrino pull later.

Neutrinos pulled from before `MJD_TOPIC_SWITCH_17_TO_BLANK` will be automatically pulled from local GFU archives (stored online on LLAMA S3 and cached locally after use in `llama.utils.CACHEDIR` in `*.npy` format). Neutrinos from after this date will be pulled from the online GFU stream.

```

DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
DETECTORS = (Detector(name='IceCube', abbrev='i3', fullname='IceCube',
url='http://wiki.icecube.wisc.edu', summary='IceCube', description='',
citations=ImmutableDict({})),)
FILENAME = 'icecube_neutrino_list.json'
MANIFEST_TYPES = (<class 'llama.files.i3.json.IceCubeNeutrinoList'>,)
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})
downselect(ra=None, dec=None, ntype=None)

Downselect the neutrino list to match given criteria. Save the old neutrino list with the original filename
prefixed by a timestamp and replace it with the downselected list.

```

Parameters

- `ra` (`list`) – Right-ascension intervals (in degrees) that the neutrinos must fall into, e.g. `[(21, 41), (92, 102)]`
- `dec` (`list`) – Declination intervals (in degrees) that the neutrinos must fall into, e.g. `[(-3, 20)]`
- `ntype` (`str`) – A type that the neutrino must match, e.g. `'observation'` for real neutrinos.

end_of_neutrino_window()

The GPS time at which we can assume that all neutrinos are in.

generated_before_all_neutrinos()

Whether this file was generated before all IceCube neutrinos in the time window were available. Returns False if the generation time cannot be determined.

is_obsolete(*checked=None*, *kwargs*)**

IceCubeNeutrinoList files can be fetched immediately after a GW event becomes available. However, we want to eventually run our analysis on all neutrinos in a certain time window around an event. We therefore want to re-fetch the neutrinos in that time window after the time window has passed (allowing a safety factor for the IceCube GFU API to process new neutrino triggers). To do this, we mark the IceCubeNeutrinoList as obsolete if the current time is after the time window + safety margin has elapsed and the neutrino list was generated before that time had passed (meaning there might have been new neutrinos saved that are useful for the analysis after the file was originally generated). Standard FileHandler obsolescence criteria are also used to determine obsolescence through a call to super.

property neutrinos

Return a list containing a Neutrino object for each neutrino; more convenient to work with when writing formulae than dealing directly with the neutrino dictionaries.

property num_triggers

The number of triggers described by this file. Useful mostly for quickly determining if this trigger list is empty.

source_locations()

Returns a list of tuples of (RA, Dec, sigma) for all point-like sources, where (RA, Dec) is the central sky position and sigma is the standard deviation of the Gaussian PSF. Since this depends on the structure of data in the relevant file handler, it must be specified for each subclass.

property template_skymap_filehandler

The HEALPixSkyMapFileHandler whose HEALPix parameters should be used as a template for the HEALPixSkyMap output by this FileHandler. It is assumed that this HEALPixSkyMapFileHandler only returns one skymap in its list; consequently, only the first skymap loaded by this file handler will be used. Note that the template skymap file handler is likely not a dependency of this file handler; it is only used as a default for formatting generated skymaps from this file handler's data.

class llama.files.IceCubeNeutrinoListCoincTxt(*eventid_or_fh*, *rundir=None*)

Bases: [llama.filehandler.FileHandler](#)

A space-delimited, table-formatted list of neutrinos meant for human consumption in GCN Circulars. Contains coincidence significance measures in addition to the base neutrino properties.

DEPENDENCIES = (<class 'llama.files.i3.json.IceCubeNeutrinoList'>, <class 'llama.files.skymap_info.SkymapInfo'>, <class 'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)

FILENAME = 'icecube_neutrino_list_coinc.txt'

MANIFEST_TYPES = (<class 'llama.files.i3.txt.IceCubeNeutrinoListCoincTxt'>,)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class 'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>, <class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class 'llama.files.i3.json.IceCubeNeutrinoList'>, <class 'llama.files.gracedb.LVCGraceDbEventData'>, <class 'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)

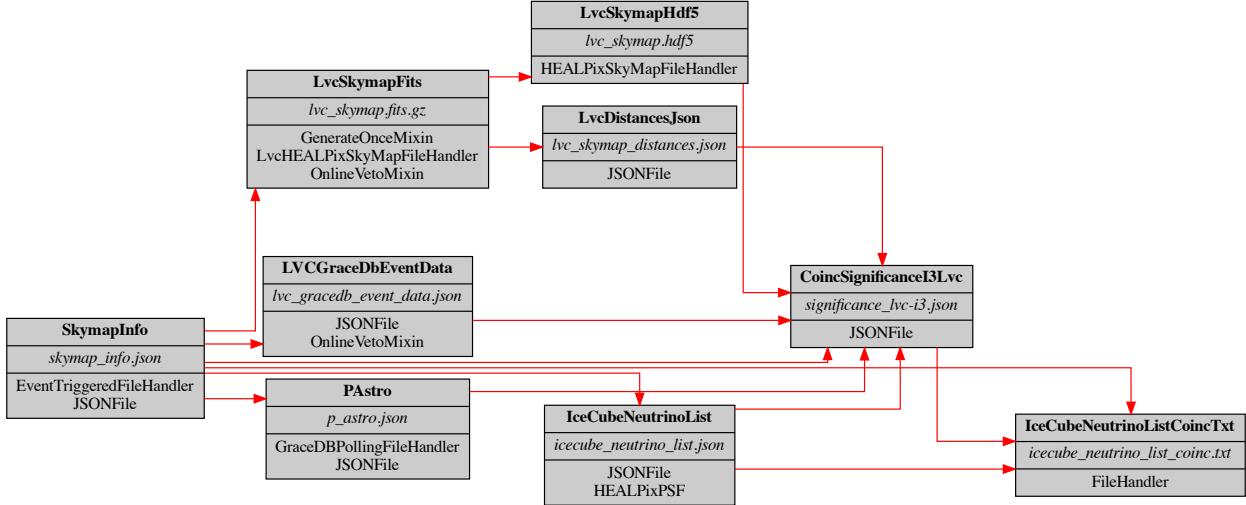


Fig. 13: Required input files for `llama.files.i3.txt.IceCubeNeutrinoListCoincTxt` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.txt.IceCubeNeutrinoListCoincTxt` to be generated.

```

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'	llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'	llama.files.skymap_info.SkymapInfo': ImmutableDict({}), <class
'	llama.files.coinc_significance.opa.CoincSignificanceI3Lvc': ImmutableDict({<class
'	llama.files.gracedb.LVCGraceDbEventData': ImmutableDict({<class
'	llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'	llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'	llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'	llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'	llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'	llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'	llama.files.lvc_skymap.LvcDistancesJson': ImmutableDict({<class
'	llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'	llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'	llama.files.gracedb.PAstro': ImmutableDict({<class
'	llama.files.skymap_info.SkymapInfo': ImmutableDict({})}))})
class llama.files.IceCubeNeutrinoListTex(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler

A list of neutrinos and their reconstructed properties as well as the significance of the joint event) in LaTeX format suitable for use in papers and coincident skymap summary plots.

DEPENDENCIES = (<class 'llama.files.i3.json.IceCubeNeutrinoList', <class
'	llama.files.skymap_info.SkymapInfo', <class
'	llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)

FILENAME = 'IceCubeNeutrinoList.tex'

MANIFEST_TYPES = (<class 'llama.files.i3.tex.IceCubeNeutrinoListTex'>,)

```

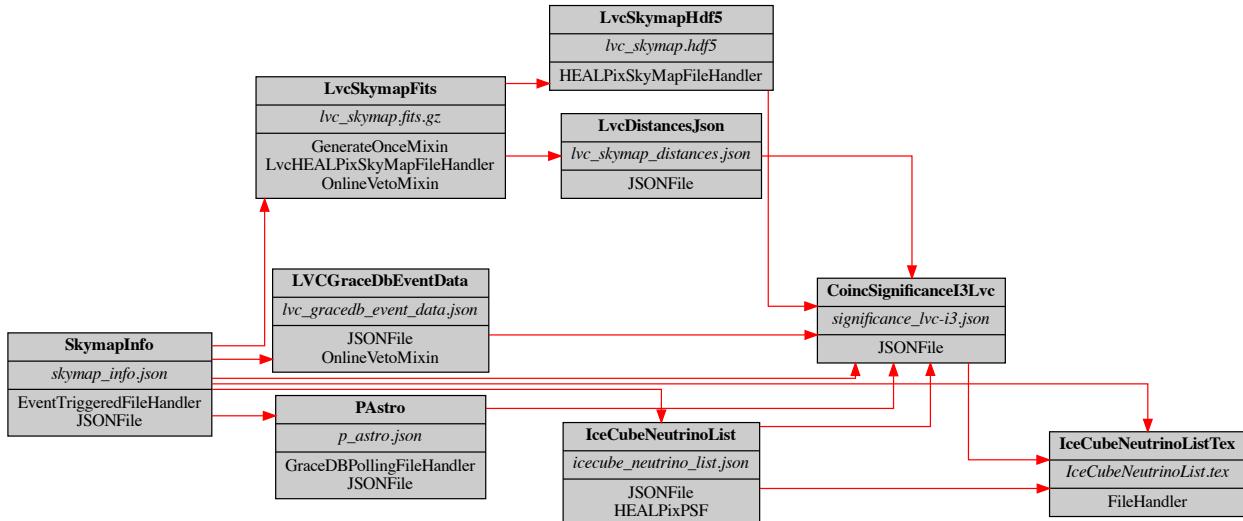


Fig. 14: Required input files for `llama.files.i3.tex.IceCubeNeutrinoListTex` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.tex.IceCubeNeutrinoListTex` to be generated.

```
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class  
'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,  
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class  
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class  
'llama.files.i3.json.IceCubeNeutrinoList'>, <class  
'llama.files.gracedb.LVCGraceDbEventData'>, <class  
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)  
  
UR_DEPENDENCY_TREE = ImmutableDict({<class  
'llama.files.i3.json.IceCubeNeutrinoList'>: ImmutableDict({<class  
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}), <class  
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({}), <class  
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>: ImmutableDict({<class  
'llama.files.gracedb.LVCGraceDbEventData'>: ImmutableDict({<class  
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}), <class  
'llama.files.i3.json.IceCubeNeutrinoList'>: ImmutableDict({<class  
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}), <class  
'llama.files.lvc_skymap.LvcSkymapHdf5'>: ImmutableDict({<class  
'llama.files.lvc_skymap.LvcSkymapFits'>: ImmutableDict({<class  
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})})}), <class  
'llama.files.lvc_skymap.LvcDistancesJson'>: ImmutableDict({<class  
'llama.files.lvc_skymap.LvcSkymapFits'>: ImmutableDict({<class  
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})})}), <class  
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({}), <class  
'llama.files.gracedb.PAstro'>: ImmutableDict({<class  
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})})})  
  
class llama.files.IceCubeNeutrinoListTxt(eventid_or_fh, rundir=None)  
Bases: llama.filehandler.FileHandler
```

A space-delimited, table-formatted list of neutrinos formatted for human consumption in GCN Circulars.

```
DEPENDENCIES = (<class 'llama.files.i3.json.IceCubeNeutrinoList'>, <class 'llama.files.skymap_info.SkymapInfo'>)
```

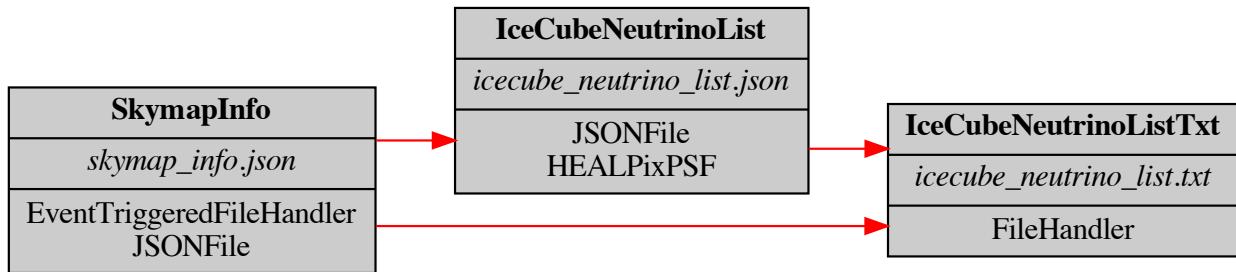


Fig. 15: Required input files for `llama.files.i3.txt.IceCubeNeutrinoListTxt` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.txt.IceCubeNeutrinoListTxt` to be generated.

```

FILENAME = 'iccube_neutrino_list.txt'
MANIFEST_TYPES = (<class 'llama.files.i3.txt.IceCubeNeutrinoListTxt'>,)
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)
UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'	llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'	llama.files.skymap_info.SkymapInfo': ImmutableDict({})}>})
class llama.files.LVAlertAdvok(eventid_or_fh, rundir=None)
Bases: llama.filehandler.EventTriggeredFileHandler, llama.filehandler.JSONFile

```



Fig. 16: `llama.files.lvalert_advok.LVAlertAdvok` is created from external triggers. It therefore has no LLAMA-representable input dependencies, but instead acts as initial input for other FileHandler classes.

If the event was tagged ADVOK via an LVAalert, this file will contain the LVAalert JSON data.

```

FILENAME = 'lvalert_advok.json'
MANIFEST_TYPES = (<class 'llama.files.lvalert_advok.LVAlertAdvok'>,)
UR_DEPENDENCIES = ()
UR_DEPENDENCY_TREE = frozenset({})
class llama.files.LVAlertJSON(eventid_or_fh, rundir=None)
Bases: llama.filehandler.EventTriggeredFileHandler, llama.filehandler.JSONFile

```

An LVAalert JSON object containing skymap info.

```

FILENAME = 'lvalert.json'
MANIFEST_TYPES = (<class 'llama.files.lvalert_json.LVAlertJSON'>,)

```



Fig. 17: `llama.files.lvalert_json.LVAAlertJSON` is created from external triggers. It therefore has no LLAMA-representable input dependencies, but instead acts as initial input for other `FileHandler` classes.

```
UR_DEPENDENCIES = ()
UR_DEPENDENCY_TREE = frozenset({})

property alert_type
    Get the alert_type of this LVAAlert.

property base_filename
    Get the unversioned filename for this file. This only works for LVAAlerts corresponding to uploads.

        Returns base_filename – The skymap filename as a SkymapFilename instance.

        Return type files.lvc_skymap.utils.SkymapFilename

        Raises KeyError – If this LVAAlert does not contain a filename.

property far
    The False Alarm Rate of this trigger.

property graceid
    The GraceID, a unique ID used on GraceDB to register gravitational wave triggers.

property notice_time_str
    Get the time at which the LVAAlert was created.

property skymap_filename
    Get the exact filename, including version information, for this file. If this LVAAlert does not correspond to a skymap upload, try to pull down the most recently generated skymap.

        Returns filename – The full canonical filename as a SkymapFilename instance.

        Return type files.lvc_skymap.utils.SkymapFilename

property skymap_version
    Get the version of the skymap with this particular filename. This only works for LVAAlerts corresponding to uploads.

        Returns version – File version corresponding to this upload.

        Return type int

class llama.files.LVCGraceDbEventData(eventid_or_fh, rundir=None)
Bases: llama.filehandler.JSONFile, llama.filehandler.mixins.OnlineVetoMixin

The event dictionary as returned by the GraceDb API. Includes richer information on the single-detector and multi-detector inspiral parameters than is provided in LVAAlerts or GCN Notices. Convenience methods are provided for certain commonly-used reconstructed parameters, but full data can always be accessed by running read_json.

DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
```

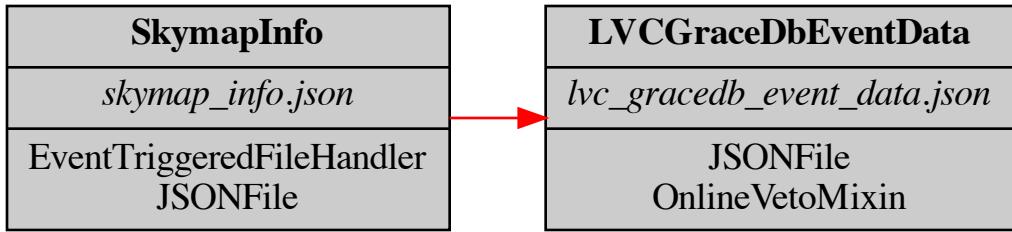


Fig. 18: Required input files for `llama.files.gracedb.LVCGraceDbEventData` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.gracedb.LVCGraceDbEventData` to be generated.

```

FILENAME = 'lvc_gracedb_event_data.json'
GW_DETECTORS = {'H1', 'L1', 'V1'}
MANIFEST_TYPES = (<class 'llama.files.gracedb.LVCGraceDbEventData'>,)
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})

property chirp_mass
    The chirp mass. Raises a KeyError if it is not defined.

property instruments
    A sorted list of the interferometers that participated in creating this trigger, e.g. ['H1', 'L1']. Raises a ValueError if unexpected instruments are found. Raises a KeyError if it is not defined.

property offline
    Whether this trigger was generated offline.

property snr
    The coincident signal-to-noise ratio (SNR). Raises a KeyError if it is not defined.

property total_mass
    The total binary mass. Raises a KeyError if it is not defined.

class llama.files.LvcDistancesJson(eventid_or_fh, rundir=None)
Bases: llama.filehandler.JSONFile

```

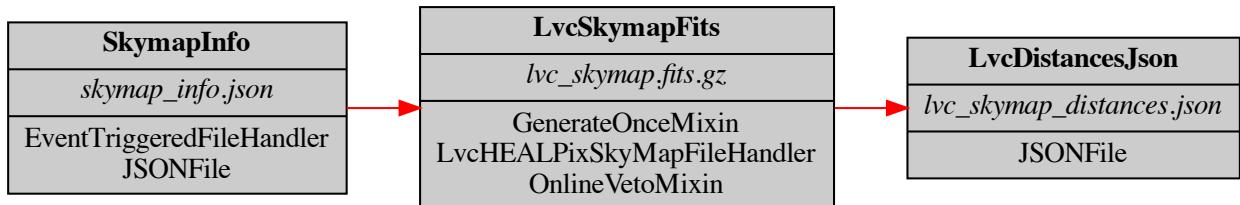


Fig. 19: Required input files for `llama.files.lvc_skymap.LvcDistancesJson` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.lvc_skymap.LvcDistancesJson` to be generated.

A simple JSON file with the reconstructed distances to an event. Contains the mean reconstructed distance as `distmean` and the standard deviation as `diststd`.

```

DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapFits'>,)
FILENAME = 'lvc_skymap_distances.json'

```

```
MANIFEST_TYPES = (<class 'llama.files.lvc_skymap.LvcDistancesJson'>,)
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvc_skymap.LvcSkymapFits'>})
```

property distmean

The mean distance to the event; a probability-weighted average of reconstructed distances. Extracted from the original LVC skymap.

property diststd

The probability-weighted standard deviation of the reconstructed distance to the event. Extracted from the original LVC skymap.

```
class llama.files.LvcGcnXml(eventid_or_fh, rundir=None)
Bases: llama.filehandler.EventTriggeredFileHandler
```

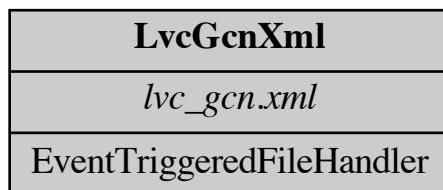


Fig. 20: `llama.files.lvc_gcn_xml.LvcGcnXml` is created from external triggers. It therefore has no LLAMA-representable input dependencies, but instead acts as initial input for other `FileHandler` classes.

A VOEvent XML file received from GCN corresponding to an LVC event notice.

```
FILENAME = 'lvc_gcn.xml'
MANIFEST_TYPES = (<class 'llama.files.lvc_gcn_xml.LvcGcnXml'>,)
UR_DEPENDENCIES = ()
UR_DEPENDENCY_TREE = frozenset({})

property alert_type
    Get the alert type for this VOEvent.

property etree
    Return an lxml.etree for this VOEvent.

property event_time_gps
    Get the time of the observed event up to nanosecond precision (less accurate than this in practice) in GPS time format.

property event_time_gps_nanoseconds
    Get the number of nanoseconds past the last GPS second at the time of the observed event. Ostensibly provides nanosecond precision, but is less accurate than this in practice.

property event_time_gps_seconds
    Get the time of the observed event in GPS seconds (truncated to the nearest second).

property event_time_mjd
    Get the time of the observed event in UTC MJD format.

property event_time_str
    Get a unicode string with the ISO date and time of the observed event straight from the VOEvent file. UTC time.
```

```

property far
    Get the false alarm rate of this VOEvent as a float value.

get_alert_type()
    Read the alert type of this GCN Notice.

get_param(param)
    Get this VOEventParam for this event.

property graceid
    Get the GraceID corresponding to this VOEvent.

property ivorn
    Get the IVORN, a unique identifier for this GCN Notice.

property notice_time_str
    Get a unicode string with the date and time of creation of the notification associated with this VOEvent, rather than the time of detection of the event itself. Should be in ISO format.

property pipeline
    Return an ALL-CAPS name of the pipeline used to generate this event, e.g. GSTLAL or CWB.

property role
    Get the role of this VOEvent as specified in the header.

property skymap_filename
    Get the filename for this skymap as it appears on GraceDB.

class llama.files.LvcRetractionXml(eventid_or_fh, rundir=None)
    Bases: llama.files.lvc\_gcn\_xml.LvcGcnXml

```

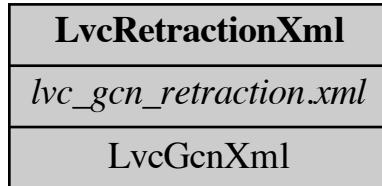


Fig. 21: `llama.files.lvc_gcn_xml.LvcRetractionXml` is created from external triggers. It therefore has no LLAMA-representable input dependencies, but instead acts as initial input for other `FileHandler` classes.

A VOEvent XML file retracting a GCN notice.

```

FILENAME = 'lvc_gcn_retraction.xml'

class llama.files.LvcSkymapFits(eventid_or_fh, rundir=None)
    Bases: llama.filehandler.GenerateOnceMixin, llama.files.healpix.LvcHEALPixSkyMapFileHandler, llama.filehandler.mixins.OnlineVetoMixin

```

The skymap suggested by the LVC Initial GCN Notice.

```

DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
DETECTORS = (Detector(name='LVC', abbrev='lvc', fullname='LVC', url='http://wiki.ligo.org', summary='LVC', description='', citations=ImmutableDict({})),)
FILENAME = 'lvc_skymap.fits.gz'
MANIFEST_TYPES = (<class 'llama.files.lvc_skymap.LvcSkymapFits'>,)
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)

```

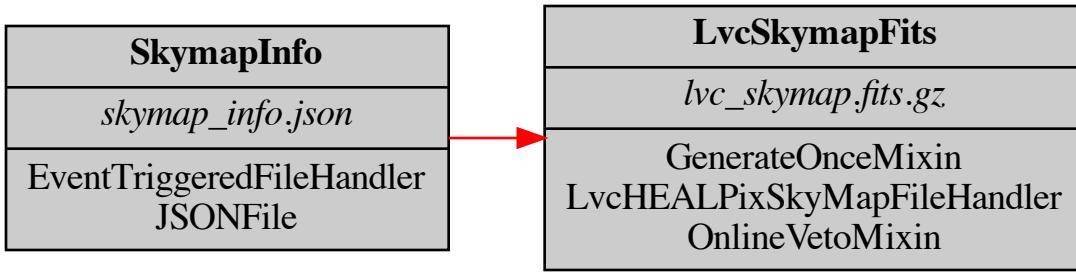


Fig. 22: Required input files for `llama.files.lvc_skymap.LvcSkymapFits` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.lvc_skymap.LvcSkymapFits` to be generated.

```
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})

class llama.files.LvcSkymapHdf5(eventid_or_fh, rundir=None)
    Bases: llama.files.healpix.HEALPixSkyMapFileHandler
```

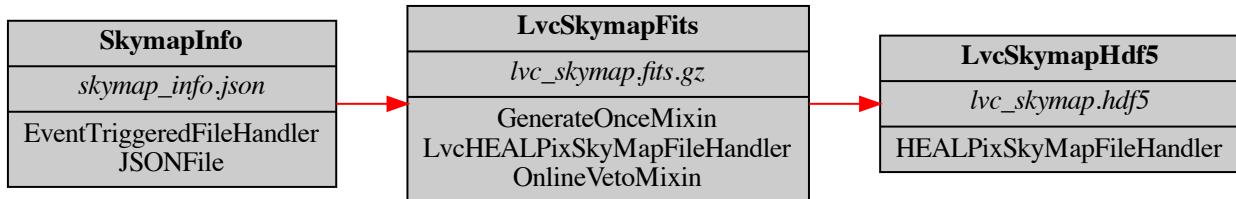


Fig. 23: Required input files for `llama.files.lvc_skymap.LvcSkymapHdf5` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.lvc_skymap.LvcSkymapHdf5` to be generated.

An HDF5-formatted copy of the initial LVC skymap. Loads more quickly.

```
DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapFits'>,)

DETECTORS = (Detector(name='LVC', abbrev='lvc', fullname='LVC',
url='http://wiki.ligo.org', summary='LVC', description='',
citations=ImmutableDict({})),)

FILENAME = 'lvc_skymap.hdf5'

MANIFEST_TYPES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>,)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>)

UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvc_skymap.LvcSkymapFits'>})

class llama.files.PAstro(eventid_or_fh, rundir=None)
    Bases: llama.files.gracedb.GraceDBPollingFileHandler, llama.filehandler.JSONFile

A classification of an event's possible sources by probability.

FILENAME = 'p_astro.json'

MANIFEST_TYPES = (<class 'llama.files.gracedb.PAstro'>,)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)

UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})
```

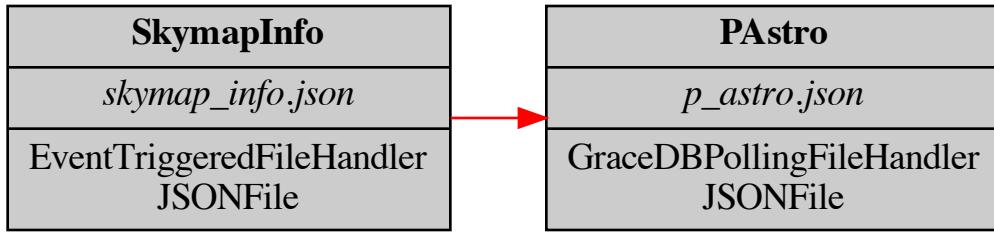


Fig. 24: Required input files for `llama.files.gracedb.PAstro` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.gracedb.PAstro` to be generated.

property most_likely_population

Get the population that this event most likely belongs too by seeing whether it is more likely than not to contain NS matter. More precisely, compares `p_bbh` to `p_bns + p_nsbh + p_mass_gap`, returning '`bbh`' if the former is larger and '`bns`' if the latter is larger. `p_terr` is ignored in this calculation.

property p_bbh

The probability that this was a BBH (binary black hole) merger.

property p_bns

The probability that this was a BNS (binary neutron star) merger.

property p_mass_gap

The probability that at least one of the compact objects in the merger was in the ambiguous mass gap between definite BH and definite NS.

property p_nsbh

The probability that this was a NSBH (neutron star/black hole) merger.

property p_terr

The probability that this event was terrestrial noise.

property remote_filename

The name of the required data in a format recognizable by the remote resource's API. This is quite possibly dynamically generated, and so it must be considered separate from the file handler's name for this file.

```

class llama.files.RctSlkI3CoincSummaryI3LvcPdf(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptIcecube
DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincSummaryI3LvcPdf'>, <class
'	llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_i3_summary_lvc-I3.pdf.json'
FILENAME_FMT = 'rct_slk_i3_{}.json'
MANIFEST_TYPES = (<class 'llama.files.coinc_plots.RctSlkI3CoincSummaryI3LvcPdf'>,)

UPLOAD
alias of llama.files.coinc_plots.CoincSummaryI3LvcPdf

```

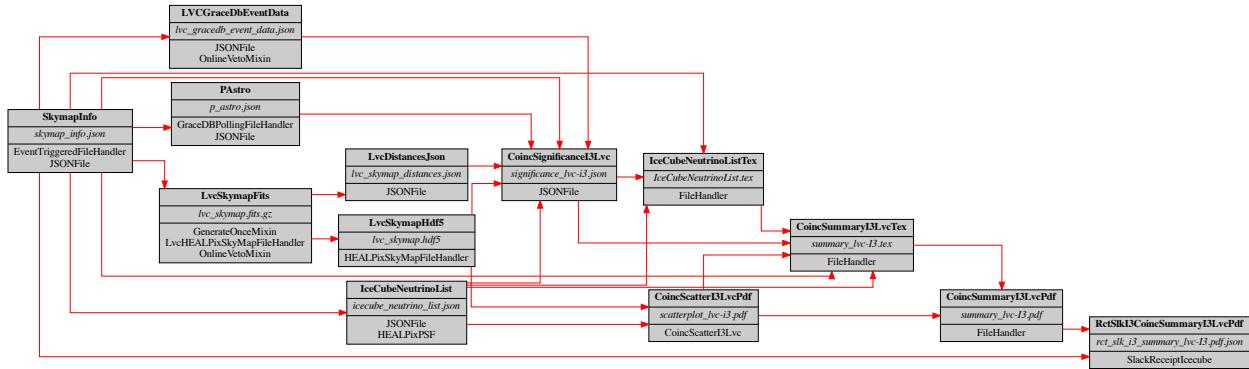


Fig. 25: Required input files for `llama.files.coinc_plots.RctSlkI3CoincSummaryI3LvcPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkI3CoincSummaryI3LvcPdf` to be generated.

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>, <class
'llama.files.coinc_plots.CoinScatterI3LvcPdf'>, <class
'llama.files.i3.tex.IceCubeNeutrinoListTex'>, <class
'llama.files.coinc_plots.CoincSummaryI3LvcTex'>, <class
'llama.files.coinc_plots.CoincSummaryI3LvcPdf'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincSummaryI3LvcPdf'>})

class_vetoes = ((<function icecube_upload_flag_false>, None),)

class llama.files.RctSlkLmaCoincScatterI3LvcPdf(eventid_or_fh, rundir=None)
    Bases: llama.files.slack.SlackReceiptLlama
    
```

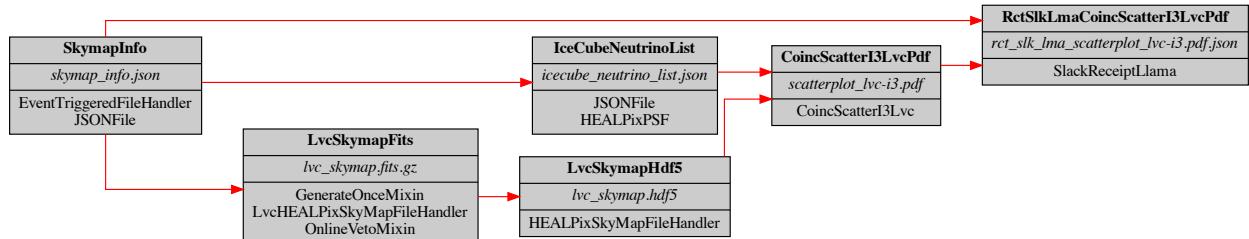


Fig. 26: Required input files for `llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPdf` to be generated.

```

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincScatterI3LvcPdf'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_scatterplot_lvc-i3.pdf.json'
    
```

```

FILENAME_FMT = 'rct_slk_lma_{}.json'
MANIFEST_TYPES = (<class 'llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPdf'>,)
UPLOAD
alias of llama.files.coinc\_plots.CoincScatterI3LvcPdf

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>})

class_vetoies = ()

class llama.files.RctSlkLmaCoincScatterI3LvcPng(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

```

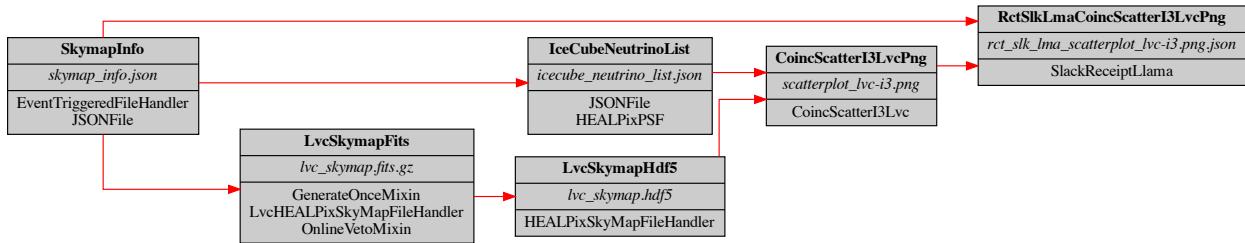


Fig. 27: Required input files for `llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPng` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPng` to be generated.

```

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincScatterI3LvcPng'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_scatterplot_lvc-i3.png'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPng'>,)

UPLOAD
alias of llama.files.coinc\_plots.CoincScatterI3LvcPng

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPng'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincScatterI3LvcPng'>})

class_vetoies = ()

class llama.files.RctSlkLmaCoincScatterZtfI3LvcPdf(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

```

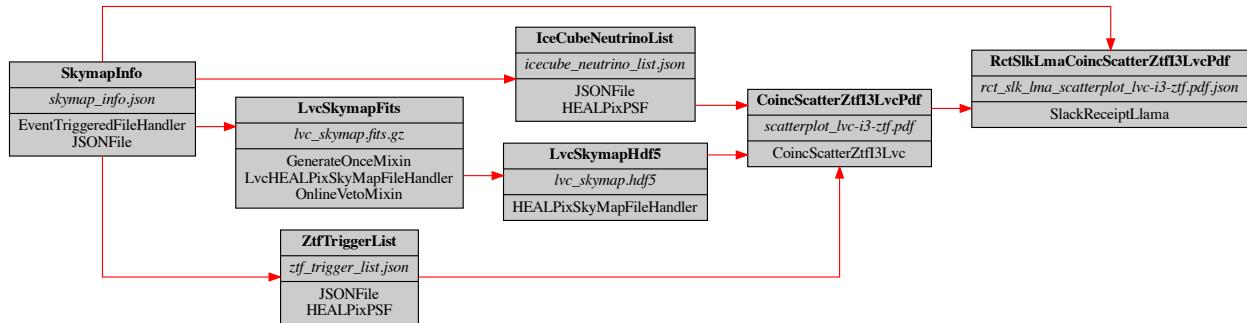


Fig. 28: Required input files for `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPdf` to be generated.

```

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincScatterZtfI3LvcPdf'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_scatterplot_lvc-i3-ztf.pdf.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class
'llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPdf'>,)

UPLOAD
    alias of llama.files.coinc\_plots.CoincScatterZtfI3LvcPdf

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_plots.CoincScatterZtfI3LvcPdf'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincScatterZtfI3LvcPdf'>)})

class_vetoies = ()

class llama.files.RctSlkLmaCoincScatterZtfI3LvcPng(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincScatterZtfI3LvcPng'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_scatterplot_lvc-i3-ztf.png.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class
'llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng'>,)

UPLOAD
    alias of llama.files.coinc\_plots.CoincScatterZtfI3LvcPng

```

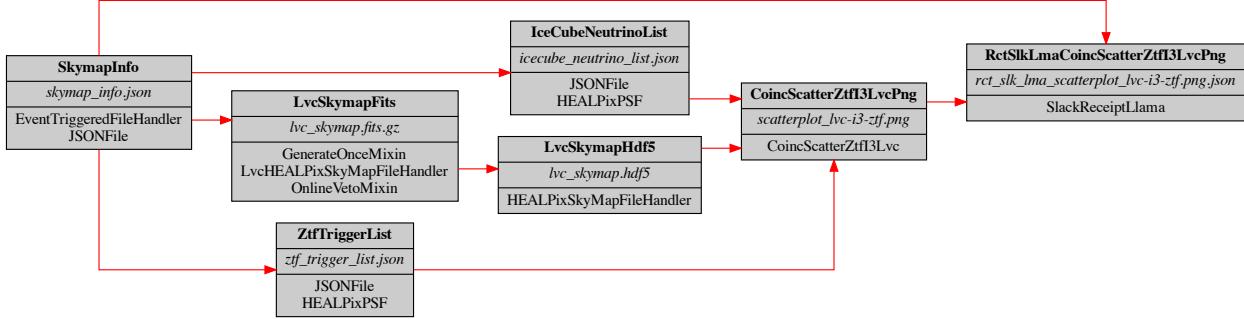


Fig. 29: Required input files for `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng` to be generated.

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_plots.CoincScatterZtfI3LvcPng'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincScatterZtfI3LvcPng'>)})

class_vetoes = ()

class llama.files.RctSlkLmaCoincScatterZtfLVCPdf(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama
    
```

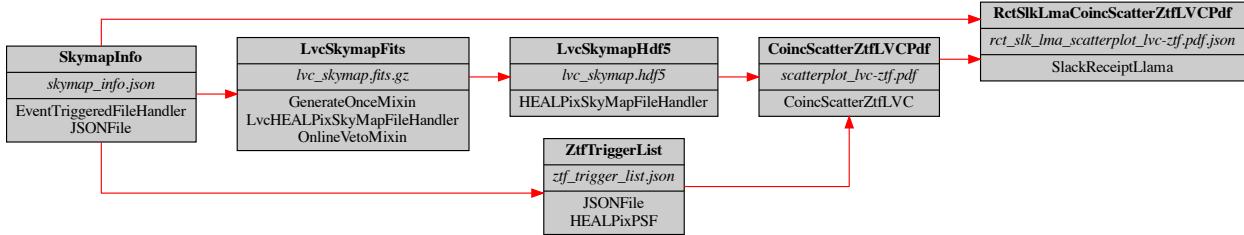


Fig. 30: Required input files for `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPdf` to be generated.

```

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincScatterZtfLVCPdf'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_scatterplot_lvc-ztf.pdf.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPdf'>,)

UPLOAD
alias of llama.files.coinc_plots.CoincScatterZtfLVCPdf
    
```

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.coinc_plots.CoincScatterZtfLVCPdf'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincScatterZtfLVCPdf'>)})

class_vetoies = ()

class llama.files.RctSlkLmaCoincScatterZtfLVCPng(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

```

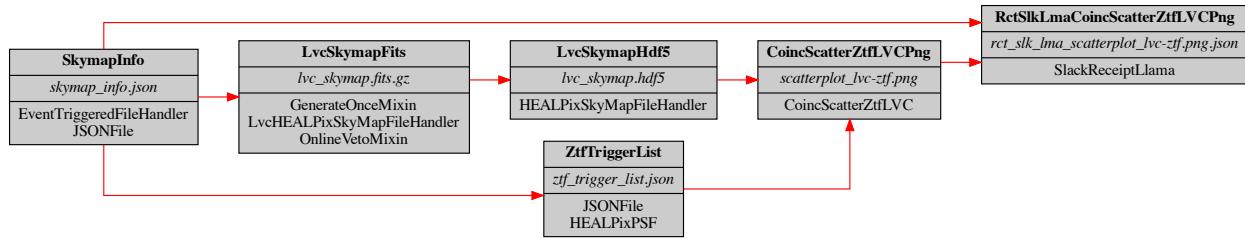


Fig. 31: Required input files for `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPng` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPng` to be generated.

```

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincScatterZtfLVCPng'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_scatterplot_lvc-ztf.png.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPng'>,) 

UPLOAD
    alias of llama.files.coinc\_plots.CoincScatterZtfLVCPng

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.coinc_plots.CoincScatterZtfLVCPng'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincScatterZtfLVCPng'>)})

class_vetoies = ()

class llama.files.RctSlkLmaCoincSignificanceI3Lvc(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

DEPENDENCIES = (<class 'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>,
<class 'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_significance_lvc-i3.json.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

```

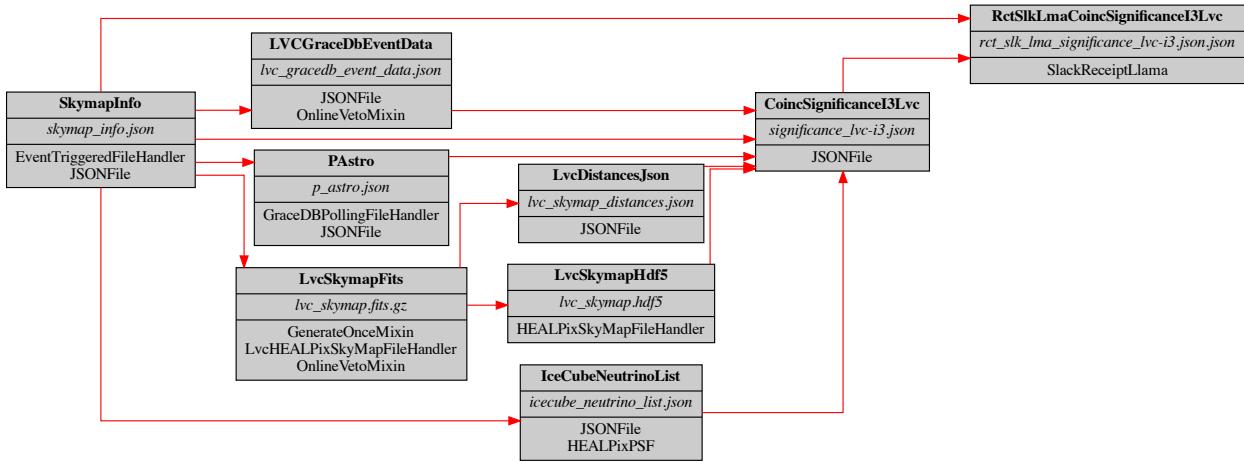


Fig. 32: Required input files for `llama.files.coinc_significance.opa.RctSlkLmaCoincSignificanceI3Lvc` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_significance.opa.RctSlkLmaCoincSignificanceI3Lvc` to be generated.

```

MANIFEST_TYPES = (<class
    'llama.files.coinc_significance.opa.RctSlkLmaCoincSignificanceI3Lvc'>,)

UPLOAD
    alias of llama.files.coinc_significance.opa.CoincSignificanceI3Lvc

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
    'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
    <class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
    'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
    'llama.files.i3.json.IceCubeNeutrinoList'>, <class
    'llama.files.gracedb.LVCGraceDbEventData'>, <class
    'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)

UR_DEPENDENCY_TREE = frozenset({<class
    'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)})

class_vetoies = ()

class llama.files.RctSlkLmaCoincSummaryI3LvcPdf(eventid_or_fh, rundir=None)
    Bases: llama.files.slack.SlackReceiptLlama
    DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincSummaryI3LvcPdf'>, <class
        'llama.files.skymap_info.SkymapInfo'>)

    FILENAME = 'rct_slk_lma_summary_lvc-I3.pdf.json'
    FILENAME_FMT = 'rct_slk_lma_{}.json'

    MANIFEST_TYPES = (<class 'llama.files.coinc_plots.RctSlkLmaCoincSummaryI3LvcPdf'>,)

    UPLOAD
        alias of llama.files.coinc_plots.CoincSummaryI3LvcPdf

```

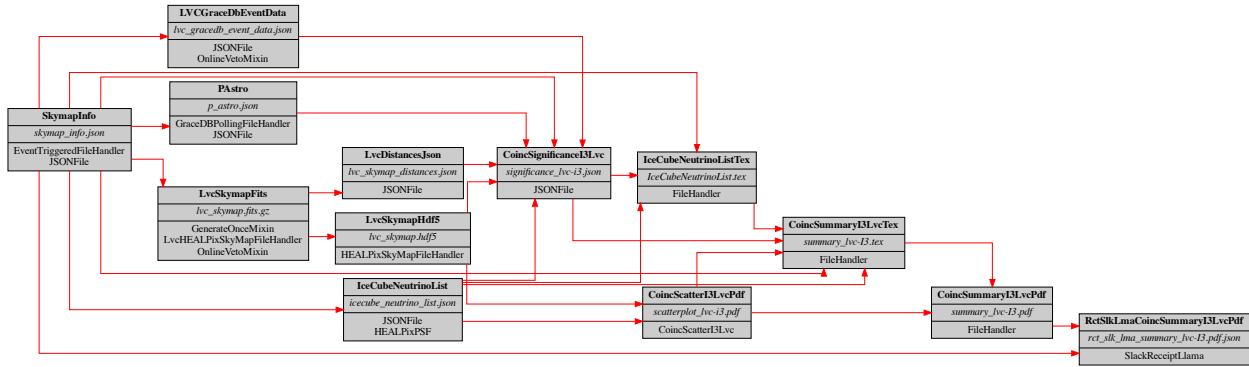


Fig. 33: Required input files for `llama.files.coinc_plots.RctSlkLmaCoincSummaryI3LvcPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkLmaCoincSummaryI3LvcPdf` to be generated.

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>, <class
'llama.files.i3.tex.IceCubeNeutrinoListTex'>, <class
'llama.files.coinc_plots.CoincSummaryI3LvcTex'>, <class
'llama.files.coinc_plots.CoincSummaryI3LvcPdf'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincSummaryI3LvcPdf'>)})

class_vetoies = ()

class llama.files.RctSlkLmaLVAlertJSON(eventid_or_fh, rundir=None)
    Bases: llama.files.slack.SlackReceiptLlama

    DEPENDENCIES = (<class 'llama.files.lvalert_json.LVAlertJSON'>, <class
'llama.files.skymap_info.SkymapInfo'>)

    FILENAME = 'rct_slk_lma_lvalert.json.json'
    FILENAME_FMT = 'rct_slk_lma_{}.json'
    MANIFEST_TYPES = (<class 'llama.files.slack.RctSlkLmaLVAlertJSON'>,)

    UPLOAD
        alias of llama.files.lvalert_json.LVAlertJSON

    UR_DEPENDENCIES = (<class 'llama.files.lvalert_json.LVAlertJSON'>, <class
'llama.files.skymap_info.SkymapInfo'>)

    UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvalert_json.LVAlertJSON'>)})

    class_vetoies = ()

class llama.files.RctSlkLmaLVCGraceDbEventData(eventid_or_fh, rundir=None)
    Bases: llama.files.slack.SlackReceiptLlama

```

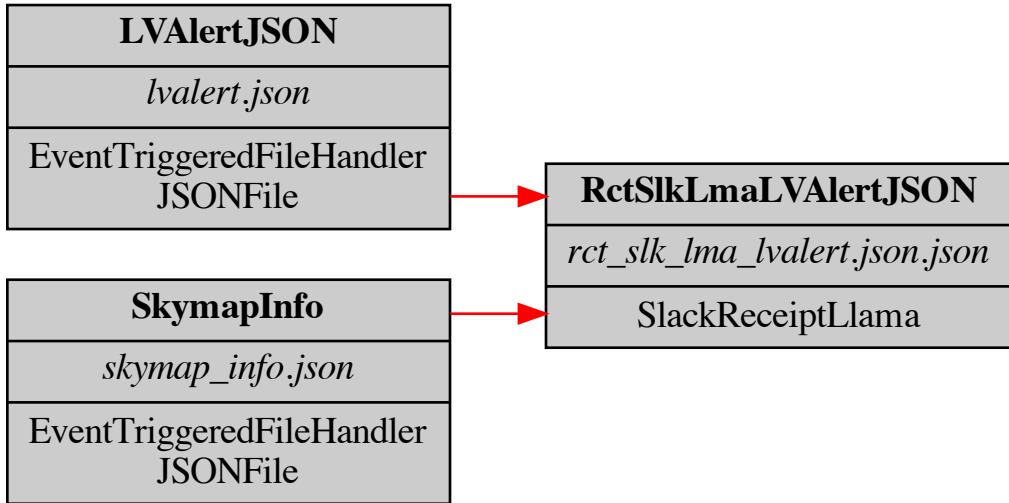


Fig. 34: Required input files for `llama.files.slack.RctSlkLmaLVAlertJSON` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.slack.RctSlkLmaLVAlertJSON` to be generated.

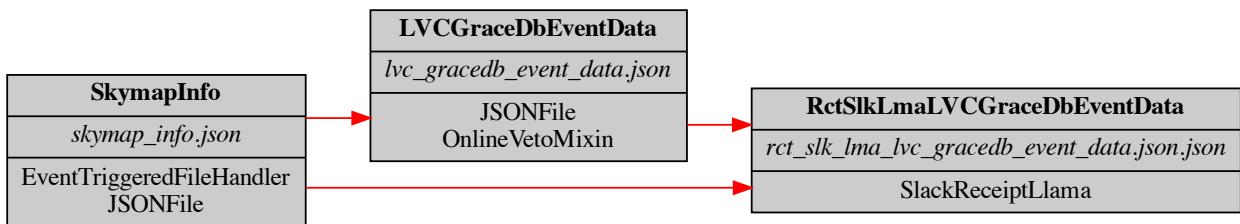


Fig. 35: Required input files for `llama.files.gracedb.RctSlkLmaLVCGraceDbEventData` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.gracedb.RctSlkLmaLVCGraceDbEventData` to be generated.

```

DEPENDENCIES = (<class 'llama.files.gracedb.LVCGraceDbEventData'>, <class
'skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_lvc_gracedb_event_data.json.json'

FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class 'llama.files.gracedb.RctSlkLmaLVCGraceDbEventData'>,)

UPLOAD
    alias of llama.files.gracedb.LVCGraceDbEventData

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'gracedb.LVCGraceDbEventData'>)

UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.gracedb.LVCGraceDbEventData'>})

class_vetoes = ()

class llama.files.RctSlkLmaLvcDistancesJson(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

```

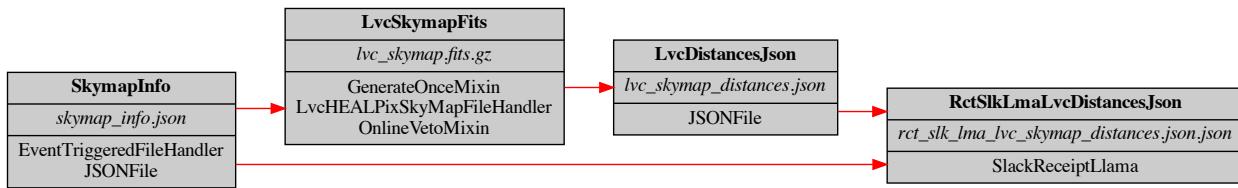


Fig. 36: Required input files for `llama.files.lvc_skymap.RctSlkLmaLvcDistancesJson` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.lvc_skymap.RctSlkLmaLvcDistancesJson` to be generated.

```

DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_lvc_skymap_distances.json.json'

FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class 'llama.files.lvc_skymap.RctSlkLmaLvcDistancesJson'>,)

UPLOAD
    alias of llama.files.lvc\_skymap.LvcDistancesJson

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'lvc_skymap.LvcSkymapFits'>, <class
'lvc_skymap.LvcDistancesJson'>)

UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvc_skymap.LvcDistancesJson'>})

class_vetoes = ()

class llama.files.RctSlkLmaLvcGcnXml(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

DEPENDENCIES = (<class 'llama.files.lvc_gcn_xml.LvcGcnXml'>, <class
'skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_lvc_gcn.xml.json'

FILENAME_FMT = 'rct_slk_lma_{}.json'

```

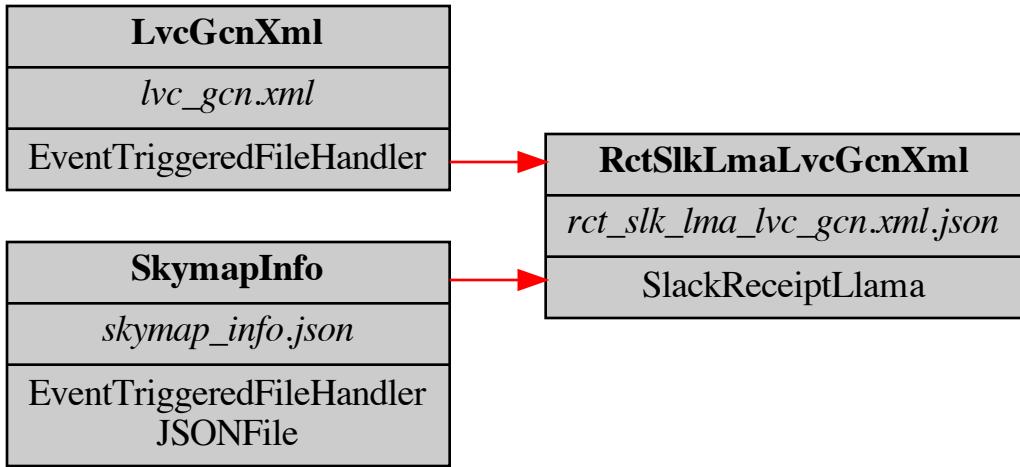


Fig. 37: Required input files for `llama.files.slack.RctSlkLmaLvcGcnXml` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.slack.RctSlkLmaLvcGcnXml` to be generated.

```

MANIFEST_TYPES = (<class 'llama.files.slack.RctSlkLmaLvcGcnXml'>,)

UPLOAD
alias of llama.files.lvc_gcn_xml.LvcGcnXml

UR_DEPENDENCIES = (<class 'llama.files.lvc_gcn_xml.LvcGcnXml'>, <class
'	llama.files.skymap_info.SkymapInfo'>)

UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvc_gcn_xml.LvcGcnXml'>})

class_vetoes = ()

class llama.files.RctSlkLmaLvcRetractionXml(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

```

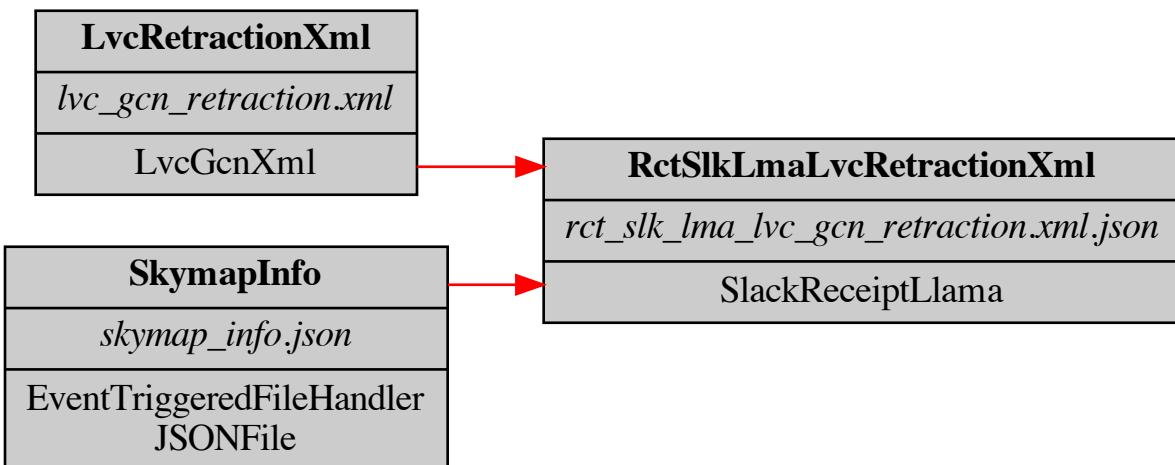


Fig. 38: Required input files for `llama.files.slack.RctSlkLmaLvcRetractionXml` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.slack.RctSlkLmaLvcRetractionXml` to be generated.

```

DEPENDENCIES = (<class 'llama.files.lvc_gcn_xml.LvcRetractionXml'>, <class
'skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_lvc_gcn_retraction.xml.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'
MANIFEST_TYPES = (<class 'llama.files.slack.RctSlkLmaLvcRetractionXml'>,)

UPLOAD
    alias of llama.files.lvc\_gcn\_xml.LvcRetractionXml

UR_DEPENDENCIES = (<class 'llama.files.lvc_gcn_xml.LvcRetractionXml'>, <class
'skymap_info.SkymapInfo'>)

UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvc_gcn_xml.LvcRetractionXml'>})

class_vetoes = ()

class llama.files.RctSlkLmaSkymapInfo(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

```

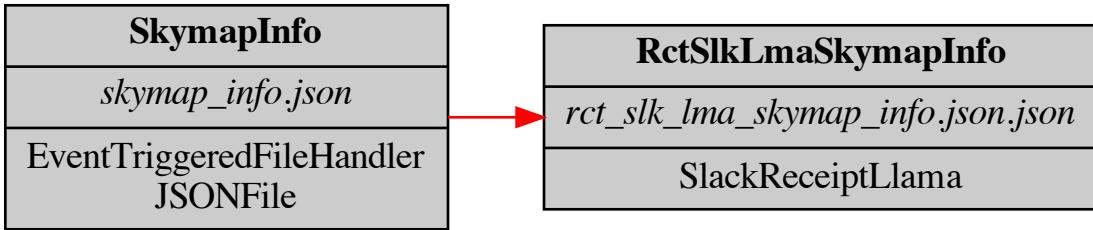


Fig. 39: Required input files for `llama.files.slack.RctSlkLmaSkymapInfo` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.slack.RctSlkLmaSkymapInfo` to be generated.

```

DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)

FILENAME = 'rct_slk_lma_skymap_info.json.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class 'llama.files.slack.RctSlkLmaSkymapInfo'>,)

UPLOAD
    alias of llama.files.skymap\_info.SkymapInfo

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)

UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})

class_vetoes = ()

class llama.files.SkymapInfo(eventid_or_fh, rundir=None)
Bases: llama.filehandler.EventTriggeredFileHandler, llama.filehandler.JSONFile

```

A JSON file containing information on where to find the latest and greatest skymap for this event. Any incoming alert file should also generate this file. The resulting file is a simple JSON object with key:value pairs concerning information about the event and the current skymap to be used.

```

FILENAME = 'skymap_info.json'

MANIFEST_TYPES = (<class 'llama.files.skymap_info.SkymapInfo'>,)

UR_DEPENDENCIES = ()

```

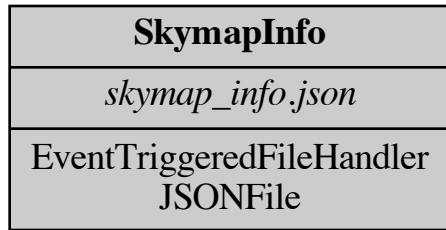


Fig. 40: `llama.files.skymap_info.SkymapInfo` is created from external triggers. It therefore has no LLAMA-representable input dependencies, but instead acts as initial input for other `FileHandler` classes.

```

UR_DEPENDENCY_TREE = frozenset({})

property alert_file_handler
    Get the file handler for this alert file type. For a manually-generated instance of this file, the return value is None; otherwise, it is a FileHandler instance matched to this event.

property alert_type
    Get the type of the alert, i.e. how this trigger was made in the LLAMA pipeline. For example, if this trigger was made as part of a gtlal offline subthreshold search, the value would be “gtlal-offline”. Look in SkymapInfo._alert_types for allowed alert type values.

NB: This is NOT related to the ``alert_type`` key in LVAlets!

property event_time_gps
    Get the time of the observed event up to nanosecond precision (less accurate than this in practice).

property event_time_gps_nanoseconds
    Get the number of nanoseconds past the last GPS second at the time of the observed event. Ostensibly provides nanosecond precision, but is less accurate than this in practice.

property event_time_gps_seconds
    Get the time of the observed event in GPS seconds (truncated to the nearest second).

property event_time_mjd
    Get the time of the observed event in Modified Julian Day format (UTC time).

property event_time_str
    Get a unicode string with the ISO date and time of the observed event straight from the VOEvent file. UTC time.

property far
    Get the false alarm rate of this VOEvent as a float value.

generate_from_gracedb(graceid: str, skymap_filename: Optional[str] = None)
    Supplement the original filename with data pulled from the GraceDB API on the event. Used for manually generating skymap_info.

generate_from_gracedb_superevent(graceid: str, skymap_filename: Optional[str] = None)
    Supplement the original filename with data pulled from the GraceDB API on the superevent. Used for manually generating skymap_info from a superevent. If no skymap is provided, pull down the most recently uploaded one defined in llama.files.lvc_skymap.SKYMAP_Filenames.

generate_from_lvc_gcn_xml(voe: llama.files.lvc_gcn_xml.LvcGcnXml)
    Generate this file from LvcGcnXml VOEvent. Used for implementation.

property gracedb_url
    Get the GraceDB URL for this GraceID, assuming it is in the expected format for GraceIDs. Will autodetect

```

if this is a superevent or regular event. Returns `None` if this doesn't look like a valid GraceID (but will not check on the actual gracedb server).

property graceid

Get the GraceID corresponding to this VOEvent.

property human_url

Get the human-viewable URL for this file (i.e. the one that can be accessed from a web-browser using interactive authentication)

property is_super

Check whether this GraceID corresponds to an event or a superevent.

property notice_time_str

Get a unicode string with the date and time of creation of the notification associated with this VOEvent, rather than the time of detection of the event itself.

property pipeline

Return an ALL-CAPS name of the pipeline used to generate this event, e.g. GSTLAL or CWB.

property skymap_filename

Get the filename of the skymap as a `SkymapFilename` as stored on GraceDB.

class `llama.files.ZtfTriggerList(eventid_or_fh, rundir=None)`

Bases: `llama.filehandler.JSONFile`, `llama.files.healpix.skymap.HEALPixPSF`

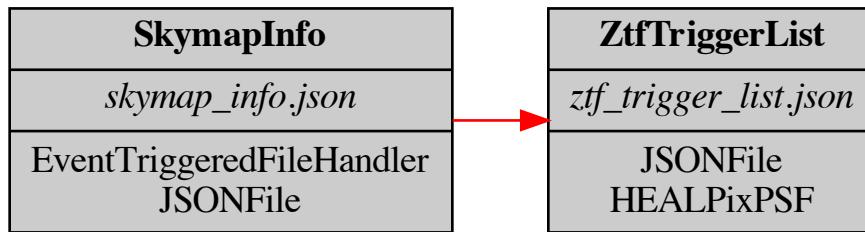


Fig. 41: Required input files for `llama.files.ztf_trigger_list.ZtfTriggerList` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.ztf_trigger_list.ZtfTriggerList` to be generated.

Takes a list of dictionaries describing neutrinos and saves it to a text file. Validates the neutrino data to make sure each neutrino contains (minimally) the following properties:

- gps
- ra
- dec

The sigma value is always expected to be negligible and is hence given as a small value (relative to GW skymap pixel sizes).

`DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)`

```

DETECTORS = (Detector(name='ZTF', abbrev='ztf', fullname='Zwicky Transient Facility', url='https://www.ptf.caltech.edu/page/ztf_technical', summary='ZTF', description='', citations=ImmutableDict({'The unblinking eye on the sky (Bellm and Kulkarni 2017)': 'http://adsabs.harvard.edu/abs/2017NatAs...1E..71B', 'The Zwicky Transient Facility (Bellm 2014)': 'http://adsabs.harvard.edu/abs/2014arXiv1410.8185B', 'The Zwicky transient facility observing system (Smith et al. 2014)': 'http://adsabs.harvard.edu/abs/2014SPIE.9147E..79S', 'The Zwicky Transient Facility Camera (Dekany et al. 2016)': 'http://adsabs.harvard.edu/abs/2016SPIE.9908E..5MD'})),)

FILENAME = 'ztf_trigger_list.json'

MANIFEST_TYPES = (<class 'llama.files.ztf_trigger_list.ZtfTriggerList'>,)

SIGMA = 0.1

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})
class_vetoes = ((<function always_veto>, None),)

property num_triggers
    The number of triggers described by this file. Useful mostly for quickly determining if this trigger list is empty.

source_locations()
    Returns a list of tuples of (RA, Dec, sigma) for all point-like sources, where (RA, Dec) is the central sky position and sigma is the standard deviation of the Gaussian PSF. Since this depends on the structure of data in the relevant file handler, it must be specified for each subclass.

property template_skymap_filehandler
    The HEALPixSkyMapFileHandler whose HEALPix parameters should be used as a template for the HEALPixSkyMap output by this FileHandler. It is assumed that this HEALPixSkyMapFileHandler only returns one skymap in its list; consequently, only the first skymap loaded by this file handler will be used. Note that the template skymap file handler is likely not a dependency of this file handler; it is only used as a default for formatting generated skymaps from this file handler's data.

property triggers
    Return a list containing a ZtfTrigger object for each trigger; more convenient to work with when writing formulae than dealing directly with the neutrino dictionaries.

```

28.1 llama.files.coinc_significance package

FileHandler classes for calculating significance of joint events with gravitational wave triggers. Includes both sub-threshold and Open Public Alert (OPA) GWEN searches.

@author: dvesk, stefco

28.1.1 `llama.files.coinc_significance.opa` module

FileHandler class for calculating significance of joint events with high-confidence gravitational wave triggers from Open Public Alerts (OPAs).

NOTE: equations after 8 increment up by 1. equations after 22 increment up by 2.

@author: dvesk, stefco

```
class llama.files.coinc_significance.opa.CoincSignificanceI3Lvc(eventid_or_fh, rundir=None)
    Bases: llama.filehandler.JSONFile
```

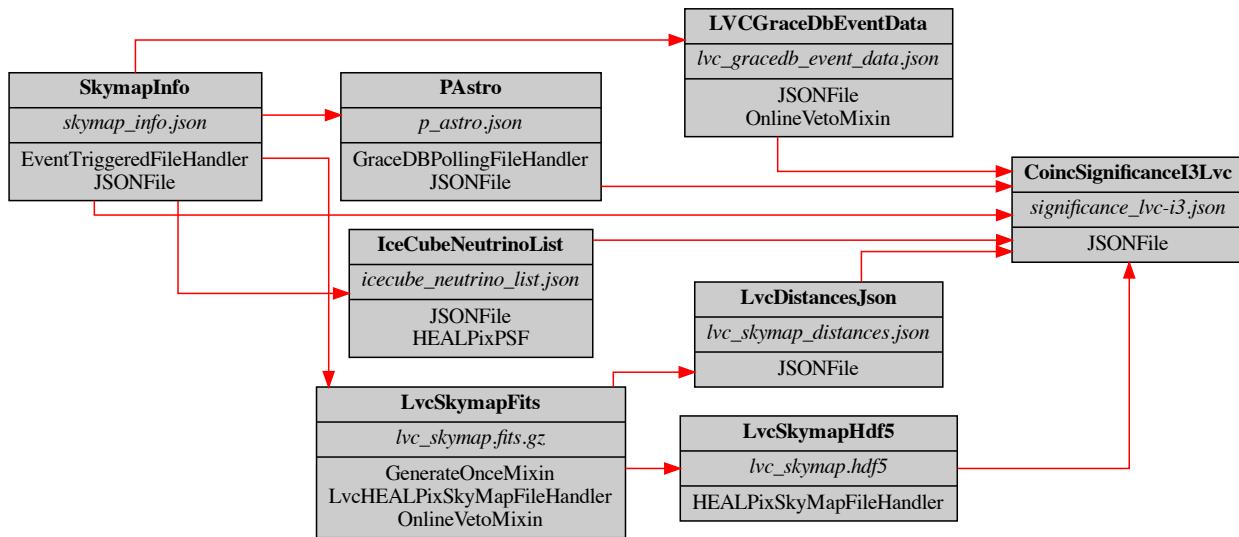


Fig. 42: Required input files for `llama.files.coinc_significance.opa.CoincSignificanceI3Lvc` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_significance.opa.CoincSignificanceI3Lvc` to be generated.

Calculates the significance (Bayes Factor) of a joint GW+HEN event.

```
DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.lvc_skymap.LvcDistancesJson'>, <class 'llama.files.gracedb.PAstro'>)

DETECTORS = (Detector(name='IceCube', abbrev='i3', fullname='IceCube',
url='http://wiki.icecube.wisc.edu', summary='IceCube', description='',
citations=ImmutableDict({})), Detector(name='LVC', abbrev='lvc', fullname='LVC',
url='http://wiki.ligo.org', summary='LVC', description='',
citations=ImmutableDict({})))

FILENAME = 'significance_lvc-i3.json'

MANIFEST_TYPES = (<class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>,)

TIMEOUT = 600
```

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.gracedb.LVCGraceDbEventData': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'llama.files.lvc_skymap.LvcDistancesJson': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}), <class
'llama.files.gracedb.PAstro': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})})})

```

property combined_p_value
Return the combined p-value for the whole event.

odds()
Return a list of the odds ratios calculated. Ordering corresponds to the ordering of neutrinos in the IceCubeNeutrinoList file used to calculate the odds ratios.

property p_values
Return a list of the p-values calculated. Ordering corresponds to the ordering of neutrinos in the IceCubeNeutrinoList file used to calculate the odds ratios.

`llama.files.coinc_significance.opa.PEgw(Egw, search_params)`
Probability of isotropic-equivalent GW emission energy Egw given the signal hypothesis, P(E_{GW}).

Parameters

- **Egw** (*float*) – The assumed isotropic GW emission energy. We will marginalize over this value to get Rdet and the signal likelihoods.
- **search_params** (*IceCubeLvcSearchParameters*) – A collection of constant parameters used for this search.

Returns `p_e_gw` – P(E_{GW}).**Return type** float`llama.files.coinc_significance.opa.PEnu(Enu, search_params)`

Probability of isotropic-equivalent neutrino emission energy Enu given the signal hypothesis, P(E_{nu}).

Parameters

- **Enu** (*float*) – The assumed isotropic neutrino emission energy. We will marginalize over this value to get Rdet and the signal likelihoods.
- **search_params** (*IceCubeLvcSearchParameters*) – A collection of constant parameters used for this search.

Returns `p_e_nu` – P(E_{nu}).**Return type** float

```
class llama.files.coinc_significance.opa.RctSlkLmaCoincSignificanceI3Lvc(eventid_or_fh,
rundir=None)

Bases: llama.files.slack.SlackReceiptLlama
```

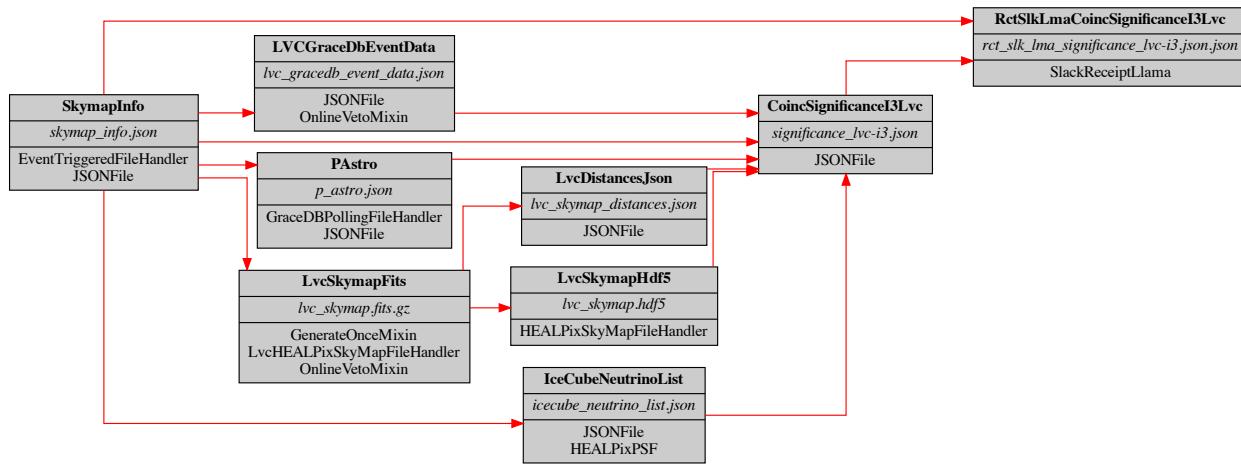


Fig. 43: Required input files for `llama.files.coinc_significance.opa.RctSlkLmaCoincSignificanceI3Lvc` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_significance.opa.RctSlkLmaCoincSignificanceI3Lvc` to be generated.

```

DEPENDENCIES = (<class 'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>,
<class 'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_significance_lvc-i3.json.json'

FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class
'	llama.files.coinc_significance.opa.RctSlkLmaCoincSignificanceI3Lvc'>,)

UPLOAD
    alias of llama.files.coinc\_significance.opa.CoincSignificanceI3Lvc

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'	llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'	llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'	llama.files.i3.json.IceCubeNeutrinoList'>, <class
'	llama.files.gracedb.LVCGraceDbEventData'>, <class
'	llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)

UR_DEPENDENCY_TREE = frozenset({<class
'	llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)})

class_vetoies = ()

llama.files.coinc_significance.opa.expnu(Enu, r, search_params)

Expected number of neutrinos for a given emission energy and distance, <math>n_{\{u\}}(E_{\{u\}}, r)>

Enu [float] Neutrino emission energy [kg Mpc**2 / sec**2].
```

r [float] Distance to the hypothesized neutrino source. [Mpc]

search_params [IceCubeLvcSearchParameters] A collection of constant parameters used for this search.

```
exp_nu [float] < n_{  
u}(E_{ u}, r) >  
llama.files.coinc_significance.opa.fA(theta, phi, t)  
The antenna pattern at direction theta, phi and time t.  
llama.files.coinc_significance.opa.five(tgw, gw_skymap, pgw, dist, neutrino, search_params, enu)  
Calculate the signal likelihood (Eq. 5).
```

Parameters

- **tgw** (float) – GPS time of the GW trigger
- **gw_skymap** (llama.files.healpix.skymap.HEALPixSkyMap) – A HEALPix skymap for the GW specifying pixel values and locations.
- **pgw** (float) – The signal-to-noise ratio (SNR) of the GW
- **dist** (float) – The probability-weighted average reconstructed distance of the GW event
- **neutrino** (llama.files.i3.Neutrino) – The neutrino candidate that is being compared to the GW skymap
- **search_params** (IceCubeLvcSearchParameters) – A collection of constant parameters used for this search.
- **enu** (float) – The neutrino energy. This should be capped at the maximum energy of a nearby background neutrino. This way the signal likelihood does not drop (as it does for increasing energy) while the background stays the same (since the background calculation always assumes at least one neutrino).

Returns

P_H_S1 –

The signal hypothesis likelihood for a single neutrino:

AllskyIntegral($P(x_{\text{gw}}, x_{\text{nu}} | \theta, H_s) * P(\theta | H_s)$)

Return type

```
llama.files.coinc_significance.opa.five2(tgw, gw_skymap, pgw, dist, neutrino, neutrino_i,  
search_params, enu, enu_i)
```

Calculate the signal likelihood for 2 coincident neutrinos (modified Eq. 5)

Parameters

- **tgw** (float) – GPS time of the GW trigger
- **gw_skymap** (llama.files.healpix.skymap.HEALPixSkyMap) – A HEALPix skymap for the GW specifying pixel values and locations.
- **pgw** (float) – The signal-to-noise ratio (SNR) of the GW
- **dist** (float) – The probability-weighted average reconstructed distance of the GW event
- **neutrino** (llama.files.i3.Neutrino) – The neutrino candidate that is being compared to the GW skymap as well as **neutrino_i**.

- **neutrino_i** (`llama.files.i3.Neutrino`) – The ith neutrino candidate that is being compared to the GW skymap as well as **neutrino**.
- **search_params** (`IceCubeLvcSearchParameters`) – A collection of constant parameters used for this search.
- **enu** (`float`) – The neutrino energy. This should be capped at the maximum energy of a nearby background neutrino. This way the signal likelihood does not drop (as it does for increasing energy) while the background stays the same (since the background calculation always assumes at least one neutrino).
- **enu_i** (`float`) – Same as **enu**, but for the ith neutrino.

Returns**P_H_S2** –

The signal hypothesis likelihood for a double-neutrino detection:

$$\text{AllskyIntegral}(P(x_{\text{gw}}, x_{\text{nu}}, x_{\text{nu_i}} | \theta, H_s) * P(\theta | H_s))$$

Return type float`llama.files.coinc_significance.opa.iA(search_params)`Integrated antenna factor for detector network given by **detectors**.`llama.files.coinc_significance.opa.odds_ratio(tgw, gw_skymap, pgw, dist, neutrino, search_params, neutrinolist, count, count2)`

Calculate the odds ratio for the signal vs. the null+chance coincidence hypothesis. This is the main measure of significance for the pipeline. Also returns the factors that went into calculating the odds ratio.

Parameters

- **tgw** (`float`) – GPS time of the GW trigger
- **gw_skymap** (`llama.files.healpix.skymap.HEALPixSkyMap`) – A HEALPix skymap for the GW specifying pixel values and locations.
- **pgw** (`float`) – The signal-to-noise ratio (SNR) of the GW
- **dist** (`float`) – The probability-weighted average reconstructed distance of the GW event
- **neutrino** (`llama.files.i3.Neutrino`) – The neutrino candidate that is being compared to the GW skymap
- **search_params** (`IceCubeLvcSearchParameters`) – A collection of constant parameters used for this search.
- **neutrinolist** (`list`) – A list of `llama.files.i3.Neutrino` instances describing the other neutrinos in this search (used for the 2-neutrino search, `five2`).
- **count** (`np.array`) – An array tracking whether two neutrinos have been compared. If an entry is 1, then they are considered to have been compared (the diagonal entries should be set to 1 at the start).
- **count2** (`int`) – The index of the current primary neutrino in **neutrinolist**.

Returns

- **odds** (`float`) –

The odds ratio:

$$P(H_s|x_{\text{gw}}, x_{\text{nu}})$$

$$P(H_0|x_{\text{gw}}, x_{\text{nu}}) + P(H_c|x_{\text{gw}}, x_{\text{nu}})$$

- **p_h_s1** (*float*) – The signal probability (times the marginal likelihood, cancels everywhere) for the single-neutrino detection case:

$$P(H_{s_1}|x_{gw}, x_{nu}) * P(x_{gw}, x_{nu})$$
- **p_h_s2** (*float*) – The signal probability (times the marginal likelihood, cancels everywhere) for the double-neutrino detection case:

$$P(H_{s_2}|x_{gw}, x_{nu1}, x_{nu2}) * P(x_{gw}, x_{nu1}, x_{nu2})$$
- **p_h_0** (*float*) – The null hypothesis probability (times the marginal likelihood, cancels everywhere):

$$P(H_0|x_{gw}, x_{nu}) * P(x_{gw}, x_{nu})$$
- **p_h_c** (*float*) – The chance coincidence hypothesis probability (times the marginal likelihood, cancels everywhere):

$$P(H_0|x_{gw}, x_{nu}) * P(x_{gw}, x_{nu})$$
- **p_value** (*float*) – The p-value or false-alarm probability (FAP) of the given `odds_ratio` assuming the given source population and detector configuration.
- **dump** (*dict*) – Local variables with scalar values from the function (for debugging). The contents of this object are not guaranteed to equal anything in particular and should not be used for anything science-related.

28.1.2 `llama.files.coinc_significance.subthreshold` module

`FileHandler` class for calculating significance of joint events with subthreshold gravitational wave triggers.

NOTE: equations after 8 increment up by 1. equations after 22 increment up by 2.

@author: dvesk, stefco

```
class llama.files.coinc_significance.subthreshold.CoincSignificanceSubthresholdI3Lvc(eventid_or_fh,
                                                                                      rundir=None)
```

Bases: `llama.filehandler.JSONFile`

Calculates the significance (Bayes Factor) of a joint GW+HEN event.

```
DEPENDENCIES = (<class 'llama.files.lvalert_json.LVAlertJSON'>, <class
'	llama.files.skymap_info.SkymapInfo'>, <class
'	llama.files.i3.json.IceCubeNeutrinoList'>, <class
'	llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'	llama.files.gracedb.LVCGraceDbEventData'>, <class
'	llama.files.lvc_skymap.LvcDistancesJson'>, <class 'llama.files.gracedb.PAstro'>)
```

```
DETECTORS = (Detector(name='IceCube', abbrev='i3', fullname='IceCube',
url='http://wiki.icecube.wisc.edu', summary='IceCube', description='',
citations=ImmutableDict({})), Detector(name='LVC', abbrev='lvc', fullname='LVC',
url='http://wiki.ligo.org', summary='LVC', description='',
citations=ImmutableDict({})))
```

```
FILENAME = 'significance_subthresh_lvc-i3.json'
```

```
MANIFEST_TYPES = (<class
'	llama.files.coinc_significance.subthreshold.CoincSignificanceSubthresholdI3Lvc'>,)
```

```
TIMEOUT = 600
```

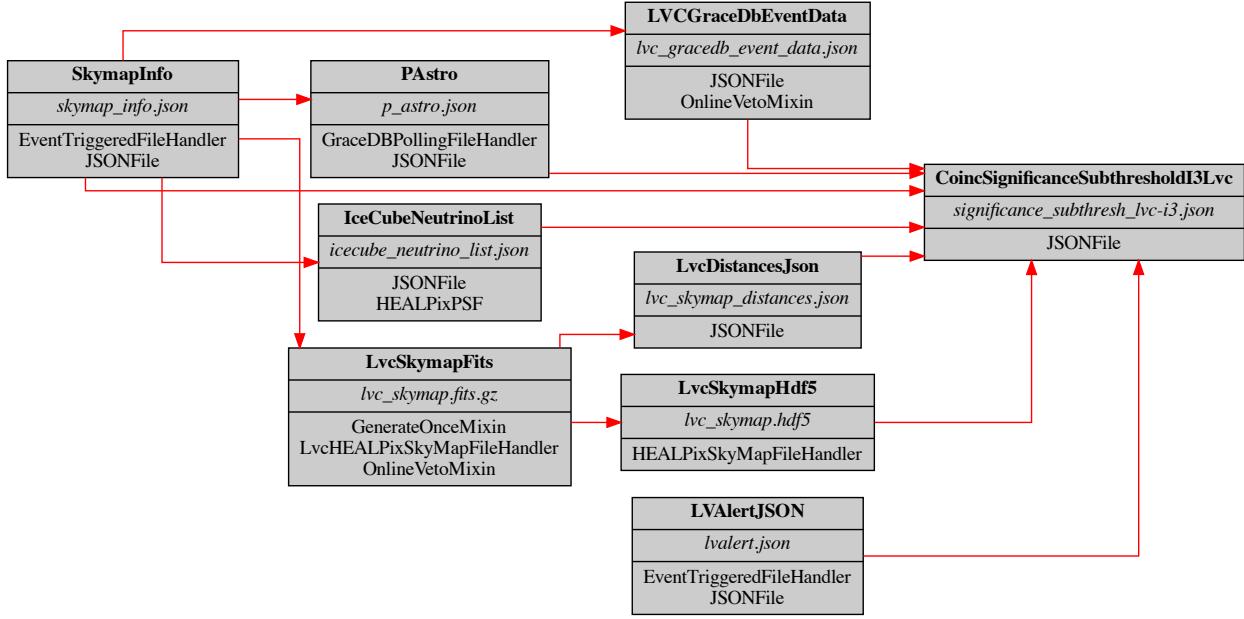


Fig. 44: Required input files for `llama.files.coinc_significance.subthreshold.CoincSignificanceSubthresholdI3Lvc` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_significance.subthreshold.CoincSignificanceSubthresholdI3Lvc` to be generated.

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.lvalert_json.LVALertJSON'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.gracedb.LVCGraceDbEventData': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})})}), <class
'llama.files.lvc_skymap.LvcDistancesJson': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})})}), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}), <class
'llama.files.gracedb.PAstro': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'lvalert_json.LVALertJSON': ImmutableDict({})}
property combined_p_value
    Return the combined p-value for the whole event.
  
```

`llama.files.coinc_significance.subthreshold.PEgw(Egw, search_params)`

Probability of isotropic-equivalent GW emission energy Egw given the signal hypothesis, P(E_{GW}).

Parameters

- `Egw` (`float`) – The assumed isotropic GW emission energy. We will marginalize over this value to get Rdet and the signal likelihoods.
- `search_params` (`IceCubeLvcSearchParameters`) – A collection of constant parameters used for this search.

Returns `p_e_gw` – $P(E_{\{GW\}})$.

Return type float

`llama.files.coinc_significance.subthreshold.PEn(En, search_params)`

Probability of isotropic-equivalent neutrino emission energy En given the signal hypothesis, P(E_{nu}).

Parameters

- `E_n` (`float`) – The assumed isotropic neutrino emission energy. We will marginalize over this value to get Rdet and the signal likelihoods.
- `search_params` (`IceCubeLvcSearchParameters`) – A collection of constant parameters used for this search.

Returns `p_e_nu` – $P(E_{\{nu\}})$.

Return type float

`class llama.files.coinc_significance.subthreshold.RctSlkLmaCoincSignificanceSubthresholdI3Lvc(eventid_or_j
rundir=None)`

Bases: `llama.files.slack.SlackReceiptLlama`

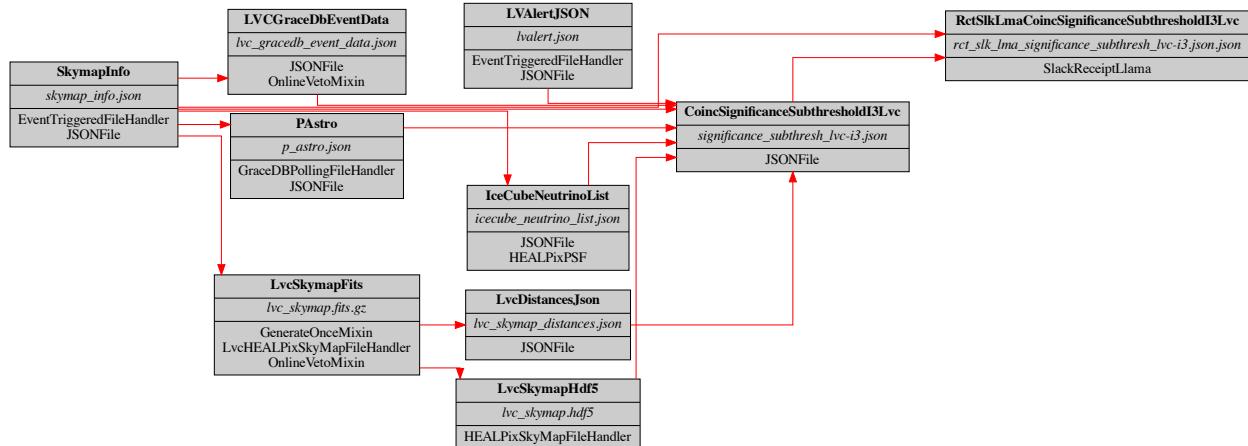


Fig. 45: Required input files for `llama.files.coinc_significance.subthreshold.RctSlkLmaCoincSignificanceSubthresholdI3Lvc` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_significance.subthreshold.RctSlkLmaCoincSignificanceSubthresholdI3Lvc` to be generated.

```

DEPENDENCIES = (<class
'llama.files.coinc_significance.subthreshold.CoincSignificanceSubthresholdI3Lvc'>,
<class 'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_significance_subthresh_lvc-i3.json.json'
  
```

```
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class 'llama.files.coinc_significance.subthreshold.RctSlkLmaCoincSignificanceSubthresholdI3Lvc'>,)

UPLOAD
    alias of llama.files.coinc\_significance.subthreshold.CoincSignificanceSubthresholdI3Lvc

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
    'llama.files.lvc_skymap.LvcSkymapFits'>, <class
    'llama.files.lvalert_json.LVAlertJSON'>, <class 'llama.files.gracedb.PAstro'>,
    <class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
    'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
    'llama.files.i3.json.IceCubeNeutrinoList'>, <class
    'llama.files.gracedb.LVCGraceDbEventData'>, <class
    'llama.files.coinc_significance.subthreshold.CoincSignificanceSubthresholdI3Lvc'>)

UR_DEPENDENCY_TREE = frozenset({<class
    'llama.files.coinc_significance.subthreshold.CoincSignificanceSubthresholdI3Lvc'>})

class_vetoes = ()
```

`llama.files.coinc_significance.subthreshold.emptyskymap(val, skymap)`
A HEALPixSkyMap initialized with a uniform value.

Parameters

- `val (float or int)` – The value to assign each pixel.
- `skymap (HEALPixSkyMap)` – A skymap whose shape should be used as the template.

Returns `skymap` – A skymap shaped like `skymap` but with all pixel values set to `val`.

Return type `HEALPixSkyMap`

`llama.files.coinc_significance.subthreshold.expnu(En, r, search_params)`
Expected number of neutrinos for a given emission energy and distance, $\langle n_{\nu}(E_{\nu}, r) \rangle$

Parameters

- `E_n (float)` – Neutrino emission energy [kg Mpc **2 / sec **2].
- `r (float)` – Distance to the hypothesized neutrino source. [Mpc]
- `search_params (IceCubeLvcSearchParameters)` – A collection of constant parameters used for this search.

Returns `exp_nu` – $\langle n_{\nu}(E_{\nu}, r) \rangle$

Return type float

`llama.files.coinc_significance.subthreshold.fA(theta, phi, t)`
The antenna pattern at direction θ , phi and time t.

`llama.files.coinc_significance.subthreshold.five(tgw, gw_skymap, pgw, dist, neutrinolist, search_params, En, pempnu)`

Calculate the signal likelihood (Eq. 5).

Parameters

- `tgw (float)` – GPS time of the GW trigger
- `gw_skymap (llama.files.healpix.skymap.HEALPixSkyMap)` – A HEALPix skymap for the GW specifying pixel values and locations.
- `pgw (float)` – The signal-to-noise ratio (SNR) of the GW

- **dist** (*float*) – The probability-weighted average reconstructed distance of the GW event
- **neutrino** (`llama.files.i3.Neutrino`) – The neutrino candidate that is being compared to the GW skymap
- **search_params** (`IceCubeLvcSearchParameters`) – A collection of constant parameters used for this search.
- **E_n** (*float*) – The neutrino energy. This should be capped at the maximum energy of a nearby background neutrino. This way the signal likelihood does not drop (as it does for increasing energy) while the background stays the same (since the background calculation always assumes at least one neutrino).

Returns

P_H_S1 –

The signal hypothesis likelihood for a single neutrino:

$$\text{AllskyIntegral}(P(x_{\text{gw}}, x_{\text{nu}} | \theta, H_s) * P(\theta | H_s))$$

Return type float

```
llama.files.coinc_significance.subthreshold.five2(tgw, gw_skymap, rho_gw, dist, neutrino, neutrino_i,
                                              search_params, En, En_i)
```

Calculate the signal likelihood for 2 coincident neutrinos (modified Eq. 5)

Parameters

- **tgw** (*float*) – GPS time of the GW trigger
- **gw_skymap** (`llama.files.healpix.skymap.HEALPixSkyMap`) – A HEALPix skymap for the GW specifying pixel values and locations.
- **rho_gw** (*float*) – The signal-to-noise ratio (SNR) of the GW
- **dist** (*float*) – The probability-weighted average reconstructed distance of the GW event
- **neutrino** (`llama.files.i3.Neutrino`) – The neutrino candidate that is being compared to the GW skymap as well as neutrino_i.
- **neutrino_i** (`llama.files.i3.Neutrino`) – The ith neutrino candidate that is being compared to the GW skymap as well as neutrino.
- **search_params** (`IceCubeLvcSearchParameters`) – A collection of constant parameters used for this search.
- **E_n** (*float*) – The neutrino energy. This should be capped at the maximum energy of a nearby background neutrino. This way the signal likelihood does not drop (as it does for increasing energy) while the background stays the same (since the background calculation always assumes at least one neutrino).
- **E_{n_i}** (*float*) – Same as E_n, but for the ith neutrino.

Returns

P_H_S2 –

The signal hypothesis likelihood for a double-neutrino detection:

$$\text{AllskyIntegral}(P(x_{\text{gw}}, x_{\text{nu}}, x_{\text{nu}_i} | \theta, H_s) * P(\theta | H_s))$$

Return type float

```
llama.files.coinc_significance.subthreshold.five2wogw(neutrino, neutrino_i, search_params, En,
En_i)
```

NOT USED Calculate the coincidence likelihood for 2 coincident neutrinos (modified Eq. 58)

Parameters

- **neutrino** ([llama.files.i3.Neutrino](#)) – The neutrino candidate that is being compared to the GW skymap as well as **neutrino_i**.
- **neutrino_i** ([llama.files.i3.Neutrino](#)) – The ith neutrino candidate that is being compared to the GW skymap as well as **neutrino**.
- **search_params** ([IceCubeLvcSearchParameters](#)) – A collection of constant parameters used for this search.
- **E_n** ([float](#)) – The neutrino energy. This should be capped at the maximum energy of a nearby background neutrino. This way the signal likelihood does not drop (as it does for increasing energy) while the background stays the same (since the background calculation always assumes at least one neutrino).
- **E_{n-i}** ([float](#)) – Same as E_n, but for the ith neutrino.

Returns

P_H_c2 –

The coincidence hypothesis likelihood for a double-neutrino detection:

```
AllskyIntegral(P(x_nu, x_nu_i | θ, H_c)*P(θ | H_c))
```

Return type

```
llama.files.coinc_significance.subthreshold.fivewogw(gw_skymap, neutrino_list, search_params, En,
pempnu)
```

Calculate the coincidence likelihood (Eq. 58).

Parameters

- **neutrino** ([llama.files.i3.Neutrino](#)) – The neutrino candidate that is being compared to the GW skymap
- **search_params** ([IceCubeLvcSearchParameters](#)) – A collection of constant parameters used for this search.
- **E_n** ([float](#)) – The neutrino energy. This should be capped at the maximum energy of a nearby background neutrino. This way the signal likelihood does not drop (as it does for increasing energy) while the background stays the same (since the background calculation always assumes at least one neutrino).

Returns

P_H_c1 –

The coincidence hypothesis likelihood for a single neutrino:

```
AllskyIntegral(P(x_nu | θ, H_c)*P(θ | H_c))
```

Return type

```
llama.files.coinc_significance.subthreshold.iA(search_params)
```

Integrated antenna factor for detector network given by **detectors**.

```
llama.files.coinc_significance.subthreshold.odds_ratio(tgw, gw_skymap, pgw, dist, search_params,
                                                     neutrinolist, p_ter)
```

Calculate the odds ratio for the signal vs. the null+chance coincidence hypothesis. This is the main measure of significance for the pipeline. Also returns the factors that went into calculating the odds ratio. **Calculates the odds ratio for an entire ensemble of neutrino candidates** in contrast with the OPA version (which calculates p-value on a per-neutrino basis).

Parameters

- **tgw** (*float*) – GPS time of the GW trigger
- **gw_skymap** ([llama.files.healpix.skymap.HEALPixSkyMap](#)) – A HEALPix skymap for the GW specifying pixel values and locations.
- **ρ_{gw}** (*float*) – The signal-to-noise ratio (SNR) of the GW
- **dist** (*float*) – The probability-weighted average reconstructed distance of the GW event
- **search_params** ([IceCubeLvcSearchParameters](#)) – A collection of constant parameters used for this search.
- **neutrinolist** (*list*) – A list of [llama.files.i3.Neutrino](#) instances describing the other neutrinos in this search (used for the 2-neutrino search, `five2`).

Returns

- **odds** (*float*) –

The odds ratio:

$$P(H_s|x_{\text{gw}}, x_{\text{nu}})$$

$$P(H_0|x_{\text{gw}}, x_{\text{nu}}) + P(H_c|x_{\text{gw}}, x_{\text{nu}})$$

- **p_h^{s1}** (*float*) – The signal probability (times the marginal likelihood, cancels everywhere) for the single-neutrino detection case:

$$P(H_{s_1}|x_{\text{gw}}, x_{\text{nu}}) * P(x_{\text{gw}}, x_{\text{nu}})$$

- **p_h^{s2}** (*float*) – The signal probability (times the marginal likelihood, cancels everywhere) for the double-neutrino detection case:

$$P(H_{s_2}|x_{\text{gw}}, x_{\text{nu1}}, x_{\text{nu2}}) * P(x_{\text{gw}}, x_{\text{nu1}}, x_{\text{nu2}})$$

- **p_h^0** (*float*) – The null hypothesis probability (times the marginal likelihood, cancels everywhere):

$$P(H_0|x_{\text{gw}}, x_{\text{nu}}) * P(x_{\text{gw}}, x_{\text{nu}})$$

- **p_h^c** (*float*) – The chance coincidence hypothesis probability (times the marginal likelihood, cancels everywhere):

$$P(H_0|x_{\text{gw}}, x_{\text{nu}}) * P(x_{\text{gw}}, x_{\text{nu}})$$

- **p_value** (*float*) – The p-value or false-alarm probability (FAP) of the given `odds_ratio` assuming the given source population and detector configuration.

- **dump** (*dict*) – Local variables with scalar values from the function (for debugging). The contents of this object are not guaranteed to equal anything in particular and should not be used for anything science-related.

28.1.3 `llama.files.coinc_significance.utils` module

Utility functions for calculating significance of joint events.

NOTE: equations after 8 increment up by 1. equations after 22 increment up by 2.

@author: dvesk, stefco

`llama.files.coinc_significance.utils.Aeff(e, theta)`

`llama.files.coinc_significance.utils.Aeff_skymap(skymap, enu)`

Get the energy- and direction-dependent effective areas of the neutrino detector for every direction provided in skymap (Eq. 21, 22)

Parameters

- `skymap` (`llama.files.healpix.HEALPixSkyMap`) – A HEALPixSkyMap instance with defined right-ascensions and declinations.
- `enu` (`float`) – The reconstructed neutrino energy.

`llama.files.coinc_significance.utils.Pempgw(pgw, gw_bg_ps)`

The empirical neutrino background for a given GW SNR `pgw`.

Parameters

- `pgw` (`float`) – The SNR of the current GW trigger.
- `gw_bg_ps` (`RemoteFileCacher`) – A file with a list of historical background SNR values.

`llama.files.coinc_significance.utils.aeff_lookup_table()`

Get a dictionary mapping from strings describing neutrino energy bounds to tuples of the form (factor, exponent) where `factor` is a lookup table of the constant factor multiplying the effective area (with indices representing 10-degree increments in the angle theta) and `exponent` is the exponent to which the neutrino energy `e` is raised (since we assume a rough power law for `Aeff`, again with indices representing 10-degree increments in the angle theta). The final value will be `factor**e**exponent`.

NOTE that we need to include an item for [180, 190) degrees because `int(180./10)` will give us 18, indexing into the 19th entry in the lookup table, and we want to include this edge case.

`llama.files.coinc_significance.utils.angle_list()`

`llama.files.coinc_significance.utils.load_neutrino_background()`

Load neutrino background data.

Returns

- `thetabg` (`array-like`) – Array of background neutrino zenith angles [deg] as reconstructed by IceCube.
- `EGeV` (`array-like`) – Array of background neutrino energies [GeV] as reconstructed by IceCube.

`llama.files.coinc_significance.utils.p_value(odds_ratio, search_type, population, sensitivity, instruments)`

Calculate the p-value for a given odds ratio.

Parameters

- `odds_ratio` (`float`) – The odds ratio for signal vs. background/null; is here being interpreted as a test statistic to make the analysis robust against constant factors in the priors.
- `search_type` (`str`) – The type of search performed. This decides which background population will be used. Distinguishes between subthreshold, open public alert, and in the future, potentially other simulated backgrounds.

- **population (str)** – The name of the source population against which this `odds_ratio` will be compared, e.g. ‘bns’ for binary neutron stars. *NOTE:* This just has to be set so that it references an existing group in the `NEUTRINO_POPULATIONS` HDF5 file; there are currently no explicit standards for naming the population. See `populations-hdf5-collater` and the contents of the directory containing `NEUTRINO_POPULATIONS` for available options.
- **sensitivity (str)** – The detector configuration and sensitivity used, e.g. ‘design’ for HLV at design sensitivity. See the *NOTE* on `population` for naming information.
- **instruments (list)** – The instruments participating in this search as a list of detector names, e.g. [H1, L1, V1] for all three O3 GW detectors. If a background population is available for this set of instruments, it will be used; otherwise, an attempt will be made to use a population averaging over all sets of participating detectors (for populations where full separate cases could not be feasibly generated).

Returns p_value – The p-value or false-alarm probability (FAP) of the given `odds_ratio` assuming the given source population and detector configuration. Returns `None` if a matching background does not yet exist for this search configuration.

Return type float or `NoneType`

`llama.files.coinc_significance.utils.r0(Egw, search_params)`

Maximum GW detection distance for isotropic-equivalent emission energy `Egw`.

`llama.files.coinc_significance.utils.search_parameters(population, instruments)`

Return an `IceCubeLvcSearchParameters` instance with all default values and the given `population` and `instruments`.

population [str] The name of the source population against which this `odds_ratio` will be compared, e.g. ‘bns’ for binary neutron stars. *NOTE:* This just has to be set so that it references an existing group in the `NEUTRINO_POPULATIONS` HDF5 file; there are currently no explicit standards for naming the population. See `populations-hdf5-collater` and the contents of the directory containing `NEUTRINO_POPULATIONS` for available options.

instruments [list] The GW detectors participating in this search as a list of detector names, e.g. [H1, L1, V1] for all three O3 GW detectors. If a background population is available for this set of instruments, it will be used; otherwise, an attempt will be made to use a population averaging over all sets of participating instruments (for populations where full separate cases could not be feasibly generated).

Returns search_params – Default search parameter values along with the provided `population` and `instruments`.

Return type `IceCubeLvcSearchParameters`

`llama.files.coinc_significance.utils.t_overlap(tgw, tnu, search_params)`

Integral of the source-time-dependent part of the likelihood integral (Eq. 5) finding the probability of getting `tgw` and `tnu` (the detection times of the gravitational wave trigger and neutrino trigger, respectively) given the specified `search_params`. Factors out from the rest of Eq. 5.

28.2 llama.files.healpix package

FileHandlers associated with HEALPix skymap loading and manipulation. Provides a subclass of FileHandler that specializes in loading and saving HEALPix FITS file skymaps and putting arbitrary skymap pixelizations and functions on the sphere into identical HEALPix pixelization schemes to simplify mathematical analyses.

Some documentation: <http://healpix.sourceforge.net/html/intronode4.htm> https://healpy.readthedocs.io/en/latest/healpy_pix.html

class `llama.files.healpix.HEALPixPlots(eventid_or_fh, rundir=None)`

Bases: `llama.filehandler.FileHandler`

Load HEALPix skymaps and save Mollweide-projection plots in PDF and PNG format to a compressed tarfile.

classmethod `set_class_attributes(subclass)`

See `FileHandler.set_class_attributes`; this method sets programatically-generated filenames for this class.

class `llama.files.healpix.HEALPixSkyMapAnalysis(eventid_or_fh, rundir=None)`

Bases: `llama.files.healpix.HEALPixSkyMapFileHandler`

Create output skymaps that are outputs of functions on input HEALPix skymaps.

`FILENAME_PREFIX = None`

static `analysis_kernel(*skymaps)`

A function that operates on an arbitrary number of skymaps and returns a single skymap. This is where the actual analysis algorithm is defined. Defaults to a simple elementwise product of skymaps, but can be overridden to do more complex analyses.

analyze_skymaps(*skymap_lists)

A function that takes an arbitrary number N of lists of input skymaps as its arguments and returns (`results, indices`), a tuple of lists where `results` is a list of results of combinations of skymaps from each skymap list (as output by `analysis_kernel`) and `indices` is a list of tuples of indices indicating which skymap was used from each skymap list.

classmethod `set_class_attributes(subclass)`

See `FileHandler.set_class_attributes`; this method additionally sets the `FILENAME` and `DETECTORS` attributes based on `subclass.DEPENDENCIES` and `subclass.FILENAME_PREFIX`.

write_healpix(skymaps, descriptions=None)

Write skymap, which should be a `HEALPixSkyMap` instance or a list containing a *single* `HEALPixSkyMap` instance, to the full path specified by this `FileHandler` instance. `outfile` must be an HDF5 file (inferred from file extension “.hdf5”).

class `llama.files.healpix.HEALPixSkyMapFileHandler(eventid_or_fh, rundir=None)`

Bases: `llama.files.healpix.skymap.HEALPixRepresentableFileHandler`

A FileHandler that corresponds to a file object containing HEALPix skymaps. Provides methods for reading HEALPix settings (i.e. `Nsides`, nesting method, and coordinate system) as well as reading and writing HEALPix files from `numpy` arrays. Methods do NOT assume a single skymap, and in general HEALPix-specific methods return an object for each skymap contained in the file described by this file handler in the form of a (possibly empty!) list.

get_healpix(nest=True, template_skymap=None)

Get a list of HEALPix skymaps stored in the file corresponding to this `FileHandler`.

Parameters

- `nest (bool, optional)` – If True, return skymaps in HEALPix NEST ordering; if False, return in RING ordering.

- **template_skymap** (`HEALPixSkyMap`, *optional*) – If provided, force the loaded skymaps to have the same HEALPix parameters as `template_skymap`. Overrides the `nest` parameter.

Returns `skymaps` – `HEALPixSkyMap` instances with the data contained in this file.

Return type list

`get_healpix_descriptions(skymap_indices=None)`

Get a list of descriptive strings for each HEALPix skymap contained in this file.

Parameters `skymap_indices` (*list*) – A list of tuples describing the index of each skymap used to produce this output skymap, with each entry in the tuple corresponding to a detector. In this case, the descriptive strings will be generated based on this `FileHandler` instance’s `DETECTORS` property and the indices of the `skymap_indices` in `skymap_indices`. The ordering of the returned descriptive strings will match the ordering of `skymap_indices`. If `skymap_indices` is not specified, the HEALPix skymaps and their descriptions will be read from file. Note that for single-detector skymaps, the elements of the `skymap_indices` list should just be single-element tuples corresponding to the index of each skymap.

`property num_triggers`

The number of triggers described by this file. Useful mostly for quickly determining if this trigger list is empty.

`static read_healpix_from_file(infile, nest=True)`

Read the HEALPix skymap stored in `infile` and return a list containing a single `HEALPixSkyMap` instance containing the skymap and metadata (returns a list so that other implementations of this class can optionally return many skymaps). By default, return in HEALPix NEST ordering; optionally, return in RING ordering by specifying `nest=False`. `infile` can be either a FITS image file (inferred from file extension “.fits” or “.fits.gz”) or a specially-formatted HDF5 file (inferred from file extension “.hdf5”) of the sort produced by `write_healpix_to_file`. Note that if loading from HDF5 files, `nest` is ignored.

```
>>> import tempfile, os
>>> f = tempfile.NamedTemporaryFile(suffix=' .hdf5 ', delete=False)
>>> fname = f.name
>>> f.close()
>>> skymap0 = HEALPixSkyMap(range(12))
>>> skymap1 = HEALPixSkyMap([1.] * 12)
>>> HEALPixSkyMapFileHandler.write_healpix_to_file(
...     [skymap0, skymap1],
...     fname
... )
>>> skymaps = HEALPixSkyMapFileHandler.read_healpix_from_file(
...     fname
... )
>>> skymaps[0] == skymap0
True
>>> skymaps[1] == skymap1
True
```

`write_healpix(skymaps, descriptions=None)`

Write skymap, which should be a `HEALPixSkyMap` instance or a list containing a *single* `HEALPixSkyMap` instance, to the full path specified by this `FileHandler` instance. `outfile` can be either a FITS image file (inferred from file extension “.fits” or “.fits.gz”) or a specially-formatted HDF5 file (inferred from file extension “.hdf5”).

`static write_healpix_to_file(skymaps, outfile, descriptions=None)`

Write skymaps to `outfile`.

Parameters

- **skymaps** (`HEALPixSkyMap or list`) – HEALPixSkyMap instance or list thereof to be written to file.
- **outfile** (`str`) – Path to write skymap to. File extension must conform to either a FITS image file (inferred from file extension “.fits” or “.fits.gz”) or a specially-formatted HDF5 file (inferred from file extension “.hdf5”).
- **descriptions** (`list, optional`) – A list of descriptive strings, each corresponding to the skymap in `skymaps` of the same index.

```
class llama.files.healpix.HEALPixSkyMapStats(eventid_or_fh, rundir=None)
Bases: llama.filehandler.JSONFile
```

Load a skymap from a HEALPixRepresentableFileHandler instance and generate stats on it, e.g. the likelihood integral for a HEALPixSkyMapAnalysis instance, and store them in JSON format. By default, only has one skymap file as a dependency and stores the integral of that skymap as `integrated_likelihood`.

```
classmethod set_class_attributes(subclass)
```

See `FileHandler.set_class_attributes`; this method sets programatically-generated filenames for this class.

```
class llama.files.healpix.LvcHEALPixSkyMapFileHandler(eventid_or_fh, rundir=None)
Bases: llama.files.healpix.HEALPixSkyMapFileHandler
```

A HEALPixSkyMapFileHandler using `astropy.table.Table` to read the input file (in order to accomodate the NUNIQ pixel ordering/multi-resolution skymaps).

```
static read_healpix_from_file(infile, nest=True)
```

Read the HEALPix skymap stored in `infile` and return a list containing a single HEALPixSkyMap instance containing the skymap and metadata (returns a list so that other implementations of this class can optionally return many skymaps). By default, return in HEALPix NEST ordering; optionally, return in RING ordering by specifying `nest=False`. `infile` can be either a FITS image file (inferred from file extension “.fits” or “.fits.gz”) or a specially-formatted HDF5 file (inferred from file extension “.hdf5”) of the sort produced by `write_healpix_to_file`. Note that if loading from HDF5 files, `nest` is ignored.

```
>>> import tempfile, os
>>> f = tempfile.NamedTemporaryFile(suffix=' .hdf5 ', delete=False)
>>> fname = f.name
>>> f.close()
>>> skymap0 = HEALPixSkyMap(range(12))
>>> skymap1 = HEALPixSkyMap([1.] * 12)
>>> HEALPixSkyMapFileHandler.write_healpix_to_file(
...     [skymap0, skymap1],
...     fname
... )
>>> skymaps = HEALPixSkyMapFileHandler.read_healpix_from_file(
...     fname
... )
>>> skymaps[0] == skymap0
True
>>> skymaps[1] == skymap1
True
```

```
llama.files.healpix.healpix_skymap_analysis_factory(dependencies, bases=(<class
    'llama.files.healpix.HEALPixSkyMapAnalysis'>,
    ), make_plots=False, make_stats=False,
    noclobber=False)
```

Programmatically create analysis classes based on input skymaps and analysis algorithm. **New classes are saved to the calling namespace** (consider this a shortcut macro for making `FileHandler` subclasses programatically).

Parameters

- **dependencies** (*list*) – `FileHandler` subclasses that implement the `get_healpix` method and can thus be used to run an analysis using some `HEALPixSkyMapAnalysis` implementation’s `analysis_kernel`.
- **bases** (*tuple*) – Base classes for the class being generated in the form of a tuple, e.g. `(base1, base2, ...)`
- **make_plots** (*bool, optional*) – A boolean specifying whether `HEALPixPlots` class implementations should be created to accompany each `HEALPixSkyMapAnalysis` class generated.
- **make_stats** (*bool, optional*) – A boolean specifying whether `HEALPixSkyMapStats` implementations should be created to accompany each `HEALPixSkyMapAnalysis` class generated.
- **noclobber** (*bool, optional*) – If true, don’t overwrite existing variable names; instead, throw a `ValueError`.

Returns `new_classes` – The new classes generated, where each key is the name of the new class and each value is the new class itself.

Return type

```
llama.files.healpix.pixel_radii_rad()
```

Get the maximum pixel radii in radians for each value of `k`, where `nside = 2**k`.

```
llama.files.healpix.scale2nside(scale, degrees=True)
```

Find the correct HEALPix scale NSIDE for a given feature resolution. For example, to capture features of size 0.2 degrees across, you want a HEALPix skymap with NSIDE large enough so that the max inter-pixel distance is less than 0.1 degrees. Note that HEALPix skymaps cannot have `NSIDE > 2**30`; if higher resolution is required, a `ValueError` will be raised.

Parameters

- **scale** (*int*) – The angular size of the smallest feature you need to capture (in the specified units).
- **degrees** (*bool, optional*) – If true, the feature size is assumed to be in degrees. Otherwise, it is taken to be in radians.

Returns `nside` – The minimum NSIDE parameter required to capture features with scale larger than `scale`. Will always be a power of 2 less than or equal to `2**30`.

Return type

Raises `ValueError` – If the feature scale is too small to represent with a HEALPix map.

```
llama.files.healpix.subpixels(indices, inputnside, nside, nest=True)
```

Take a list of indices into a HEALPix skymap with `NSIDE=inputnside` and return a list of indices into a HEALPix skymap with `NSIDE=nside` where the returned list of indices correspond to subpixels of the pixels indexed from the original skymap size. Useful for getting higher-resolution subpixels for a list of pixels of interest.

Returns subindices – Indices for a HEALPix skymap with `NSIDE=nside` that correspond to subpixels from the input pixel indices, sorted in ascending order.

Return type array

Parameters

- **indices** (array) – Indices for a HEALPix skymap with `NSIDE=inputnside`. Repeated indices are ignored.
- **inputnside** (int) – HEALPix NSIDE (i.e. resolution) for the input pixels.
- **nside** (int) – NSIDE for the output pixels (must be greater than `inputnside`)
- **nest** (bool, optional) – If true, input and output skymaps are interpreted as having NEST ordering. If false, they are interpreted as having RING ordering.

Raises ValueError – If `nside` or `inputnside` are not valid NSIDE HEALPix parameters (i.e. not equal to 2^{**k} for some integer k between 0 and 29), if `inputnside` is greater than `nside`, or if the `indices` are invalid.

28.2.1 llama.files.healpix.plotters module

More complex plotting commands for joint events in more human-readable formats, e.g. the GW skymap plots from O2 with their overlaid crosses marking neutrino positions.

```
class llama.files.healpix.plotters.JointSkymapScatter(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler, llama.files.healpix.skymap.MollviewMixin, llama.
filehandler.mixins.FileExtensionMixin
```

Plot a base skymap but with a scatterplot added on top. This is a bit different from joint skymaps that contain information on pixelwise likelihood products since it is only meant to be a human-readable representation.

The following class attributes must be defined in subclasses, either manually or programmatically (see: `FileHandler.set_class_attributes` and its implementations in subclasses).

A dictionary mapping `TriggerList` and `HEALPixPSF` subclass instances (i.e. pointlike-source lists) to a dict of style properties that can be fed to `matplotlib.pyplot.scatterplot` as kwargs. In particular, you MUST define `marker` and `color` in order to distinguish between different types of sources.

The `HEALPixRepresentableFileHandler` containing skymaps that should be used as the continuous background (against which point-sources will be plotted).

`BACKGROUND_SKYMAP = None`

`POINT_SOURCES = None`

`mollview(**kwargs)`

Return a Mollweide projection of this skymap as a `matplotlib` figure. Remaining `**kwargs` are passed to `healpy.mollview`.

`classmethod set_class_attributes(subclass)`

See `FileHandler.set_class_attributes`; this method additionally sets the `DEPENDENCIES` and `FILENAME` attributes based on `subclass.POINT_SOURCES` and `subclass.BACKGROUND_SKYMAP`.

```
llama.files.healpix.plotters.add_scatterplot(right_ascensions, declinations, marker='$\bigodot$', color='green', zorder=1000, **kwargs)
```

Add a scatterplot to the current HEALPy axis. Useful for plotting neutrinos etc.; remaining `kwargs` are passed to `healpy.projsscatter`. Returns the current figure.

```
llama.files.healpix.plotters.default_cmap()
```

Get the `matplotlib` colormap.

```
llama.files.healpix.plotters.outline_effect()
```

Get a `matplotlib.patheffects.withStroke` effect that outlines text nicely to improve plot readability.

```
llama.files.healpix.plotters.plot_healpix(skymap, title=None, unit='Probability Density',
                                            central_longitude=180, central_latitude=0,
                                            rotation_about_center=0, dpi=200, cbar=None, cmap=None,
                                            alpha=None, **kwargs)
```

Plot a healpix skymap in a LLAMA-specific style. Works on skymap objects with a HEALpy-esque `mollview` method. Returns a figure object for this skymap plot. No colorbar by default. Extra kwargs are passed to `skymap.mollview`. Define the opacity of the skymap with `alpha` (must be in [0,1]). In particular, use `hold=True` to use the current axis for this plot.

28.2.2 `llama.files.healpix.psf` module

Functions for working with point sources, applying point spread functions (PSFs) thereto, and making those PSFs compatible with other HEALPix skymaps.

class `llama.files.healpix.psf.Psf`

Bases: `abc.ABC`

A point spread function that can be applied to a HEALPix skymap.

abstract calculate()

Calculate a Point Spread Function (PSF) over a bunch of NUNIQ HEALPix pixels.

Returns `densities` – An array of densities for this skymap. See `Psf.indices` for the corresponding pixel indices.

Return type array

property dangle

Return the angular distances of the *important* pixels in this PSF from the center of the PSF in units of [degrees]; the indices of these pixels are given by the `indices` property in NUNIQ ordering. Omitted pixels are too far from the center of the PSF to be important.

abstract property dec

The declination of the point having a spread function applied to it [degrees].

property indices

Return the indices of the *important* pixels in this PSF in NUNIQ ordering; all other pixels will be assumed to be zero. Indices are sorted in ascending order and are guaranteed to be unique.

property max_nside

Get the maximum HEALPix NSIDE scale, i.e. the scale at which `1.2*self.max_scale` (the maximum relevant radius of the PSF, times a conservative factor to account for the fact that pixels are not perfectly square) will be smaller than the average width of a pixel. At this scale, the pixel containing the point-source and its 8 neighbors will completely contain all of the relevant PSF pixels at `self.min_nside` resolution returned by `self.indices`.

abstract property max_scale

The angular size of a circle in which nearly all of the probability will be contained [deg].

property max_scale_indices

Get the 9 large pixel indices in NUNIQ ordering at the NSIDE scale defined by `self.max_nside` that will completely contain the PSF.

property min_nside

Get the minimum HEALPix NSIDE required to provide adequate resolution for this PSF. Just find the NSIDE that will give pixel spaces smaller than `self.min_scale`.

abstract property min_scale

The rough scale [deg] of the smallest feature in this PSF. A discrete grid on which the PSF is being calculated should have pixel distances no greater than this.

property pix_area

The area per-pixel in [deg**2] of the returned indices (note that they are at the same resolution, so this is a scalar).

abstract property ra

The right ascension of the point having a spread function applied to it [degrees].

skymap_product(indices, values, density=True)

Take the pixelwise product of self.calculate with another HEALPix NUNIQ-ordered skymap. Will cleverly downselect and resize the other skymap's pixels to efficiently use memory and CPU resources.

Parameters

- **indices** (array) – Indices of other skymap's HEALPix pixels in NUNIQ ordering.
- **values** (array) – Corresponding values of the other skymap's pixels.
- **density** (bool, optional) – Whether the other skymap's pixel values are some sort of density. If True (the default), the pixel values will not be modified during rescaling (since they don't depend on pixel area). Otherwise, *THE PIXEL VALUES WILL BE CONVERTED TO DENSITIES*.

Returns

- **product_indices** (array) – The HEALPix NUNIQ indices of the pixels of the product of the two skymaps.
- **product_values** (array) – The pixel values of the product of the two skymaps corresponding to the **product_indices**.

Examples

The product between a PSF and a lower-resolution density value of 1 should be the same as the original PSF:

```
>>> import numpy as np
>>> identity_inds = np.arange(4, 16) # NSIDE = 1
>>> identity_values = np.ones_like(identity_inds)
>>> psf = PsfGaussian(30, 10, 1) # use a Gaussian PSF as an example
>>> inds, values = psf.skymap_product(identity_inds, identity_values,
...                                         density=True)
>>> np.all(psf.indices == inds)
True
>>> np.all(psf.calculate() == values)
True
```

```
class llama.files.healpix.psf.PsfGaussian(ra, dec, sigma)
```

Bases: `llama.files.healpix.psf.GaussTuple`, `llama.files.healpix.psf.Psf`

A Point Spread Function (PSF) for a Gaussian distribution.

ra [float] Right Ascension of the center of the distribution [degrees]

dec [float] Declination of the center of the distribution [degrees]

sigma [float] Standard deviation of the distribution [degrees]

calculate()

Return a 2-D Gaussian PDF function to be used as a PSF (point spread function) on the sphere. Note that this Gaussian PDF *does not* take account of curvature and is accurate only for small angular spreads. Angles must be in degrees.

property max_scale

The angular size of a circle in which nearly all of the probability will be contained [deg].

property min_scale

The rough scale [deg] of the smallest feature in this PSF. A discrete grid on which the PSF is being calculated should have pixel distances no greater than this.

llama.files.healpix.psf.resolutions()

Get an array mapping from HEALPix NSIDE values to corresponding resolutions in square degrees.

28.2.3 llama.files.healpix.skymap module

Define class containing a HEALPix skymap and convenience methods for manipulating, creating, and reformatting skymaps.

class llama.files.healpix.skymap.HEALPixPSF(eventid_or_fh, rundir=None)

Bases: [llama.files.healpix.skymap.HEALPixRepresentableFileHandler](#)

A class with utility functions for generating skymaps in-memory that follow the HEALPix parameters (NSIDE, ORDERING) of the file described by a template HEALPixSkyMapFileHandler. Used to dynamically generate skymaps that are compatible with the template skymap based on some PSF (Gaussian by default). Implementations of this class should contain point-like events with some sort of point-spread function that can be used to generate skymaps.

get_healpix(nest=None, template_skymap=None)

Generate a list of HEALPixSkyMap instances from the events contained in this FileHandler.

Parameters

- **nest** (*bool, optional*) – If True, return skymaps with NESTED ordering. If False, return skymaps with RING ordering. Overrides the ordering (but not NSIDE) defined in `template_skymap_filehandler`.
- **template_skymap** ([HEALPixSkyMap](#), *optional*) – Provide a HEALPixSkyMap as a template to override the ones provided by `template_skymap_filehandler`. Also overrides `nest` (if provided).

Returns skymaps – A list of HEALPixSkyMap instances of the triggers in this FileHandler that conform to the HEALPixSkyMap parameters in this class's `template_skymap_filehandler`

Return type list**abstract source_locations()**

Returns a list of tuples of (RA, Dec, sigma) for all point-like sources, where (RA, Dec) is the central sky position and sigma is the standard deviation of the Gaussian PSF. Since this depends on the structure of data in the relevant file handler, it must be specified for each subclass.

abstract property template_skymap_filehandler

The HEALPixSkyMapFileHandler whose HEALPix parameters should be used as a template for the HEALPixSkyMap output by this FileHandler. It is assumed that this HEALPixSkyMapFileHandler only returns one skymap in its list; consequently, only the first skymap loaded by this file handler will be used. Note that the template skymap file handler is likely not a dependency of this file handler; it is only used as a default for formatting generated skymaps from this file handler's data.

class `llama.files.healpix.skymap.HEALPixRepresentableFileHandler`(`eventid_or_fh`, `rundir=None`)
Bases: `llama.filehandler.TriggerList`

A FileHandler class containing triggers that can be represented in HEALPix. Provides `get_healpix` and `get_healpix_descriptions` methods for this purpose. Implementations need not store data in a HEALPix format natively, as long as they provide a `get_healpix` method for returning valid HEALPixSkyMap instances.

abstract `get_healpix`(`nest=None`, `template_skymap=None`)

Return a list of HEALPix skymaps representing the triggers in this FileHandler.

`get_healpix_descriptions`(`skymap_indices=None`)

Get a list of descriptive strings for each HEALPix skymap contained in this file.

Parameters `skymap_indices` (`list`) – A list of tuples describing the index of each skymap used to produce this output skymap, with each entry in the tuple corresponding to a detector. In this case, the descriptive strings will be generated based on this FileHandler instance's `detector` property and the indices of the `skymap_indices` in `skymap_indices`. The ordering of the returned descriptive strings will match the ordering of `skymap_indices`. Note that for single-detector skymaps, the elements of the `skymap_indices` list should just be single-element tuples corresponding to the index of each skymap.

Returns `descriptions` – Descriptions of each HEALPix skymap in the order in which they appear in the file.

Return type `list`

class `llama.files.healpix.skymap.HEALPixSkyMap`(`pixels`, `nest=True`, `title=None`)

Bases: `llama.files.healpix.skymap.SkyMap`, `llama.files.healpix.skymap.MollviewMixin`

A HEALPix skymap object residing in memory, including methods for working with the skymap and combining skymaps with each other.

`ang2pix`(`ra`, `dec`, `degrees=True`)

Return the pixel index on this skymap at right ascensions `ra` and declinations `dec`.

`conform_skymaps`(*`args`)

Make sure all skymaps contained in `*args` have the same ordering (NESTED or RING) as the current skymap (`self`), reordering the pixels if necessary, and returning a tuple of the (possibly reordered) input skymaps.

`dangle`(`ra`, `dec`, `degrees=True`)

Get a new skymap with the same pixelization where each pixel value is the distance from that pixel to the point with right ascension `ra` and declination `dec`. By default, `ra` and `dec` are in degrees, though this can be overridden by setting `degrees=False`, in which case they will be interpreted as radians. The measured angle differences in the returned skymap will always be in degrees. Dot product formula per-pixel is:

$$\sin(\text{DEC1}) * \sin(\text{DEC2}) + \cos(\text{DEC1}) * \cos(\text{DEC2}) * \cos(\text{RA2}-\text{RA1})$$

So we get the angle by taking the arccos of this function.

Examples

We should find that the distance from any pixel to the North pole is equal to 90 minus the declination (within some small error):

```
>>> skymap = HEALPixSkyMap([1]*12)
>>> skymap.dangle(32, 90).pixels + skymap.dec - 90 < 1e-12
array([ True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True])
```

Likewise, the distance from any pixel to the South pole should be equal to 90 plus the declination:

```
>>> skymap.dangle(359, -90).pixels - skymap.dec - 90. < 1e-12
array([ True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True])
```

property dec

Declinations in degrees.

mollview(**kwargs)

Return a Mollweide projection of this skymap as a `matplotlib` figure with the correct nesting. Remaining `**kwargs` are passed to `healpy.mollview`. Use the LIGO standard rotation where 180 degrees is at the center of the map.

property nside

The NSIDE HEALPix parameter indicating how many times each base pixel has been subdivided in each of the 2 dimensions on the sphere.

classmethod nside2ang(nside, nest=True, degrees=True)

Return (RA, Dec) in degrees for the pixels corresponding to a HEALPix skymap with `NSIDE=nside` where `nest=True` (DEFAULT) indicates NEST ordering and `nest=False` indicates RING ordering. If `degrees=True` (DEFAULT), the return value will be in degrees; otherwise, radians.

Examples

```
>>> import numpy as np
>>> ra, dec = HEALPixSkyMap.nside2ang(1, nest=True, degrees=True)
>>> all(1e-13 > ra - np.array([
...     45., 135., 225., 315., 0., 90., 180., 270., 45., 135.,
...     225., 315.
... ]))
True
```

psf_gaussian(ra, dec, sigma, degrees=True, normalize=True)

Calculate a Gaussian PSF at all pixels in this skymap.

Parameters

- `ra` (`float`) – Right Ascension of the Gaussian mean.
- `dec` (`float`) – Declination of the Gaussian mean.
- `sigma` (`float`) – Standard deviation of the Gaussian PSF.
- `degrees` (`bool, optional`) – If true, input arguments (`ra, dec, sigma`) are assumed to be in degrees. Otherwise, they are assumed to be in radians.

- **normalize** (*bool, optional*) – If true, normalize the returned skymap so that the integral over area is 1 (useful for when the Gaussian is meant to represent a probability density function).

Returns skymap – A new skymap with the same pixelization where a 2-D Gaussian PSF (point spread function) centered at `(ra, dec)` with standard deviation `sigma` has been calculated for each pixel in this skymap.

Return type `HEALPixSkyMap`

property ra

Right Ascensions in degrees.

to_nest()

Return a skymap with the same pixel values and NESTED ordering.

to_ring()

Return a skymap with the same pixel values and RING ordering.

class llama.files.healpix.skymap.MollviewMixin

Bases: abc.ABC

Defines an interface for plotting Mollweide projections.

abstract mollview(kwargs)**

Return a Mollweide projection of this skymap as a `matplotlib` figure.

class llama.files.healpix.skymap.SkyMap(*pixels, ra, dec, degrees=True, area_per_pixel=None*)

Bases: abc.ABC

An array representing a skymap.

property area_per_pixel

Get the solid angle per pixel. Can be set to a scalar value or a vector with the same length as the list of pixels; in either case, the pixel areas are returned as a vector for each pixel. If not set, assumes that all pixel areas are identical and that the pixel areas sum to a full sphere.

property as_pdf

Return a skymap whose pixels represent probability/pixel (rather than probability per solid angle, i.e. a proper PDF) and putting them into units of probability per solid angle. Returns a new skymap with units of inverse solid angle. If this skymap is already a PDF in units of inverse solid angle, this operation returns the original skymap.

confidence_region_size(*percent=90*)

Calculate the size of the confidence region in square degrees. By default, finds the 90% confidence region, though the percentage can be manually specified. Does NOT normalize pixels before summing them (in case the total probability does not add to 1, e.g. if we are already dealing with a partial skymap).

abstract dangle(*ra, dec, degrees=True*)

Get a new skymap with the same pixelization where each pixel value is the distance from that pixel to the point with right ascension `ra` and declination `dec`. By default, `ra` and `dec` are in degrees, though this can be overridden by setting `degrees=False`, in which case they will be interpreted as radians. The measured angle differences in the returned skymap will always be in degrees. Dot product formula per-pixel is:

$$\sin(\text{DEC1}) * \sin(\text{DEC2}) + \cos(\text{DEC1}) * \cos(\text{DEC2}) * \cos(\text{RA2} - \text{RA1})$$

property dec

Declinations in degrees.

integrate()

If `pixels` is given in units of inverse solid angle, return the integral of the pixel values over the whole sky. If `pixels` is dimensionless, assume the pixel values are probability per-pixel and simply return the sum

of pixel values. Either way, the integral is dimensionless and equals the total probability of the SkyMap (given the stated assumptions).

map(function)

Map a function to the pixel locations of this skymap and return an identically sized and ordered skymap with the output values of that function as its pixel values. `function` must take vector arguments `ra` and `dec` and return a list of pixel values of the same length. No normalization takes place.

normalize(norm_to=1)

Scale the pixel values of this skymap so that they integrate to the value specified by `norm_to` (DEFAULT: 1). Returns a new skymap with the same units as `self.pixels` (assuming `self.integrate` is defined for `self.units` used in `self.pixels`).

property npix

property ra

Right Ascensions in degrees.

property unit

Get the units of the pixels of this skymap. If `pixels` is already an `astropy.Quantity`, simply return its `unit`. Otherwise, return `astropy.units.Unit("")` (i.e. dimensionless).

`llama.files.healpix.skymap.psf_gaussian(delta_angle, sigma, degrees=True)`

Return a 2-D Gaussian PDF function to be used as a PSF (point spread function) on the sphere. Note that this Gaussian PDF *does not* take account of curvature and is accurate only for small angular spreads.

Parameters

- **delta_angle (array_like)** – A scalar or array of angular distances from the center of the Gaussian. Must be in same units (degrees or radians) as `sigma`.
- **sigma (float)** – The standard deviation of the Gaussian. Must be in same units (degrees or radians) as `delta_angle`.
- **degrees (bool, optional)** – Whether angles are given in degrees. If False, units are assumed to be in radians.

Returns

- **densities (array_like)** – The values of the Gaussian at the distances given by `delta_angle` in units of [1/steradian] (even if inputs are given in degrees!).
- `>>> import numpy as np`
- `>>> psf_gaussian(1, 1, degrees=False) == 1/(2*np.pi) * np.exp(-1/2)`
- `True`

`llama.files.healpix.skymap.psf_gaussian_scales(sigma)`

Get the characteristic maximum and minimum scales of a Gaussian PSF for a given angle `sigma`.

Parameters sigma (int or float) – The angular standard deviation of the Gaussian PSF in degrees or radians.

Returns

- **min_scale (float)** – The scale of inter-pixel distances required to capture the features of a Gaussian with the given standard deviation. In same units as `sigma`.
- **max_scale (float)** – The diameter of a pixel mesh centered on the Gaussian mean that is large enough to capture almost all of the probability in the distribution. In same units as `sigma`.

28.2.4 `llama.files.healpix.utils` module

Utility functions used across `healpix_skymap` classes.

`llama.files.healpix.utils.check_valid_nside(nside)`

Checks whether `nside` is a valid HEALPix NSIDE value. Raises a `ValueError` if it is not.

Parameters `nside` (*int or array*) – An integer or array of integers representing HEALPix NSIDE values.

Raises ValueError – If any of the values are not valid HEALPix NSIDE values.

`llama.files.healpix.utils.check_valid_nuniq(indices)`

Checks that `indices` are valid NUNIQ indices.

Raises ValueError – If `indices` are not valid NUNIQ indices, i.e. they are not integers greater than 3.

`llama.files.healpix.utils.dangle_rad(ra, dec, mapra, mapdec)`

Get an array of angular distances in radians between the point specified in `ra`, `dec` and the points specified in `mapra`, `mapdec`. All angles, including the return value, are specified in radians.

$$\sin(\text{DEC1})\sin(\text{DEC2}) + \cos(\text{DEC1})\cos(\text{DEC2})\cos(\text{RA2}-\text{RA1})$$

So we get the angle by taking the arccos of this function.

Parameters

- `ra` (*float*) – Right ascension of the single point [radians]
- `dec` (*float*) – Declination of the single point [radians]
- `mapra` (*array*) – Right ascensions whose angular distances to the single point will be calculated; must be same length as `mapdec`. [radians]
- `mapdec` (*array*) – Declinations whose angular distances to the single point will be calculated; must be same length as `mapra`. [radians]

Returns `dangles` – Angular distances between the single point provided and the arrays of points [radians].

Return type

`llama.files.healpix.utils.downres_mask_nest(mask_nested, mask_nside, nside, mask_sorted=False)`

Get a lower-resolution set of pixel indices that fully cover the region specified in `mask_nested`. The returned indices correspond to larger pixels which might contain part of the sky that are not included in the original mask (e.g. if the larger pixel only contains one of the original pixels in `mask_nested`).

Parameters

- `mask_nested` (*array-like*) – The set of nested HEALPix pixels that you'd like to have at lower resolution.
- `mask_nside` (*int*) – The HEALPix `nside` parameter of this mask. Must be a power of 2.
- `nside` (*int*) – The `nside` value you would like for your output. Must be a power of 2.
- `mask_sorted` (*bool, optional*) – Whether the input `mask_nested` is already sorted. If it is sorted and is large, specifying this will skip a sort step, potentially saving time.

Returns `lowres_mask_nested` – The smallest sky region that can be specified at the resolution given by `nside` that contains the sky region specified in `mask_nested` at `mask_nside`. This will be roughly a factor of $(\text{nside}/\text{mask_nside})^{**2}$ smaller, but do **not** count on this ratio, since the pixels in the returned mask might not have all of their corresponding subpixels in the original `mask_nested`; in pathological cases, the returned array can be up to the same size (if every pixel

in `mask_nested` only appears in a single larger pixel in at resolution `nside`). Will be a new array independent of inputs, so feel free to mutate it.

Return type `np.array`

Raises `ValueError` – if `mask_nside` is smaller than `nside`

`llama.files.healpix.utils.nest2uniq(indices, nside)`

Return the NUNIQ pixel indices for nested `indices` with NSIDE=```nside```.

Parameters

- `indices` (`array`) – Indices of HEALPix pixels in NEST ordering.
- `nside` (`int`) – A valid NSIDE value not greater than any of the NSIDE values of the `indices`.

Returns `uniq` – An array of HEALPix pixels in NEST ordering corresponding to the input `indices` and `nside`.

Return type `array`

Raises `ValueError` – If `nside` is an invalid NSIDE value, i.e. not a power of 2.

Examples

```
>>> import numpy as np
>>> nest2uniq(np.array([284, 286, 287, 10, 8, 2, 279, 278, 285]), 8)
array([540, 542, 543, 266, 264, 258, 535, 534, 541])
```

`llama.files.healpix.utils.read_partial_skymap(infile, mask_nested, mask_nside, mask_sorted=False, memmap=True)`

Read in pixels from a FITS skymap (or a gzip-compressed FITS skymap) that lie in a specific sky region. Attempts to minimize memory usage by memory-mapping pixels in the input file and only loading those specified in `mask_nested`.

Parameters

- `infile` (`str`) – A FITS HEALPix skymap (optionally compressed; see note under `memmap`)
- `mask_nested` (`bool, optional`) – HEALPix pixel indices in `nested` ordering specifying the region of the skymap that should be loaded
- `mask_nside` (`int`) – The resolution of the pixel mask
- `mask_nested` – Whether `mask_nested` is already sorted. You can specify `mask_sorted=True`, potentially saving some sorting time
- `memmap` (`bool, optional`) – If True, use memory mapping when reading in files. This can VASTLY reduce memory required for high-resolution skymaps. If `infile` is gzip-compressed and `memmap` is True, then `infile` will be decompressed to a temporary file and data will be read from it (necessary to constrain memory usage); for high-resolution skymaps, this can require the availability of several gigabytes of tempfile storage.

Returns `partial_skymap` – A partial skymap table in `nested` ordering. Has two columns: INDICES and PROB. If the resolution of the original HEALPix skymap file is lower than that of the mask, then any pixels overlapping with those in `mask_nested` will be used; this might result in a larger portion of the skymap being used than that specified in `mask_nested`. The resolution of this skymap will be the resolution of the smallest pixel loaded from the input file (in the case of `ring` or `nested` ordering, this is just the resolution of the input skymap).

Return type astropy.table.Table

```
llama.files.healpix.utils.sky_area()  
Get the area of the entire sky as an Astropy.units.Quantity.
```

```
llama.files.healpix.utils.sky_area_deg()  
Get the area of the entire sky in square degrees.
```

```
llama.files.healpix.utils.uniq2nest_and_nside(indices)
```

Parameters `indices` (array) – A scalar or numpy array of NUNIQ indices

Returns

- `nest_indices` (array) – The indices in nest ordering at their respective resolutions
- `nside` (array) – The resolution expressed as NSIDE of the indices given in `nest_indices`

Examples

```
>>> import numpy as np  
>>> nuniq = np.array([540, 542, 543, 266, 264, 258, 535, 534, 541])  
>>> uniq2nest_and_nside(nuniq)  
(array([284, 286, 287, 10, 8, 2, 279, 278, 285]), array([8, 8, 8, 8, 8, 8, 8,  
     8, 8]))
```

Confirm that this is the inverse of `nest2uniq`: >>> all(nest2uniq(*uniq2nest_and_nside(nuniq)) == nuniq) True

```
llama.files.healpix.utils.uniq2nside(indices)  
Get the NSIDE value of the given NUNIQ-ordered indices.
```

Raises ValueError – If `indices` are not valid NUNIQ indices, i.e. they are not integers greater than 3.

```
llama.files.healpix.utils.uniq2uniq_lowres(indices, nside)
```

Find the indices at resolutions `nside` (each of which MUST be lower resolution than that of the corresponding pixel in `indices`) which contain the higher resolution pixels described by the corresponding `indices`.

Parameters

- `indices` (array) – Indices of HEALPix pixels in NUNIQ ordering.
- `nside` (array) – A valid NSIDE value not greater than any of the NSIDE values of the indices. Each of these corresponds to an index in `indices`.

Returns `lowres_indices` – Indices of HEALPix pixels in NUNIQ ordering with resolution `nside` which contain the corresponding pixels in `indices`.

Return type array

Raises ValueError – If the NSIDE resolution of any index in `indices` is smaller than `nside`, or if `nside` is an invalid NSIDE value, i.e. not a power of 2.

Examples

Take a few NUNIQ pixels at NSIDE=16 and an NUNIQ pixel at NSIDE=1024 and find which NUNIQ pixels they correspond to at NSIDE=8. The first four should all correspond to the same pixel at NSIDE=8.

```
>>> import numpy as np
>>> indices = np.array([1024, 1025, 1026, 11295603])
>>> uniq2uniq_lowres(indices, 8)
array([256, 256, 256, 689])
```

`llama.files.healpix.utils.uniq_intersection(indices1, indices2)`

Downselect the pixel indices given in `indices1` to the set that overlaps with pixels in `indices2` and return pairs of indices into both of the input index lists showing which pixel in `indices2` each downselected pixel from `indices1` overlaps with. Use this to get rid of pixels outside of a given region, or alternatively use it as part of a calculation involving two multi-resolution skymaps whose pixel sizes are non-identical. Should have stable O(len(`indices1`)*len(`indices2`)) performance, so make sure that your index lists are not both huge.

Parameters

- `indices1` (`array`) – Indices of HEALPix pixels in NUNIQ ordering.
- `indices2` (`array`) – Indices of HEALPix pixels in NUNIQ ordering.

Returns

- `indices_into_indices1` (`array`) – Indices *into* `indices1` that overlap with `indices2`.
- `indices_into_indices2` (`array`) – Corresponding indices *into* `indices2` that overlap with `indices1`.

Examples

Some pixels with NSIDE of 16, 32, and 64, respectively: >>> import numpy as np >>> indices1 = np.array([1024, 4100, 44096])

Pixels at NSIDE = 32 that overlap with only the first and last pixels of `indices1`: >>> indices2 = np.array([4096, 4097, 11024])

We should see correspondence between index 0 of `indices1` and indices 0, 1 of `indices2`; and correspondence between index 2 of `indices1` and index 2 of `indices2`: >>> uniq_intersection(indices1, indices2) (array([0, 0, 2]), array([0, 1, 2]))

`llama.files.healpix.utils.uniq_rasterize(indices, values, density=True, nside=None)`

Increase resolution of an NUNIQ HEALPix skymap and return NUNIQ indices and values at the same higher resolution (same NSIDE). Doesn't do any interpolation; just used to make calculations easier.

Parameters

- `indices` (`array`) – Non-overlapping HEALPix NUNIQ indices into the skymap.
- `values` (`array`) – Pixel values corresponding to the `indices`.
- `density` (`bool, optional`) – Whether the pixel values are some sort of density. If True (the default), the pixel values will not be modified (since they don't depend on pixel area). Otherwise, the pixel value will be split evenly between sub-pixels.
- `nside` (`int, optional`) – The NSIDE value to convert the skymap to. If not provided, the skymap will be converted to the largest NSIDE of the provided pixels.

Returns

- **raster_indices** (*array*) – A unique flat array of HEALPix NUNIQ indices into the rasterized skymap, sorted in ascending order.
 - **raster_values** (*array*) – A flat array of pixel values of the rasterized skymap corresponding to `raster_indices`.

Raises `ValueError` – If the given `nside` value is either an invalid NSIDE value or if it is lower than the highest NSIDE value calculated from `indices`, or if the given `indices` are overlapping.

Examples

Some example inputs with NSIDE of 16, 32, and 64, respectively: >>> import numpy as np >>> indices = np.array([1024, 4100, 44096]) >>> values = np.ones_like(indices)

Rasterize to the max resolution of NSIDE=64, assuming values represent a density: >>> uniq_rasterize(indices, values) (array([16384, 16385, 16386, 16387, 16388, 16389, 16390, 16391, 16392,

Rasterize just the first two indices, this time explicitly to NSIDE=64 (vs. the default max NSIDE, 32 in this case), explicitly treating the values NOT as a density (and therefore splitting them amongst subpixels): >>> uniq_rasterize(indices[0:2], values[0:2], density=False, nside=64) (array([16384, 16385, 16386, 16387, 16388, 16389, 16390, 16391, 16392,

`llama.files.healpix.utils.upres_nest(mask_nested, mask_nside, nside, mask_sorted=False)`
Get nested indices for a skymap mask at a higher resolution. Output will be sorted.

Parameters

- **mask_nested** (*array-like*) – The set of nested HEALPix pixels that you’d like to have at higher resolution.
 - **mask_nside** (*int*) – The HEALPix nside parameter of this mask. Must be a power of 2.
 - **nside** (*int*) – The nside value you would like for your output. Must be a power of 2.
 - **mask_sorted** (*bool, optional*) – Whether the input `mask_nested` is already sorted. If it is sorted and is large, specifying this will skip a sort step, potentially saving time.

Returns `highres_mask_nested` – The same sky region specified in `mask_nested` at `mask_nside` but at the resolution specified in `nside`. This will be a factor of $(\text{nside}/\text{mask_nside})^{**2}$ larger. Will be a new array independent of inputs, so feel free to mutate it.

Return type np.array

Raises ValueError – if `mask_nside` is larger than `nside`

28.3 llama.files.i3 package

FileHandler classes and utilities associated with retrieving GFU neutrino data from IceCube (see: `llama.detectors.IceCube`).

```
class llama.files.i3.IceCubeNeutrinoList(eventid_or_fh, rundir=None)
Bases: llama.filehandler.JSONFile, llama.files.healpix.skymap.HEALPixPSF
```

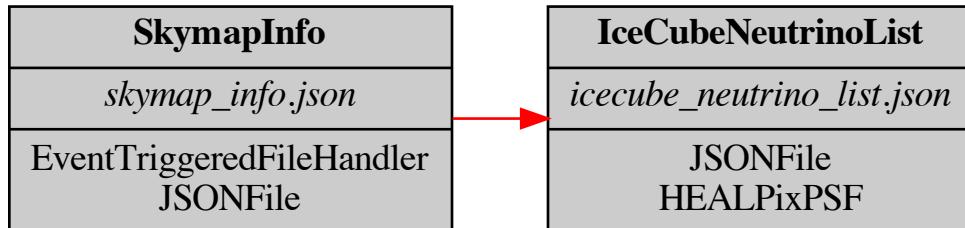


Fig. 46: Required input files for `llama.files.i3.json.IceCubeNeutrinoList` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.json.IceCubeNeutrinoList` to be generated.

Takes a list of dictionaries describing neutrinos and saves it to a text file. Validates the neutrino data to make sure each neutrino contains (minimally) the following properties:

- mjd
- zenith
- azimuth
- sigma
- energy
- type (one the values found in `NEUTRINO_TYPES`)

If this file is generated within the temporal search window for neutrinos, it is possible not all neutrinos needed for the final result will be available. In this case, the file will automatically become `obsolete` and will be regenerated after the window has elapsed.

Neutrinos from IceCube can be delayed due to communications issues; to combat this, when neutrinos are pulled from the online data stream after the temporal search window has elapsed, a check will be run to see if the next neutrino after the end of the search window is yet available. If it is not available, it is assumed that not all neutrinos needed for the search have arrived from south pole, and a `GenerationError` will be raised, allowing the pipeline to cooldown and retry the neutrino pull later.

Neutrinos pulled from before `MJD_TOPIC_SWITCH_17_TO_BLANK` will be automatically pulled from local GFU archives (stored online on LLAMA S3 and cached locally after use in `llama.utils.CACHEDIR` in `*.npy` format). Neutrinos from after this date will be pulled from the online GFU stream.

```
DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
DETECTORS = (Detector(name='IceCube', abbrev='i3', fullname='IceCube',
url='http://wiki.icecube.wisc.edu', summary='IceCube', description='',
citations=ImmutableDict({})),)
FILENAME = 'icecube_neutrino_list.json'
MANIFEST_TYPES = (<class 'llama.files.i3.json.IceCubeNeutrinoList'>,)
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
```

```
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})
```

```
downselect(ra=None, dec=None, ntype=None)
```

Downselect the neutrino list to match given criteria. Save the old neutrino list with the original filename prefixed by a timestamp and replace it with the downselected list.

Parameters

- **ra** (*list*) – Right-ascension intervals (in degrees) that the neutrinos must fall into, e.g. [(21, 41), (92, 102)]
- **dec** (*list*) – Declination intervals (in degrees) that the neutrinos must fall into, e.g. [(-3, 20)]
- **ntype** (*str*) – A type that the neutrino must match, e.g. 'observation' for real neutrinos.

```
end_of_neutrino_window()
```

The GPS time at which we can assume that all neutrinos are in.

```
generated_before_all_neutrinos()
```

Whether this file was generated before all IceCube neutrinos in the time window were available. Returns False if the generation time cannot be determined.

```
is_obsolete(checked=None, **kwargs)
```

IceCubeNeutrinoList files can be fetched immediately after a GW event becomes available. However, we want to eventually run our analysis on all neutrinos in a certain time window around an event. We therefore want to re-fetch the neutrinos in that time window after the time window has passed (allowing a safety factor for the IceCube GFU API to process new neutrino triggers). To do this, we mark the IceCubeNeutrinoList as obsolete if the current time is after the time window + safety margin has elapsed and the neutrino list was generated before that time had passed (meaning there might have been new neutrinos saved that are useful for the analysis after the file was originally generated). Standard FileHandler obsolescence criteria are also used to determine obsolescence through a call to super.

```
property neutrinos
```

Return a list containing a Neutrino object for each neutrino; more convenient to work with when writing formulae than dealing directly with the neutrino dictionaries.

```
property num_triggers
```

The number of triggers described by this file. Useful mostly for quickly determining if this trigger list is empty.

```
source_locations()
```

Returns a list of tuples of (RA, Dec, sigma) for all point-like sources, where (RA, Dec) is the central sky position and sigma is the standard deviation of the Gaussian PSF. Since this depends on the structure of data in the relevant file handler, it must be specified for each subclass.

```
property template_skymap_filehandler
```

The HEALPixSkyMapFileHandler whose HEALPix parameters should be used as a template for the HEALPixSkyMap output by this FileHandler. It is assumed that this HEALPixSkyMapFileHandler only returns one skymap in its list; consequently, only the first skymap loaded by this file handler will be used. Note that the template skymap file handler is likely not a dependency of this file handler; it is only used as a default for formatting generated skymaps from this file handler's data.

```
class llama.files.i3.IceCubeNeutrinoListCoincTxt(eventid_or_fh, rundir=None)
```

Bases: *llama.filehandler.FileHandler*

A space-delimited, table-formatted list of neutrinos meant for human consumption in GCN Circulars. Contains coincidence significance measures in addition to the base neutrino properties.

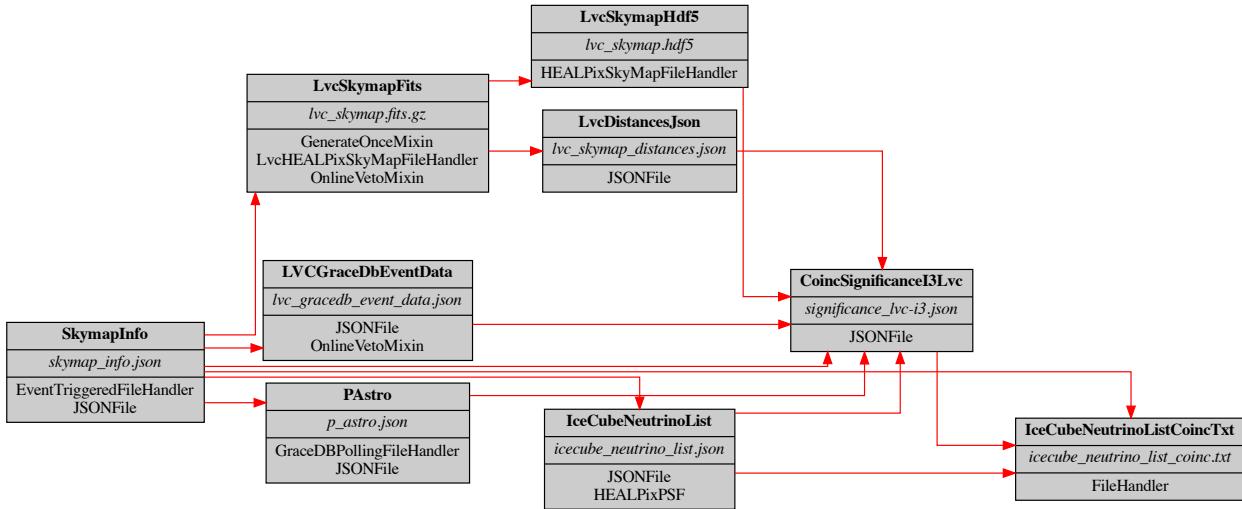


Fig. 47: Required input files for `llama.files.i3.txt.IceCubeNeutrinoListCoincTxt` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.txt.IceCubeNeutrinoListCoincTxt` to be generated.

```

DEPENDENCIES = (<class 'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)

FILENAME = 'icecube_neutrino_list_coinc.txt'

MANIFEST_TYPES = (<class 'llama.files.i3.txt.IceCubeNeutrinoListCoincTxt'>)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)
    
```

```

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{})), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}), <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc': ImmutableDict({<class
'llama.files.gracedb.LVCGraceDbEventData': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{})), <class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{})), <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}))), <class
'llama.files.lvc_skymap.LvcDistancesJson': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}))), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}), <class
'llama.files.gracedb.PAstro': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}))}))})

```

class `llama.files.i3.IceCubeNeutrinoListTex(eventid_or_fh, rundir=None)`

Bases: `llama.filehandler.FileHandler`

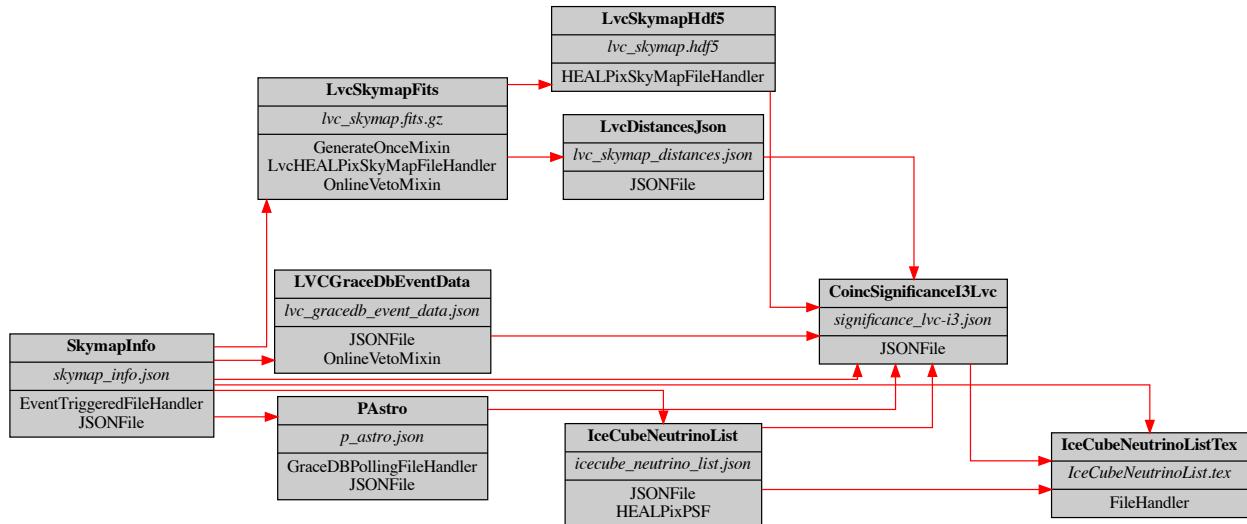


Fig. 48: Required input files for `llama.files.i3.tex.IceCubeNeutrinoListTex` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.tex.IceCubeNeutrinoListTex` to be generated.

A list of neutrinos and their reconstructed properties as well as the significance of the joint event) in LaTeX format suitable for use in papers and coincident skymap summary plots.

```

DEPENDENCIES = (<class 'llama.files.i3.json.IceCubeNeutrinoList', <class
'	llama.files.skymap_info.SkymapInfo', <class
'	llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)

FILENAME = 'IceCubeNeutrinoList.tex'

MANIFEST_TYPES = (<class 'llama.files.i3.tex.IceCubeNeutrinoListTex',)

```

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{})), <class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}), <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>: ImmutableDict({<class
'llama.files.gracedb.LVCGraceDbEventData'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{})), <class
'llama.files.i3.json.IceCubeNeutrinoList'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{})), <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>: ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}))), <class
'llama.files.lvc_skymap.LvcDistancesJson'>: ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}))), <class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}), <class
'llama.files.gracedb.PAstro'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{})))))

class llama.files.i3.IceCubeNeutrinoListTxt(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler

```

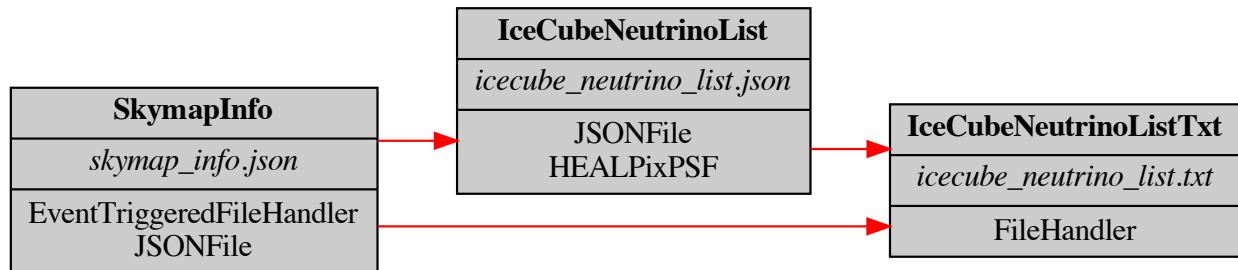


Fig. 49: Required input files for `llama.files.i3.txt.IceCubeNeutrinoListTxt` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.txt.IceCubeNeutrinoListTxt` to be generated.

A space-delimited, table-formatted list of neutrinos formatted for human consumption in GCN Circulars.

```

DEPENDENCIES = (<class 'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'icecube_neutrino_list.txt'

MANIFEST_TYPES = (<class 'llama.files.i3.txt.IceCubeNeutrinoListTxt'>)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)

```

```
UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}})})
```

```
class llama.files.i3.Neutrino(**kwargs)
```

Bases: object

A class for holding neutrino trigger data in easily accessible form. Also provides convenience methods for coordinate conversions of underlying data. Good for plugging values into formulae.

All properties listed are REQUIRED and must be passed as kwargs.

mjd [(independent)] Modified Julian Date in UTC epoch at which the neutrino candidate was detected.

zenith [(independent)] Zenith angle of neutrino source direction, measured from IceCube's location at South Pole.

azimuth [(independent)] Azimuthal angle of neutrino source direction, measured from IceCube's location at South Pole.

energy [(independent)] Reconstructed energy of the neutrino candidate in GeV.

sigma [(independent)] Effective uncertainty radius in degrees.

gps [(dependent)] GPS time, i.e. number of GPS seconds since start of GPS epoch, when the neutrino candidate was detected.

ra [(dependent)] Right Ascension of neutrino candidate in degrees.

dec [(dependent)] Declination of neutrino candidate in degrees.

psf [(dependent)] A Psf object for this neutrino that can be used to take products of this neutrino's localization PSF with other skymap objects.

property azimuth

(independent) Azimuthal angle of neutrino source direction, measured from IceCube's location at South Pole.

property dec

(dependent) Declination of neutrino candidate in degrees.

property energy

(independent) Reconstructed energy of the neutrino candidate in GeV.

property gps

(dependent) GPS time, i.e. number of GPS seconds since start of GPS epoch, when the neutrino candidate was detected.

property mjd

(independent) Modified Julian Date in UTC epoch at which the neutrino candidate was detected.

property psf

(dependent) A Psf object for this neutrino that can be used to take products of this neutrino's localization PSF with other skymap objects.

property ra

(dependent) Right Ascension of neutrino candidate in degrees.

property sigma

(independent) Effective uncertainty radius in degrees.

```
property zenith
  (independent) Zenith angle of neutrino source direction, measured from
    IceCube's location at South Pole.
```

28.3.1 llama.files.i3.json module

Methods and FileHandler classes associated with fetching IceCube neutrino triggers.

```
class llama.files.i3.json.IceCubeNeutrinoList(eventid_or_fh, rundir=None)
  Bases: llama.filehandler.JSONFile, llama.files.healpix.skymap.HEALPixPSF
```

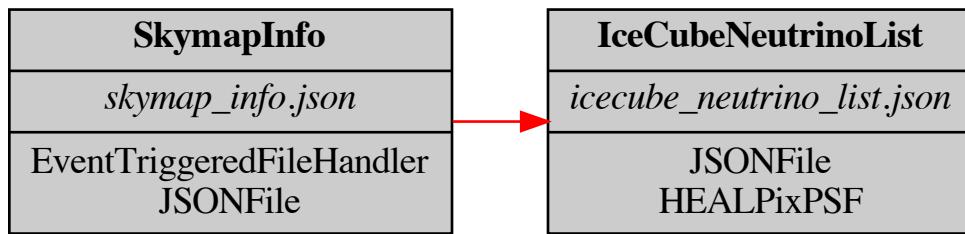


Fig. 50: Required input files for `llama.files.i3.json.IceCubeNeutrinoList` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.json.IceCubeNeutrinoList` to be generated.

Takes a list of dictionaries describing neutrinos and saves it to a text file. Validates the neutrino data to make sure each neutrino contains (minimally) the following properties:

- mjd
- zenith
- azimuth
- sigma
- energy
- type (one the values found in `NEUTRINO_TYPES`)

If this file is generated within the temporal search window for neutrinos, it is possible not all neutrinos needed for the final result will be available. In this case, the file will automatically become `obsolete` and will be regenerated after the window has elapsed.

Neutrinos from IceCube can be delayed due to communications issues; to combat this, when neutrinos are pulled from the online data stream after the temporal search window has elapsed, a check will be run to see if the next neutrino after the end of the search window is yet available. If it is not available, it is assumed that not all neutrinos needed for the search have arrived from south pole, and a `GenerationError` will be raised, allowing the pipeline to cooldown and retry the neutrino pull later.

Neutrinos pulled from before `MJD_TOPIC_SWITCH_17_TO_BLANK` will be automatically pulled from local GFU archives (stored online on LLAMA S3 and cached locally after use in `llama.utils.CACHEDIR` in `*.npy` format). Neutrinos from after this date will be pulled from the online GFU stream.

```
DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
DETECTORS = (Detector(name='IceCube', abbrev='i3', fullname='IceCube',
url='http://wiki.icecube.wisc.edu', summary='IceCube', description='',
citations=ImmutableDict({})),)
```

```
FILENAME = 'icecube_neutrino_list.json'
MANIFEST_TYPES = (<class 'llama.files.i3.json.IceCubeNeutrinoList'>,)
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})
downselect(ra=None, dec=None, ntype=None)
```

Downselect the neutrino list to match given criteria. Save the old neutrino list with the original filename prefixed by a timestamp and replace it with the downselected list.

Parameters

- **ra** (*list*) – Right-ascension intervals (in degrees) that the neutrinos must fall into, e.g. [(21, 41), (92, 102)]
- **dec** (*list*) – Declination intervals (in degrees) that the neutrinos must fall into, e.g. [(-3, 20)]
- **ntype** (*str*) – A type that the neutrino must match, e.g. 'observation' for real neutrinos.

end_of_neutrino_window()

The GPS time at which we can assume that all neutrinos are in.

generated_before_all_neutrinos()

Whether this file was generated before all IceCube neutrinos in the time window were available. Returns `False` if the generation time cannot be determined.

is_obsolete(*checked=None, **kwargs*)

`IceCubeNeutrinoList` files can be fetched immediately after a GW event becomes available. However, we want to eventually run our analysis on all neutrinos in a certain time window around an event. We therefore want to re-fetch the neutrinos in that time window after the time window has passed (allowing a safety factor for the IceCube GFU API to process new neutrino triggers). To do this, we mark the `IceCubeNeutrinoList` as obsolete if the current time is after the time window + safety margin has elapsed and the neutrino list was generated before that time had passed (meaning there might have been new neutrinos saved that are useful for the analysis after the file was originally generated). Standard `FileHandler` obsolescence criteria are also used to determine obsolescence through a call to `super`.

property neutrinos

Return a list containing a `Neutrino` object for each neutrino; more convenient to work with when writing formulae than dealing directly with the neutrino dictionaries.

property num_triggers

The number of triggers described by this file. Useful mostly for quickly determining if this trigger list is empty.

source_locations()

Returns a list of tuples of (RA, Dec, sigma) for all point-like sources, where (RA, Dec) is the central sky position and sigma is the standard deviation of the Gaussian PSF. Since this depends on the structure of data in the relevant file handler, it must be specified for each subclass.

property template_skymap_filehandler

The `HEALPixSkyMapFileHandler` whose `HEALPix` parameters should be used as a template for the `HEALPixSkyMap` output by this `FileHandler`. It is assumed that this `HEALPixSkyMapFileHandler` only returns one skymap in its list; consequently, only the first skymap loaded by this file handler will be used. Note that the template skymap file handler is likely not a dependency of this file handler; it is only used as a default for formatting generated skymaps from this file handler's data.

```
class llama.files.i3.json.RctGdbIceCubeNeutrinoList(eventid_or_fh, rundir=None)
Bases: llama.files.gracedb.GraceDBReceipt
```

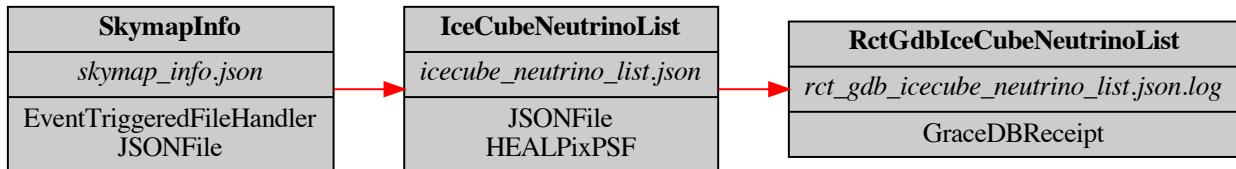


Fig. 51: Required input files for `llama.files.i3.json.RctGdbIceCubeNeutrinoList` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.json.RctGdbIceCubeNeutrinoList` to be generated.

```

DEPENDENCIES = (<class 'llama.files.i3.json.IceCubeNeutrinoList'>,)
FILENAME = 'rct_gdb_icecube_neutrino_list.json.log'
MANIFEST_TYPES = (<class 'llama.files.i3.json.RctGdbIceCubeNeutrinoList'>,)
UPLOAD
    alias of llama.files.i3.json.IceCubeNeutrinoList
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.i3.json.IceCubeNeutrinoList'>})
property log_message
    GraceDB upload log message for <class 'llama.files.i3.json.IceCubeNeutrinoList'>
class llama.files.i3.json.RctSlkLmaIceCubeNeutrinoList(eventid_or_fh, rundir=None)
    Bases: llama.files.slack.SlackReceiptLlama
  
```

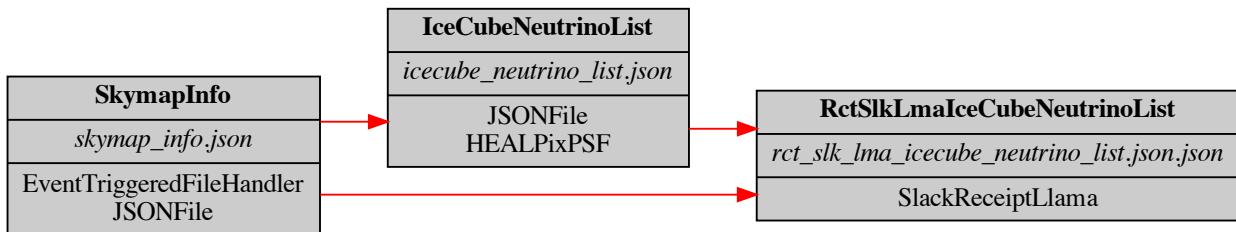


Fig. 52: Required input files for `llama.files.i3.json.RctSlkLmaIceCubeNeutrinoList` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.json.RctSlkLmaIceCubeNeutrinoList` to be generated.

```

DEPENDENCIES = (<class 'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.skymap_info.SkymapInfo'>)
FILENAME = 'rct_slk_lma_icecube_neutrino_list.json.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'
MANIFEST_TYPES = (<class 'llama.files.i3.json.RctSlkLmaIceCubeNeutrinoList'>,)
UPLOAD
    alias of llama.files.i3.json.IceCubeNeutrinoList
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)
  
```

```
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.i3.json.IceCubeNeutrinoList'>})
class_veto = ()

llama.files.i3.json.blinder(neutrino, unblinded_times=())
Randomize neutrino azimuth (and hence right ascension) for neutrinos triggered at times that are not explicitly approved for unblinded analysis.
```

Parameters

- **neutrino** (*dict*) – A neutrino dictionary in the format returned by `format_neutrinos`.
- **unblinded_times** (*array_like, optional*) – An iterable of [start, end] times (specified in UTC MJD format) in which we are approved by IceCube to pull unblinded neutrinos for our analysis. If not provided, it is assumed that ALL neutrinos should be blinded (DEFAULT: `tuple()`).

Returns `blinded` – The same neutrino provided as a parameter with the azimuth (and hence right ascension) with RA scrambled if necessary. If the neutrino's MJD falls inside one of these intervals, it will be returned unchanged. If it falls outside all of these intervals, its azimuth will be set to a random value. This is a fresh copy of the neutrino; the neutrino provided as input is *not* modified in place.

Return type dict

Examples

```
>>> neutrino_time = 57990
>>> unblinded_times = [[neutrino_time-1, neutrino_time+1]]
>>> neutrino = {
...     "mjd": neutrino_time,
...     "zenith": 90,
...     "sigma": 0.3,
...     "energy": 5000,
...     "azimuth": 270,
...     "bdt_up": 1,
... }
>>> blind = blinder(neutrino, unblinded_times[])
>>> blind['azimuth'] == neutrino['azimuth']
False
>>> unblind = blinder(neutrino, unblinded_times=unblinded_times)
>>> unblind['azimuth'] == neutrino['azimuth']
True
```

`llama.files.i3.json.format_neutrinos(neutrinos)`

Read in ICECUBE neutrinos as returned by `realtime_tools.live.get_events` and put them in the format that LLAMA expects.

`llama.files.i3.json.get_archive(mjd_start, mjd_end)`

Fetch neutrinos from a local archive. Archive must be defined and stored on LLAMA S3 for the queried neutrinos, and you must have access to LLAMA S3 API credentials to retrieve the archive.

Parameters

- **mjd_start** (*float*) – The start of the time window in which to search for neutrinos in Modified Julian Date (MJD) format. Use `gps2mjd` or `utc2mjd` from the `llama.utils` module to make the time conversions to the appropriate format.

- **mjd_end** (*float*) – The end of the time window in which to search for neutrinos.

Returns formatted – A list of neutrino dictionaries formatted as expected by IceCubeNeutrinoList.
If mjd_start > mjd_end, no error will be raised; instead, an empty list of neutrinos will be returned.

Return type list

Raises FileNotFoundError – If data for the requested time range is not available.

Examples

You can confirm that the archival neutrino data is available by fetching a specific trigger from the archive: >>>
neutrino = get_archive(58392, 58393)[0] >>> neutrino['run'] 131569 >>> neutrino['event'] 43188689

```
llama.files.i3.json.get_archive_and_real_neutrinos(mjd_start, mjd_end, blinded_neutrinos=None,
check_complete=False)
```

Get a mix of archival and real online GFU neutrinos (automatically calls `get_archive` and `get_real` as appropriate). Pulls online neutrinos for dates after `MJD_TOPIC_SWITCH_17_TO_BLANK` and archival neutrinos for dates prior.

Parameters

- **mjd_start** (*float*) – The start of the time window in which to search for neutrinos in Modified Julian Date (MJD) format. Use `gps2mjd` or `utc2mjd` from the `llama.utils` module to make the time conversions to the appropriate format.
- **mjd_end** (*float*) – The end of the time window in which to search for neutrinos.
- **blinded_neutrinos** (*bool, optional*) – Whether the neutrinos should be forcefully blinded or not. Use this to blind the neutrinos even if they are from time windows approved for unblinding (e.g. when doing background sensitivity studies). If not specified, use `UNBLINDED_TIMES` to determine which neutrinos can be unblinded.
- **check_complete** (*bool, optional*) – Neutrinos from IceCube can be delayed due to communications issues; to combat this, when neutrinos are pulled from the online data stream and `check_complete` is `True`, a check will be run to see if the next neutrino after the end of the search window is yet available. If it is not available, it is assumed that not all neutrinos needed for the search have arrived from south pole, and a `GenerationError` will be raised, allowing the pipeline to cooldown and retry the neutrino pull later.

Returns formatted – A list of neutrino dictionaries formatted as expected by `IceCubeNeutrinoList`.
If mjd_start > mjd_end, no error will be raised; instead, an empty list of neutrinos will be returned.
Each neutrino will be blinded if `blinded_neutrinos` is `True` or if its event time falls outside the list of approved `UNBLINDED_TIMES`.

Return type list

Examples

Pull some blinded archival neutrinos >>> a_start = MJD_TOPIC_SWITCH_17_TO_BLANK-1.77 >>> a_end = MJD_TOPIC_SWITCH_17_TO_BLANK >>> arc = get_archive_and_real_neutrinos(a_start, a_end, True) >>>
len(arc) > 0 True

Pull some blinded GFU neutrinos >>> l_start = MJD_TOPIC_SWITCH_17_TO_BLANK >>> l_end = MJD_TOPIC_SWITCH_17_TO_BLANK + 500/86400. >>> live = get_archive_and_real_neutrinos(l_start, l_end, True) >>> len(live) > 0 True

Pull a mix of archival and blinded neutrinos >>> mix = get_archive_and_real_neutrinos(a_start, l_end, True)
>>> len(arc) + len(live) == len(mix) True

`llama.files.i3.json.get_dec_filter(dec)`

Return a function that will filter neutrinos based on DECLINATION, e.g. for use with the `filter` function.

Parameters `dec` (*list*) – A list of declination intervals (also specified as lists) into which a selected neutrino should fall, e.g. [[21, 41],[92, 102]].

Returns `dec_filter` – A function which returns True when passed a neutrino that falls within at least one of the declination intervals passed included in `dec`. The neutrino is expected to be in the format returned by `format_neutrinos`.

Return type function

Examples

```
>>> neutrino = {  
...     'azimuth': 246.16444118441663,  
...     'bdt_up': -0.0155,  
...     'energy': 498.10789,  
...     'mjd': 57982.5321185039,  
...     'sigma': 1.5478695443038595,  
...     'zenith': 113.7504569829126  
... }  
>>> ra, dec = zen_az2ra_dec(  
...     neutrino['mjd'],  
...     neutrino['zenith'],  
...     neutrino['azimuth'])  
... )  
>>> f"RA: {ra:.3f}, Dec: {dec:.3f}"  
'RA: 1.253, Dec: 23.653'  
>>> get_dec_filter([[0,30],[50,60]])(neutrino)  
True  
>>> get_dec_filter([[0,20],[50,60]])(neutrino)  
False
```

`llama.files.i3.json.get_ra_filter(ra)`

Return a function that will filter neutrinos based on RIGHT ASCENSION, e.g. for use with the `filter` function.

Parameters `ra` (*list*) – A list of right ascension intervals (also specified as lists) into which a selected neutrino should fall, e.g. [[21, 41],[92, 102]].

Returns `ra_filter` – A function which returns True when passed a neutrino that falls within at least one of the right ascension intervals passed included in `ra`. The neutrino is expected to be in the format returned by `format_neutrinos`.

Return type function

Examples

```
>>> neutrino = {
...     'azimuth': 246.16444118441663,
...     'bdt_up': -0.0155,
...     'energy': 498.10789,
...     'mjd': 57982.5321185039,
...     'sigma': 1.5478695443038595,
...     'zenith': 113.7504569829126
... }
>>> ra, dec = zen_az2ra_dec(
...     neutrino['mjd'],
...     neutrino['zenith'],
...     neutrino['azimuth']
... )
>>> f"RA: {ra:.3f}, Dec: {dec:.3f}"
'RA: 1.253, Dec: 23.653'
>>> get_ra_filter([[0, 30], [50, 60]])(neutrino)
True
>>> get_ra_filter([[5, 30], [50, 60]])(neutrino)
False
```

`llama.files.i3.json.get_real(mjd_start, mjd_end, check_complete=False)`

Fetch real neutrinos from ICECUBE's database; takes gps time of the events and the time range about this central value over which to search for neutrinos.

Parameters

- **mjd_start** (*float*) – The start of the time window in which to search for neutrinos in Modified Julian Date (MJD) format. Use `gps2mjd` or `utc2mjd` from the `llama.utils` module to make the time conversions to the appropriate format.
- **mjd_end** (*float*) – The end of the time window in which to search for neutrinos.
- **check_complete** (*bool, optional*) – Neutrinos from IceCube can be delayed due to communications issues; to combat this, when `check_complete` is `True`, a check will be run to see if the next neutrino after the end of the search window is yet available. If it is not available, it is assumed that not all neutrinos needed for the search have arrived from south pole, and a `GenerationError` will be raised, allowing the pipeline to cooldown and retry the neutrino pull later.

Returns `formatted` – A list of neutrino dictionaries formatted as expected by `IceCubeNeutrinoList`. If `mjd_start > mjd_end`, no error will be raised; instead, an empty list of neutrinos will be returned.

Return type

Raises `ValueError` – If the start time of the neutrino pull predates the switch from the ‘neutrino17’ topic to the ‘neutrino’ topic (in late 2018), a `ValueError` is raised. The date of this switch is given by `MJD_TOPIC_SWITCH_17_TO_BLANK`.

Examples

Get some neutrinos from late-2018 on the ‘neutrino’ topic: >>> from llama.utils import utc2mjd >>> mjd = utc2mjd("2018-09-30") >>> len(get_real(mjd-500/86400., mjd+500/86400.)) > 0 True

If your end time is earlier than your start time, you’ll get an empty neutrino list: >>> get_real(mjd, mjd-1) []

You should be able to fetch this neutrino from the online GFU database: >>> neutrino = get_archive(58392, 58393)[0] >>> neutrino['run'] 131569 >>> neutrino['event'] 43188689

llama.files.i3.json.neutrino_archive_available_years()

Get a list of years for which archival GFU neutrinos are available. These files are stored remotely on LLAMA S3 (see `llama.com.s3`) as .npy files and are cached locally in `llama.utils.OBJECT_DIR`; these are private files that require LLAMA S3 credentials to access. Archival neutrino releases available via LLAMA S3 are recorded in `ARCHIVAL_NEUTRINOS`, and the default archival release version is contained in `DEFAULT_ARCHIVE`.

Examples

Confirm that we have access to archival neutrinos from 2011 through 2018:

```
>>> neutrino_archive_available_years()  
[2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018]
```

`llama.files.i3.json.random()` → x in the interval [0, 1).

28.3.2 `llama.files.i3.realtime_tools_stubs` module

Provide `realtime_tools` functionality from IceCube so that the pipeline can run without installing a full-fledged IceCube environment. In particular, implement `live.get_events` and `neutrino_singlet_pulls.calc_cr_pull`.

User must provide a valid IceCube username and password as environmental variables `ICECUBE_USERNAME` and `ICECUBE_PASSWORD`, respectively.

llama.files.i3.realtime_tools_stubs.calc_cr_pull(*cr_zen*, *cr_azi*, *zenith*, *muex*)

Calculate CR pull corrected value, based on *cr_zen*, *cr_azi*, *zenith* and *muex* values based on Thomas’s polynomial fit for neutrino16 sample

NOTE: This is modified from the IceCube code to run stand-alone.

llama.files.i3.realtime_tools_stubs.events_dtype()

Datatypes for events.

llama.files.i3.realtime_tools_stubs.get_events(*topic*, *starttime*, *stoptime*, *timekey=None*, *retries=10*, *delay=30*)

Query events from the I3Live database.

Parameters

- **topic** (*str*) – Name of the stream (e.g. ‘neutrino16’).
- **starttime** (*int or float*) – start of timewindow
- **stoptime** (*int or float*) – stop of timewindow
- **timekey** (*str*) – which date/time to compare to, such as `insert_time` or `value.data.eventtime` (default: auto-detect)
- **retries** (*int*) – Number of retries in case of HTTP errors
- **delay** (*int or float*) – time delay between retries (default: 10 seconds)

Returns

- **events** (*list or None*) – Events matching the given criteria or None in case of (repeated) HTTP error.
- **NOTE** (*This is modified from the IceCube code to run stand-alone.*)

`llama.files.i3.realtime_tools_stubs.load_spline(filename)`

Taken from:

<https://code.icecube.wisc.edu/projects/icecube/browser/IceCube/> projects/realtime_gfu/releases/V19-02-00/python/angular_errors.py

`llama.files.i3.realtime_tools_stubs.parse_neutrino_event_stream(messages, dtype=None)`

Stub version of the `icecube.realtime_gfu.eventcache.NeutrinoEventStream()`.`parse` method, published at:

<https://code.icecube.wisc.edu/projects/icecube/browser/IceCube/>
projects/realtime_gfu/releases/V19-02-00/python/eventcache.py

Parses and formats a bunch of neutrinos in the format returned by `icecube.realtime_tools.get_live`, picking the correct fields from those neutrinos such that the result is suitable for analyses. Based on discussions with Thomas Kintscher and Josh Wood.

`llama.files.i3.realtime_tools_stubs.pull_correct_online_2017(cramerrao, paraboloid, bootstrap, logE, mode='MuEX')`

Taken from:

<https://code.icecube.wisc.edu/projects/icecube/browser/IceCube/> projects/realtime_gfu/releases/V19-02-00/python/angular_errors.py

`llama.files.i3.realtime_tools_stubs.pull_correction(values, logE, splines, fixedFallback=None)`

Apply pull-correction to angular error estimators. Taken from:

<https://code.icecube.wisc.edu/projects/icecube/browser/IceCube/>
projects/realtime_gfu/releases/V19-02-00/python/angular_errors.py

Parameters

- **values** (*list*) – List of NumPy arrays with the sigma estimates.
- **logE** (*array*) – The logE of the events.
- **splines** (*list*) – Splines to be used for the corrections.
- **fixedFallback** (*float*) – Should all estimators fail, use this fixed value

`llama.files.i3.realtime_tools_stubs.to_datestring(value)`

Convert a given timestamp into a string of ‘YYYY-MM-DD hh:mm:ss.fffff’.

The input must be either an MJD (float) or a datetime object.

NOTE: This is modified from the IceCube code to run stand-alone. It *DOES NOT ACCEPT* I3Time instances.

28.3.3 `llama.files.i3.tex` module

`FileHandler` for making a nicely-formatted LaTeX table of IceCube neutrinos including their reconstructed properties and their joint significance when combined with a gravitational wave.

```
class llama.files.i3.tex.IceCubeNeutrinoListTex(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler
```

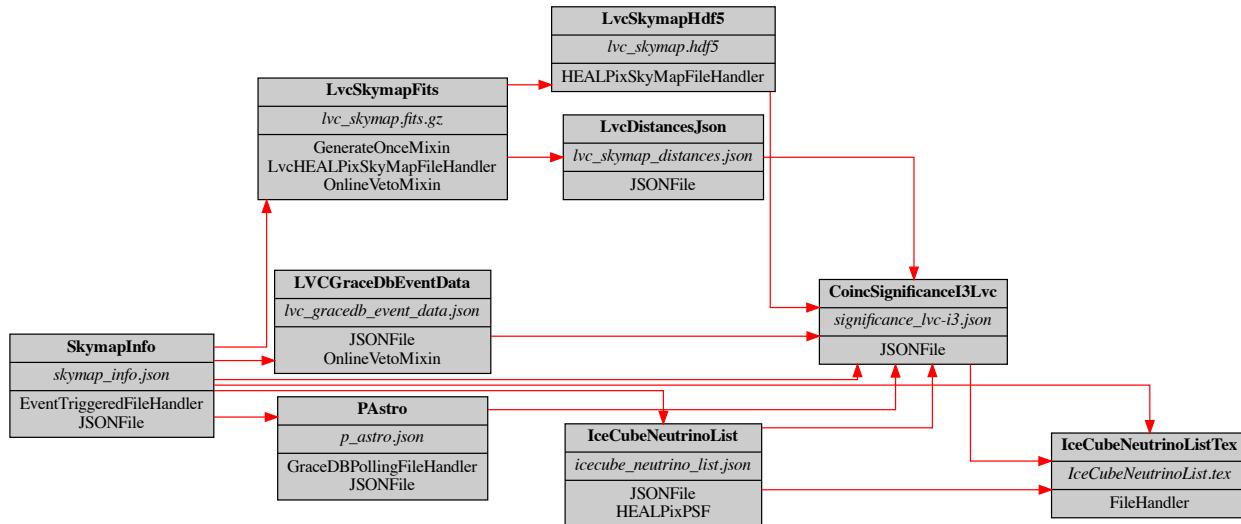


Fig. 53: Required input files for `llama.files.i3.tex.IceCubeNeutrinoListTex` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.tex.IceCubeNeutrinoListTex` to be generated.

A list of neutrinos and their reconstructed properties as well as the significance of the joint event) in LaTeX format suitable for use in papers and coincident skymap summary plots.

```
DEPENDENCIES = (<class 'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'	llama.files.skymap_info.SkymapInfo'>, <class
'	llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)

FILENAME = 'IceCubeNeutrinoList.tex'

MANIFEST_TYPES = (<class 'llama.files.i3.tex.IceCubeNeutrinoListTex'>,)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'	llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'	llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'	llama.files.i3.json.IceCubeNeutrinoList'>, <class
'	llama.files.gracedb.LVCGraceDbEventData'>, <class
'	llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)
```

```

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc': ImmutableDict({<class
'llama.files.gracedb.LVCGraceDbEventData': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.lvc_skymap.LvcDistancesJson': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.gracedb.PAstro': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
llama.files.i3.tex.convert_json_neutrinos_to_latex(infilename, outfilename, gpstime=None,
n_values=None)

```

Take a json-formatted input filename and an output filename (both full paths) as arguments. Convert the neutrino list in the infile into a LaTeX table that can be inserted into a LaTeX document and save it to the path specified in outfilename. If an optional gpstime is given, then the time column will be given as the time difference in seconds between the neutrino's arrival and the GPS time provided, i.e. the neutrino detection time with t=0 defined as the GPS time. If a list of p_values corresponding to the neutrinos in infile are provided, then an extra column is added with the P-values.

```
llama.files.i3.tex.latex_header(num_cols)
```

Return a header with the appropriate number of columns for a LaTeX table section.

28.3.4 `llama.files.i3.txt` module

`FileHandler` for making a nicely-formatted ascii table of IceCube neutrinos including their reconstructed properties and their joint significance when combined with a gravitational wave. Can also be used as a stand-alone command-line script to generate a human-readable table with neutrinos for this event from an input JSON file. Used for GCN circulars.

```
class llama.files.i3.txt.IceCubeNeutrinoListCoincTxt(eventid_or_fh, rundir=None)
```

Bases: `llama.filehandler.FileHandler`

A space-delimited, table-formatted list of neutrinos meant for human consumption in GCN Circulars. Contains coincidence significance measures in addition to the base neutrino properties.

```
DEPENDENCIES = (<class 'llama.files.i3.json.IceCubeNeutrinoList'>, <class 'llama.files.skymap_info.SkymapInfo'>, <class 'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)
```

```
FILENAME = 'icecube neutrino list coinc.txt'
```

MANIFEST TYPES = (<class 'llama.files.i3.txt.IceCubeNeutrinoListCoincTxt'>,)

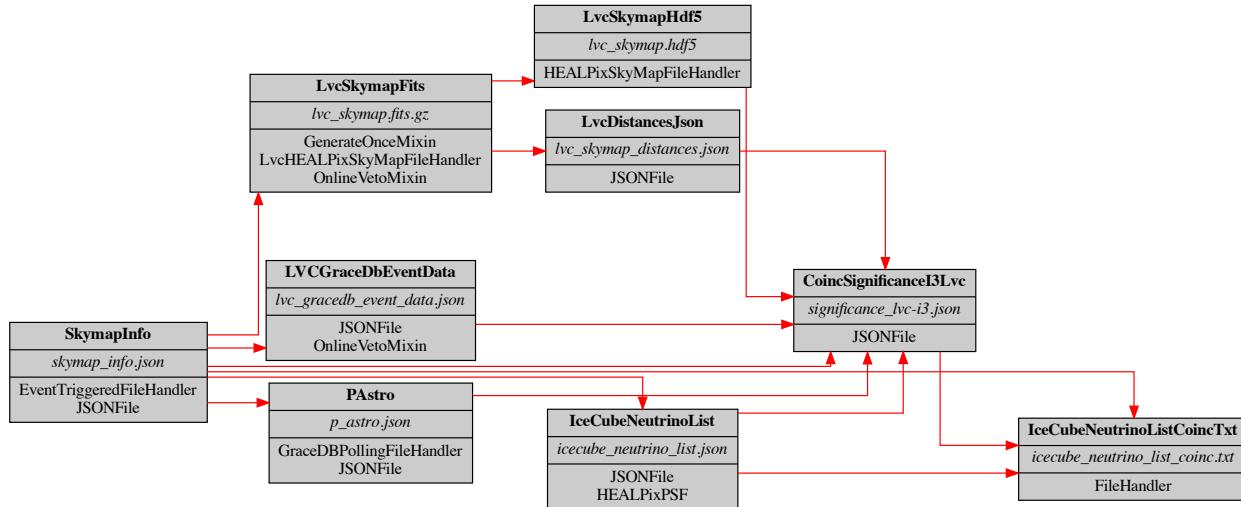


Fig. 54: Required input files for `llama.files.i3.txt.IceCubeNeutrinoListCoincTxt` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.txt.IceCubeNeutrinoListCoincTxt` to be generated.

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}), <class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({}), <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>: ImmutableDict({<class
'llama.files.gracedb.LVCGraceDbEventData'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}), <class
'llama.files.i3.json.IceCubeNeutrinoList'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}), <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>: ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}), <class
'llama.files.lvc_skymap.LvcDistancesJson'>: ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}), <class
'llama.files.gracedb.PAstro'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})})})})
  
```

```

class llama.files.i3.txt.IceCubeNeutrinoListTxt(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler
  
```

A space-delimited, table-formatted list of neutrinos formatted for human consumption in GCN Circulars.

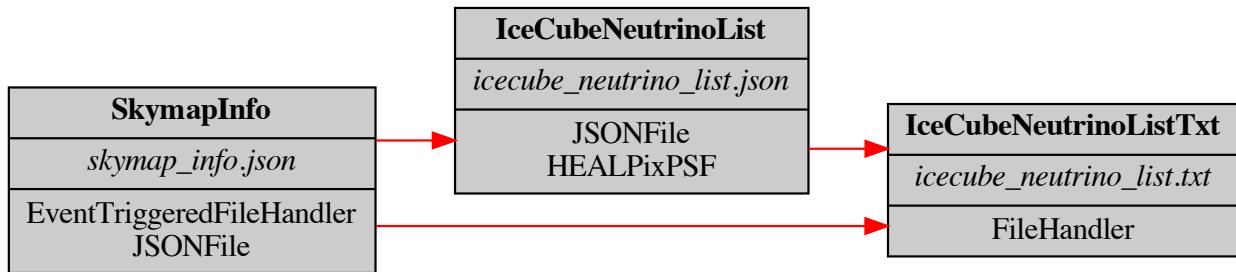


Fig. 55: Required input files for `llama.files.i3.txt.IceCubeNeutrinoListTxt` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.txt.IceCubeNeutrinoListTxt` to be generated.

```

DEPENDENCIES = (<class 'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'icecube_neutrino_list.txt'

MANIFEST_TYPES = (<class 'llama.files.i3.txt.IceCubeNeutrinoListTxt'>,)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})})

class llama.files.i3.txt.RctGwaIceCubeNeutrinoListTxt(eventid_or_fh, rundir=None)
Bases: llama.files.gwastro.GWAstroReceipt

```

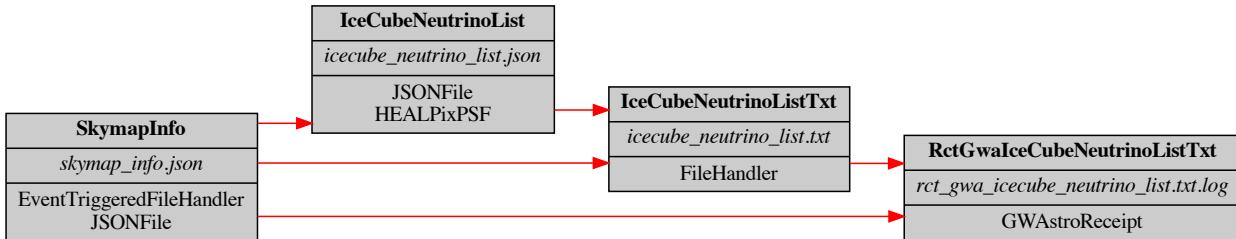


Fig. 56: Required input files for `llama.files.i3.txt.RctGwaIceCubeNeutrinoListTxt` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.i3.txt.RctGwaIceCubeNeutrinoListTxt` to be generated.

```

DEPENDENCIES = (<class 'llama.files.i3.txt.IceCubeNeutrinoListTxt'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_gwa_icecube_neutrino_list.txt.log'

MANIFEST_TYPES = (<class 'llama.files.i3.txt.RctGwaIceCubeNeutrinoListTxt'>,)

RSYNC_DEST = 'gwhen@gw-astronomy.org:/home/gwhen/content/events/'

UPLOAD
alias of llama.files.i3.txt.IceCubeNeutrinoListTxt

```

```
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.i3.txt.IceCubeNeutrinoListTxt'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.i3.txt.IceCubeNeutrinoListTxt'>)})

llama.files.i3.txt.convert_json_neutrinos_to_txt(infilename, outfilename, gpstime=None,
                                                p_values=None)
```

Take a json-formatted input filename and an output filename (both full paths) as arguments. Convert the neutrino list in the infile into a space-delimited, table-formatted list of neutrinos intended for human readability and saved to the path specified in outfilename. If an optional gpstime is given, then the time column will be given as the time difference in seconds between the neutrino's arrival and the GPS time provided, i.e. the neutrino detection time with t=0 defined as the GPS time.

```
llama.files.i3.txt.main()
```

Can also use this module as a command-line script to make this file conversion.

28.3.5 `llama.files.i3.utils` module

Coordinate conversions for IceCube.

```
llama.files.i3.utils.icecube_coords()
```

Get the `astropy.coordinates.EarthLocation` of the IceCube detector.

```
llama.files.i3.utils.ra_dec2zen_az(mjd, ra_deg, dec_deg)
```

Take the right ascension/declination equatorial coordinates (used by LIGO and others) of the neutrino and convert them into the zenith and azimuth measured from IceCube (the earth-based coordinates measured from the ICECUBE site at the south pole).

Parameters

- `mjd` (*float or array-like*) – The Modified Julian Date at which to make the conversion.
- `ra_deg` (*float or array-like*) – The right-ascension in degrees of the point or points on the sky.
- `dec_deg` (*float or array-like*) – Same, but for the declination.

Returns

- `zen_deg` (*float or array-like*) – The zenith angle of the point on the sky as measured from IceCube's location in degrees. Will be a scalar or vector value based on the input.
- `az_deg` (*float or array-like*) – Same, but for the azimuthal angle.

Examples

Pull an archival neutrino and compare its official zenith/azimuth to the conversion provided by this function.

```
>>> import numpy as np
>>> from llama.files.i3.json import get_archive
>>> ns = get_archive(58392, 58393)
>>> mjds = np.array([n['mjd'] for n in ns])
>>> ras = np.array([n['ra'] for n in ns])
>>> decs = np.array([n['dec'] for n in ns])
>>> azs = np.array([n['azimuth'] for n in ns])
>>> zens = np.array([n['zenith'] for n in ns])
```

(continues on next page)

(continued from previous page)

```
>>> czens, cazs = ra_dec2zen_az(mjds, ras, decs)
>>> max(abs(czens - zens)) < 0.001
True
>>> max(abs(cazs - azs)) < 0.001
True
```

`llama.files.i3.utils.zen_az2ra_dec(mjd, zen_deg, az_deg)`

Take the zenith and azimuth of the neutrino (these are earth-based coordinates measured from the ICECUBE site at the south pole) and convert them into the right ascension/declination equatorial coordinates used by LIGO and others.

Parameters

- `mjd (float or array-like)` – The Modified Julian Date at which to make the conversion.
- `zen_deg (float or array-like)` – The zenith angle of the point on the sky as measured from IceCube's location in degrees.
- `az_deg (float or array-like)` – Same, but for the azimuthal angle.

Returns

- `ra_deg (float or array-like)` – The right-ascension in degrees of the input point or points. Will be a scalar or vector value based on the input.
- `dec_deg (float or array-like)` – Same, but for the declination.

Examples

Pull an archival neutrino and compare its official RA/dec to the RA dec conversion provided by this function.

```
>>> import numpy as np
>>> from llama.files.i3.json import get_archive
>>> ns = get_archive(58392, 58393)
>>> mjds = np.array([n['mjd'] for n in ns])
>>> ras = np.array([n['ra'] for n in ns])
>>> decs = np.array([n['dec'] for n in ns])
>>> azs = np.array([n['azimuth'] for n in ns])
>>> zens = np.array([n['zenith'] for n in ns])
>>> cras, cdecs = zen_az2ra_dec(mjds, zens, azs)
>>> max(abs(cras - ras)) < 0.001
True
>>> max(abs(cdecs - decs)) < 0.001
True
```

28.4 llama.files.lvc_skymap package

FileHandler classes related to working with skymaps provided by the LIGO-Virgo Collaboration (LVC).

```
class llama.files.lvc_skymap.LvcDistancesJson(eventid_or_fh, rundir=None)
    Bases: llama.filehandler.JSONFile
```

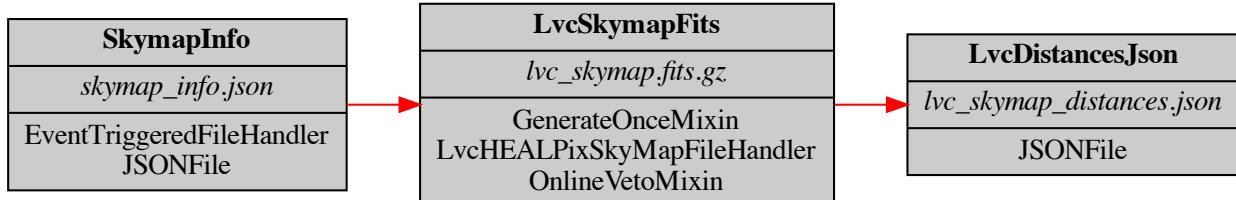


Fig. 57: Required input files for `llama.files.lvc_skymap.LvcDistancesJson` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.lvc_skymap.LvcDistancesJson` to be generated.

A simple JSON file with the reconstructed distances to an event. Contains the mean reconstructed distance as `distmean` and the standard deviation as `diststd`.

```
DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapFits'>,)
FILENAME = 'lvc_skymap_distances.json'
MANIFEST_TYPES = (<class 'llama.files.lvc_skymap.LvcDistancesJson'>,)
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvc_skymap.LvcSkymapFits'>})
property distmean
    The mean distance to the event; a probability-weighted average of reconstructed distances. Extracted from the original LVC skymap.
property diststd
    The probability-weighted standard deviation of the reconstructed distance to the event. Extracted from the original LVC skymap.
```

```
class llama.files.lvc_skymap.LvcSkymapFits(eventid_or_fh, rundir=None)
    Bases: llama.filehandler.GenerateOnceMixin, llama.files.healpix.
    LvcHEALPixSkyMapFileHandler, llama.filehandler.mixins.OnlineVetoMixin
```

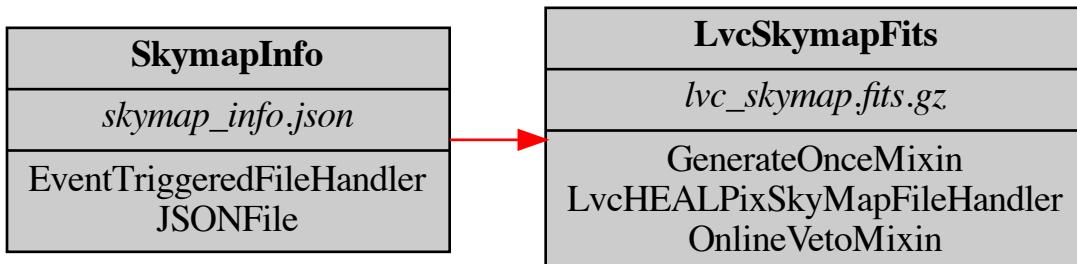


Fig. 58: Required input files for `llama.files.lvc_skymap.LvcSkymapFits` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.lvc_skymap.LvcSkymapFits` to be generated.

The skymap suggested by the LVC Initial GCN Notice.

```
DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
DETECTORS = (Detector(name='LVC', abbrev='lvc', fullname='LVC',
url='http://wiki.ligo.org', summary='LVC', description='',
citations=ImmutableDict({})),)
FILENAME = 'lvc_skymap.fits.gz'
MANIFEST_TYPES = (<class 'llama.files.lvc_skymap.LvcSkymapFits'>,)
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})
class llama.files.lvc_skymap.LvcSkymapHdf5(eventid_or_fh, rundir=None)
Bases: llama.files.healpix.HEALPixSkyMapFileHandler
```

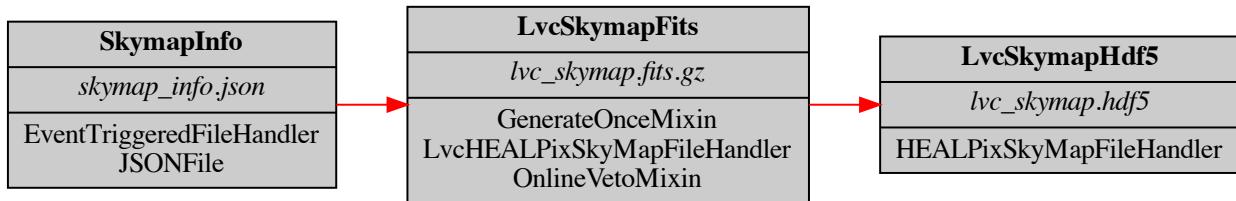


Fig. 59: Required input files for `llama.files.lvc_skymap.LvcSkymapHdf5` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.lvc_skymap.LvcSkymapHdf5` to be generated.

An HDF5-formatted copy of the initial LVC skymap. Loads more quickly.

```
DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapFits'>,)
DETECTORS = (Detector(name='LVC', abbrev='lvc', fullname='LVC',
url='http://wiki.ligo.org', summary='LVC', description='',
citations=ImmutableDict({})),)
FILENAME = 'lvc_skymap.hdf5'
MANIFEST_TYPES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>,)
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'	llama.files.lvc_skymap.LvcSkymapFits'>)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvc_skymap.LvcSkymapFits'>})
exception llama.files.lvc_skymap.SkymapNotFoundError
Bases: OSError
```

Raised when a valid skymap cannot be found or downloaded from GraceDb.

```
llama.files.lvc_skymap.available_skymaps(graceid)
Get a list of available skymap logs in ascending order of creation time from GraceDB for a given graceid.
llama.files.lvc_skymap.skymap_filenames(filenames)
Return only filenames from filenames that could be skymap filenames.
```

28.4.1 `llama.files.lvc_skymap.utils` module

Utilities for working with LVC skymaps

```
class llama.files.lvc_skymap.utils.SkymapFilename(string)
```

Bases: str

A skymap filename on GraceDB with built-in parsing of skymap file along with convenience methods used for selecting and canonicalizing skymap filenames.

property basename

Remove the filename extension and just get the base name. Since this base name corresponds to the information contained in the skymap (vs. the format of that information, given by `SkymapFilename.extension`), this should be a good indicator of the contents of the skymap.

```
>>> skymap = SkymapFilename("bayestar.multiorder.fits")
>>> skymap.basename
'bayestar'
```

```
>>> skymap0 = SkymapFilename("bayestar.multiorder.fits,0")
>>> skymap0.basename
'bayestar'
```

canonicalize(graceid, canonical_date=None)

Return the canonicalized filename (including version information) for this skymap filename for the given graceid at the given date (instance of `datetime.datetime` or `None` to use the current time). If this filename already contains a version, just return self. Queries GraceDB to resolve this information. Raises a `ligo.gracedb.rest.HTTPError` if a GraceDB query fails to connect.

For this event, a second version (v1) of `bayestar.fits` was created at 2019-05-10 04:03:40 UTC; canonicalizing at an earlier date should return v0 (the original) while a later date should return v1 (the update):

```
>>> from dateutil.parser import parse
>>> skymap = SkymapFilename("bayestar.fits")
>>> skymap.canonicalize("S190510g", parse("2019-05-10 04:03:30 UTC"))
'bayestar.fits,0'
>>> skymap.canonicalize("S190510g", parse("2019-05-10 04:03:50 UTC"))
'bayestar.fits,1'
```

If the skymap name was already canonicalized, then the original value will be returned:

```
>>> canonical = SkymapFilename("bayestar.fits,0")
>>> canonical == canonical.canonicalize("foo")
True
```

If the file does not exist at the requested time, then the version should be assumed to be the zeroth version (since this will be the first available version):

```
>>> nonexistent = SkymapFilename("bayestar.fits")
>>> unix0 = datetime.datetime.fromtimestamp(0)
>>> nonexistent.canonicalize("S190510g", unix0)
'bayestar.fits,0'
```

property extension

The file extension, drawn from `SKYMAP_FILE_EXTENSIONS`.

```
>>> skymap = SkymapFilename("bayestar.multiorder.fits")
>>> skymap.extension
' .multiorder.fits'
```

```
>>> skymap0 = SkymapFilename("bayestar.multiorder.fits,0")
>>> skymap0.extension
' .multiorder.fits'
```

property filename

The filename of this skymap, including the file extension, with the version number removed.

```
>>> skymap = SkymapFilename("bayestar.multiorder.fits")
>>> skymap.filename
'bayestar.multiorder.fits'
```

```
>>> skymap0 = SkymapFilename("bayestar.multiorder.fits,0")
>>> skymap0.filename
'bayestar.multiorder.fits'
```

property version

The version of this skymap filename as an integer (since many versions of the same filename can be uploaded). Returns None if not specified.

```
>>> skymap = SkymapFilename("bayestar.multiorder.fits")
>>> skymap.version == None
True
```

```
>>> skymap0 = SkymapFilename("bayestar.multiorder.fits,0")
>>> skymap0.version
0
```

`llama.files.lvc_skymap.utils.available_skymaps(graceid)`

Get a list of available skymap logs in ascending order of creation time from GraceDB for a given graceid.

`llama.files.lvc_skymap.utils.skymap_filenames(filenames)`

Return only filenames from `filenames` that could be skymap filenames.

28.5 `llama.files.skymap_info` package

`SkymapInfo` records basic information about GW triggers/skymaps being used for a LLAMA analysis in a uniform, simple format.

`class llama.files.skymap_info.FarThresholdMixin`
Bases: `llama.filehandler.mixins.ObservingVetoMixin`

An `ObservingVetoMixin` that only triggers on plausible-seeming events whose FAR is less than 10 per day.

`class_vetoes = ((<function event_far_greater_than_10_per_day>, None),)`

`class llama.files.skymap_info.SkymapInfo(eventid_or_fh, rundir=None)`
Bases: `llama.filehandler.EventTriggeredFileHandler, llama.filehandler.JSONFile`

A JSON file containing information on where to find the latest and greatest skymap for this event. Any incoming alert file should also generate this file. The resulting file is a simple JSON object with key:value pairs concerning information about the event and the current skymap to be used.

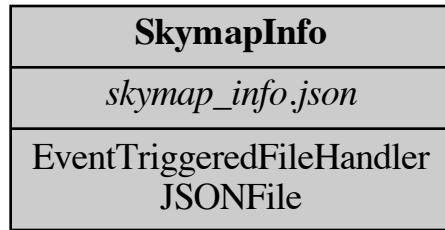


Fig. 60: `llama.files.skymap_info.SkymapInfo` is created from external triggers. It therefore has no LLAMA-representable input dependencies, but instead acts as initial input for other `FileHandler` classes.

```
FILENAME = 'skymap_info.json'
MANIFEST_TYPES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
UR_DEPENDENCIES = ()
UR_DEPENDENCY_TREE = frozenset({})

property alert_file_handler
    Get the file handler for this alert file type. For a manually-generated instance of this file, the return value is None; otherwise, it is a FileHandler instance matched to this event.

property alert_type
    Get the type of the alert, i.e. how this trigger was made in the LLAMA pipeline. For example, if this trigger was made as part of a gstlal offline subthreshold search, the value would be "gstlal-offline". Look in SkymapInfo._alert_types for allowed alert type values.

NB: This is NOT related to the ``alert_type`` key in LVAalerts!

property event_time_gps
    Get the time of the observed event up to nanosecond precision (less accurate than this in practice).

property event_time_gps_nanoseconds
    Get the number of nanoseconds past the last GPS second at the time of the observed event. Ostensibly provides nanosecond precision, but is less accurate than this in practice.

property event_time_gps_seconds
    Get the time of the observed event in GPS seconds (truncated to the nearest second).

property event_time_mjd
    Get the time of the observed event in Modified Julian Day format (UTC time).

property event_time_str
    Get a unicode string with the ISO date and time of the observed event straight from the VOEvent file. UTC time.

property far
    Get the false alarm rate of this VOEvent as a float value.

generate_from_gracedb(graceid: str, skymap_filename: Optional[str] = None)
    Supplement the original filename with data pulled from the GraceDB API on the event. Used for manually generating skymap_info.

generate_from_gracedb_superevent(graceid: str, skymap_filename: Optional[str] = None)
    Supplement the original filename with data pulled from the GraceDB API on the superevent. Used for manually generating skymap_info from a superevent. If no skymap is provided, pull down the most recently uploaded one defined in llama.files.lvc_skymap.SKYMAP_FILERAMES.
```

generate_from_lvc_gcn_xml(*voe*: llama.files.lvc_gcn_xml.LvcGcnXml)

Generate this file from LvcGcnXml VOEvent. Used for implementation.

property gracedb_url

Get the GraceDB URL for this GraceID, assuming it is in the expected format for GraceIDs. Will autodetect if this is a superevent or regular event. Returns None if this doesn't look like a valid GraceID (but will not check on the actual gracedb server).

property graceid

Get the GraceID corresponding to this VOEvent.

property human_url

Get the human-viewable URL for this file (i.e. the one that can be accessed from a web-browser using interactive authentication)

property is_super

Check whether this GraceID corresponds to an event or a superevent.

property notice_time_str

Get a unicode string with the date and time of creation of the notification associated with this VOEvent, rather than the time of detection of the event itself.

property pipeline

Return an ALL-CAPS name of the pipeline used to generate this event, e.g. GSTLAL or CWB.

property skymap_filename

Get the filename of the skymap as a SkymapFilename as stored on GraceDB.

llama.files.skymap_info.event_far_greater_than_10_per_day(*eventdir*)

Veto if the FAR defined in SkymapInfo is greater than ten day.

llama.files.skymap_info.gracedb_url(*graceid*)

Return the GraceDB URL for this GraceID (assuming it exists and is in a recognized GraceID format). If this doesn't look like a valid GraceID, returns None.

llama.files.skymap_info.is_regular_graceid(*graceid*)

Check whether this is a normal GraceID (by seeing whether is_super throws an error).

28.5.1 llama.files.skymap_info.cli module

CLI for making new SkymapInfo files.

class llama.files.skymap_info.cli.Parsers

Bases: object

CLI flags related to LIGO/Virgo searches. Same idea as llama.cli.Parsers but with narrower scope.

```
graceid = CliParser(prog='sphinx-build', usage=None, description=None,
formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error',
add_help=False)
```

```
skymap = CliParser(prog='sphinx-build', usage=None, description=None,
formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error',
add_help=False)
```

28.5.2 `llama.files.skymap_info.utils` module

Utilities for working with GW searches that depend on SkymapInfo.

`llama.files.skymap_info.utils.gracedb_human_file_url(graceid, filename, version=None)`

Get a human-viewable (i.e. browser-viewable) URL for a specific filename and event combination on GraceDB.

You can either specify the version as an argument, omit the version, or pass the version straight in as part of the filename (though explicitly passing the version number as an argument will override a version provided as part of the filename argument). Will check whether the event is a superevent and return the appropriate URL in that case.

`llama.files.skymap_info.utils.is_super(graceid)`

Check whether this GraceID corresponds to an event or a superevent.

28.6 `llama.files.slack` package

FileHandler classes that log file uploads to Slack.

Slack API documentation: <https://api.slack.com/methods/chat.postMessage> <https://api.slack.com/messaging/composing/formatting#>

(Link to permissions in top right of the above pages.)

`class llama.files.slack.RctSlkLmaLVAlertJSON(eventid_or_fh, rundir=None)`
Bases: `llama.files.slack.SlackReceiptLlama`

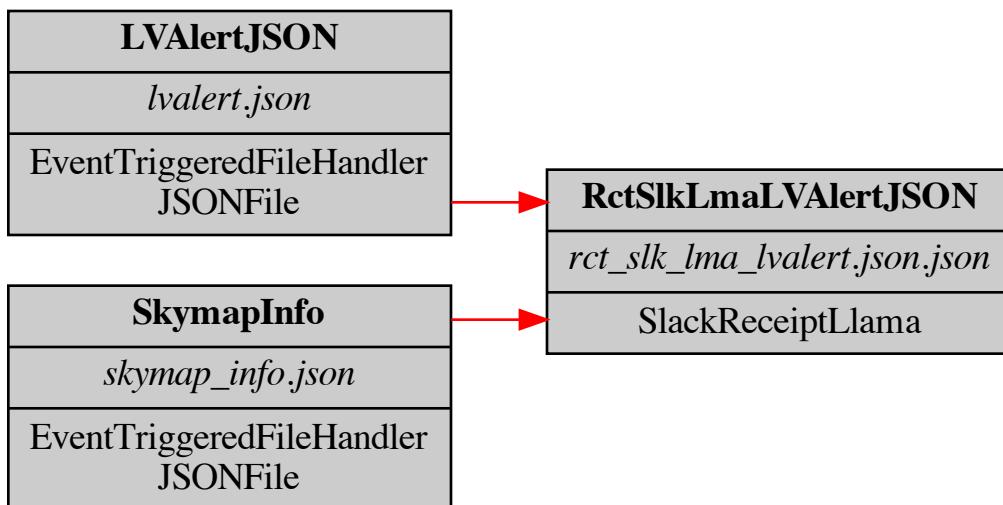


Fig. 61: Required input files for `llama.files.slack.RctSlkLmaLVAlertJSON` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.slack.RctSlkLmaLVAlertJSON` to be generated.

```
DEPENDENCIES = (<class 'llama.files.lvalert_json.LVAAlertJSON'>, <class 'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_lvalert.json.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class 'llama.files.slack.RctSlkLmaLVAlertJSON'>,)
```

```

UPLOAD
    alias of llama.files.lvalert_json.LVAlertJSON

UR_DEPENDENCIES = (<class 'llama.files.lvalert_json.LVAlertJSON'>, <class 'llama.files.skymap_info.SkymapInfo'>)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvalert_json.LVAlertJSON'>})
class_vetoes = ()

class llama.files.slack.RctSlkLmaLvcGcnXml(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

```

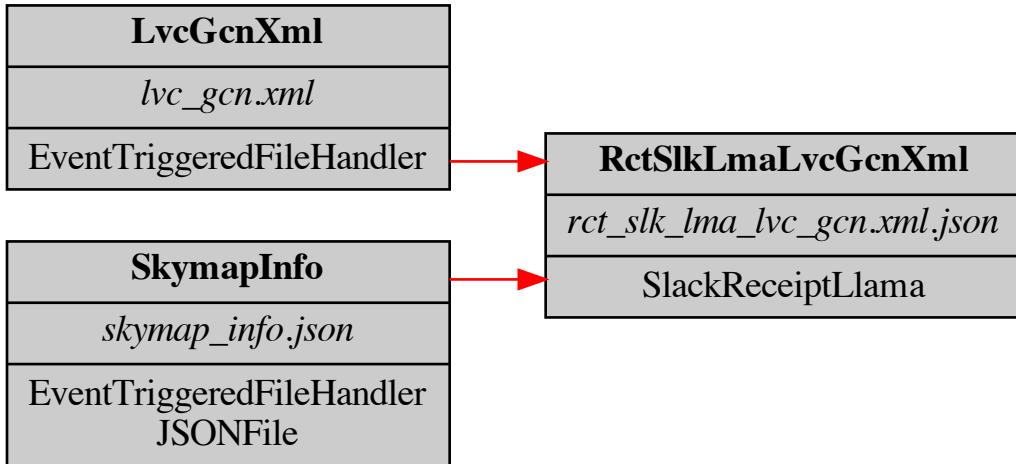


Fig. 62: Required input files for `llama.files.slack.RctSlkLmaLvcGcnXml` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.slack.RctSlkLmaLvcGcnXml` to be generated.

```

DEPENDENCIES = (<class 'llama.files.lvc_gcn_xml.LvcGcnXml'>, <class 'llama.files.skymap_info.SkymapInfo'>)
FILENAME = 'rct_slk_lma_lvc_gcn.xml.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'
MANIFEST_TYPES = (<class 'llama.files.slack.RctSlkLmaLvcGcnXml'>,)
UPLOAD
    alias of llama.files.lvc_gcn_xml.LvcGcnXml

UR_DEPENDENCIES = (<class 'llama.files.lvc_gcn_xml.LvcGcnXml'>, <class 'llama.files.skymap_info.SkymapInfo'>)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvc_gcn_xml.LvcGcnXml'>})
class_vetoes = ()

class llama.files.slack.RctSlkLmaLvcRetractionXml(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

DEPENDENCIES = (<class 'llama.files.lvc_gcn_xml.LvcRetractionXml'>, <class 'llama.files.skymap_info.SkymapInfo'>)
FILENAME = 'rct_slk_lma_lvc_gcn_retraction.xml.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

```

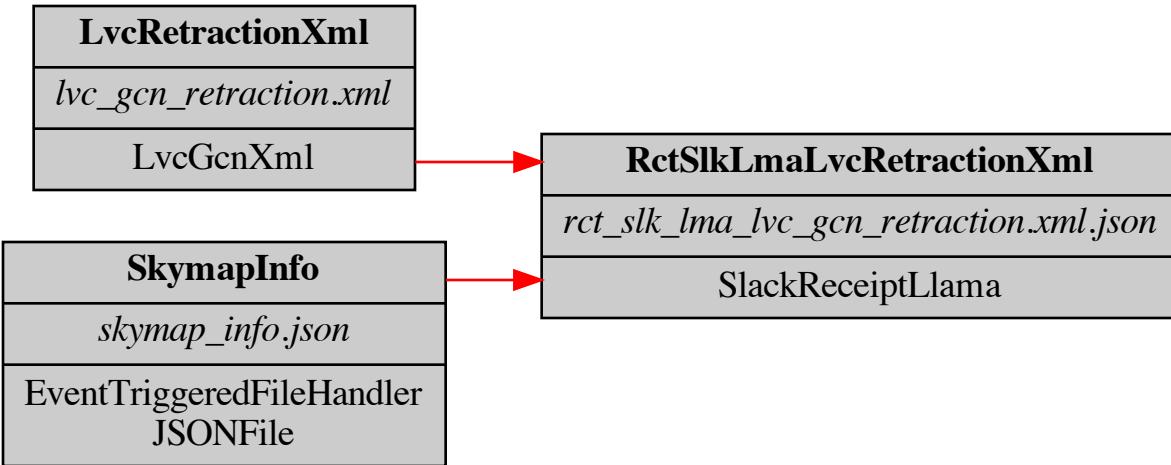


Fig. 63: Required input files for `llama.files.slack.RctSlkLmaLvcRetractionXml` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.slack.RctSlkLmaLvcRetractionXml` to be generated.

```

MANIFEST_TYPES = (<class 'llama.files.slack.RctSlkLmaLvcRetractionXml'>,)

UPLOAD
alias of llama.files.lvc\_gcn\_xml.LvcRetractionXml

UR_DEPENDENCIES = (<class 'llama.files.lvc_gcn_xml.LvcRetractionXml'>, <class
'llama.files.skymap_info.SkymapInfo'>)

UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvc_gcn_xml.LvcRetractionXml'>})

class_veto = ()

class llama.files.slack.RctSlkLmaSkymapInfo(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

```

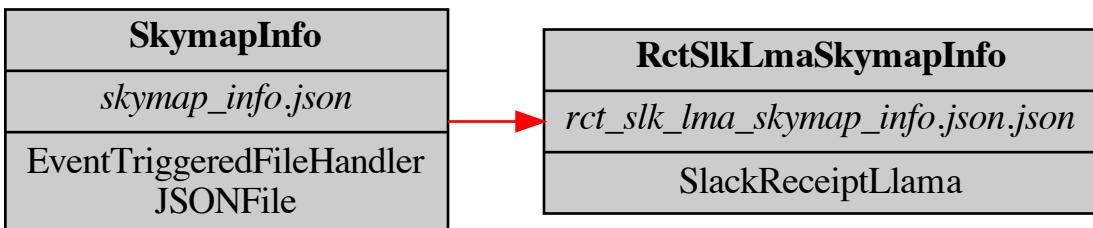


Fig. 64: Required input files for `llama.files.slack.RctSlkLmaSkymapInfo` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.slack.RctSlkLmaSkymapInfo` to be generated.

```

DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)

FILENAME = 'rct_slk_lma_skymap_info.json.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class 'llama.files.slack.RctSlkLmaSkymapInfo'>,)

UPLOAD
alias of llama.files.skymap\_info.SkymapInfo

```

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})
class_vetoes = ()

class llama.files.slack.SlackReceipt(eventid_or_fh, rundir=None)
Bases: llama.com.utils.UploadReceipt, llama.filehandler.JSONFile

A log file created after an upload attempt to Slack. Use these filehandlers to upload files to slack. Additional vetoes that turn a specific upload receipt off are specified in VETOES.

ORGANIZATION = None

property comment
    An explanation of this upload FileHandler to be printed to team slack.

classmethod set_class_attributes(subclass)
    See UploadReceipt.set\_class\_attributes; this method first sets the FILENAME_FMT and CLASSNAME_FMT attributes based on subclass.ORGANIZATION. Also adds SkymapInfo to subclass.DEPENDENCIES if it is not already a member and sets class_vetoes to the ones defined for this organization.

property upload_title
    Print a title for this upload.

class llama.files.slack.SlackReceiptIcecube(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceipt

A SlackReceipt for organization IceCube.

CLASSNAME_FMT = 'RctSlkI3{}'

ORGANIZATION = 'IceCube'

class llama.files.slack.SlackReceiptLlama(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceipt

A SlackReceipt for organization LLAMA.

CLASSNAME_FMT = 'RctSlkLma{}'

ORGANIZATION = 'LLAMA'

llama.files.slack.icecube_upload_flag_false(eventdir)
    Return whether a trigger directory has its "ICECUBE_UPLOAD" flag set to "false".

```

28.7 [llama.files.advok module](#)

A FileHandler that specifies whether an event has been flagged ADVOK, allowing information about it to be shared with EM follow up partners.

```

class llama.files.advok.Advok(eventid_or_fh, rundir=None)
Bases: llama.filehandler.EventTriggeredFileHandler, llama.filehandler.JSONFile

```

The existence of this file should be taken as an indication that the event has been marked as ADVOK. This may have occurred either through the actual application of the ADVOK label to the event on GraceDB, or through the sending of a GCN notice for this event (which has happened even when ADVOK was not applied, as in G298048), or when a LLAMA operator chooses to manually mark the event as EM followup ready by creating this file. Once this file exists, dependent files will assume that the event is legitimate and will carry on with any communication efforts they are programmed to perform.

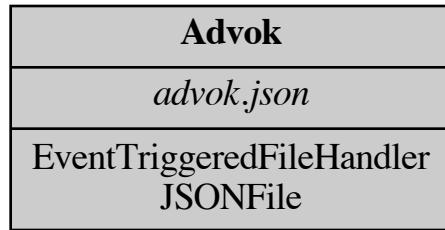


Fig. 65: `llama.files.advok`.`Advok` is created from external triggers. It therefore has no LLAMA-representable input dependencies, but instead acts as initial input for other `FileHandler` classes.

Should be created either simultaneously with a GCN notice or simultaneously with an LVAAlert adding the ADVOK label to an event (these are the use cases at time of writing, anyway).

```

FILENAME = 'advok.json'

MANIFEST_TYPES = (<class 'llama.files.advok.Advok'>,)

UR_DEPENDENCIES = ()

UR_DEPENDENCY_TREE = frozenset({})

property alert_file_handler
    Get the file handler which tells us that this event is ready for EM followup.

property alert_type
    A string describing the provenance of this trigger, i.e. what “alert” caused us to create this event (and with it, this SkymapInfo file).

property time
    The time at which this became ADVOK.
  
```

28.8 `llama.files.coinc_analyses` module

FileHandlers associated with measuring coincidences between triggers.

```

class llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3(eventid_or_fh, rundir=None)
Bases: llama.files.healpix.HEALPixPlots
  
```

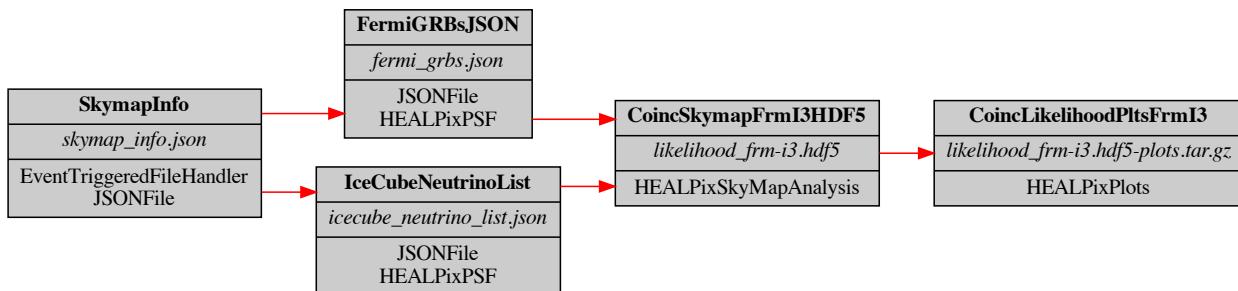


Fig. 66: Required input files for `llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3` to be generated.

Skymap plots of analysis products of FermiGRBsJSON and IceCubeNeutrinoList skymaps generated using the HEALPixSkyMapAnalysis.analysis_kernel algorithm.

```

DEPENDENCIES = (<class 'llama.files.coinc_analyses.CoincSkymapFrmI3HDF5'>,)
FILENAME = 'likelihood_frm-i3.hdf5-plots.tar.gz'
MANIFEST_TYPES = (<class 'llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3'>,
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.fermi_grb.FermiGRBsJSON'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_analyses.CoincSkymapFrmI3HDF5'>)
UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_analyses.CoincSkymapFrmI3HDF5'>})

class llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3Lvc(eventid_or_fh, rundir=None)
Bases: llama.files.healpix.HEALPixPlots

```

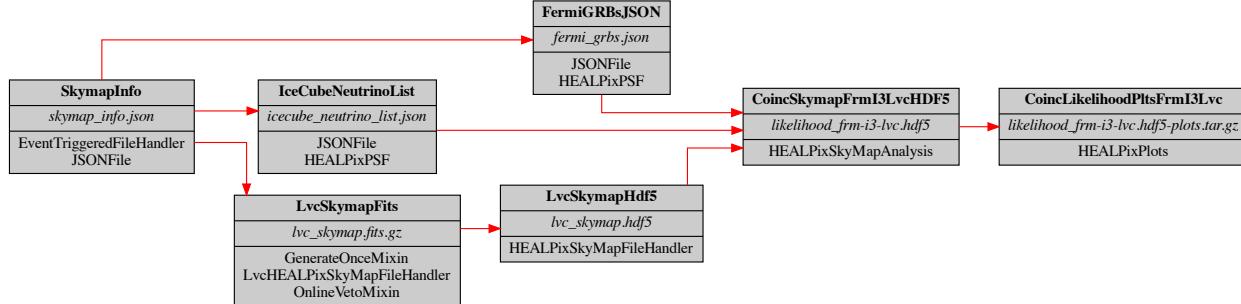


Fig. 67: Required input files for `llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3Lvc` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3Lvc` to be generated.

Skymap plots of analysis products of LvcSkymapHdf5, FermiGRBsJSON, and IceCubeNeutrinoList skymaps generated using the HEALPixSkyMapAnalysis.analysis_kernel algorithm.

```

DEPENDENCIES = (<class 'llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDF5'>,)
FILENAME = 'likelihood_frm-i3-lvc.hdf5-plots.tar.gz'
MANIFEST_TYPES = (<class 'llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3Lvc'>,
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.fermi_grb.FermiGRBsJSON'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDF5'>)
UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDF5'>})

class llama.files.coinc_analyses.CoincLikelihoodPltsFrmLvc(eventid_or_fh, rundir=None)
Bases: llama.files.healpix.HEALPixPlots

```

Skymap plots of analysis products of LvcSkymapHdf5 and FermiGRBsJSON skymaps generated using the HEALPixSkyMapAnalysis.analysis_kernel algorithm.

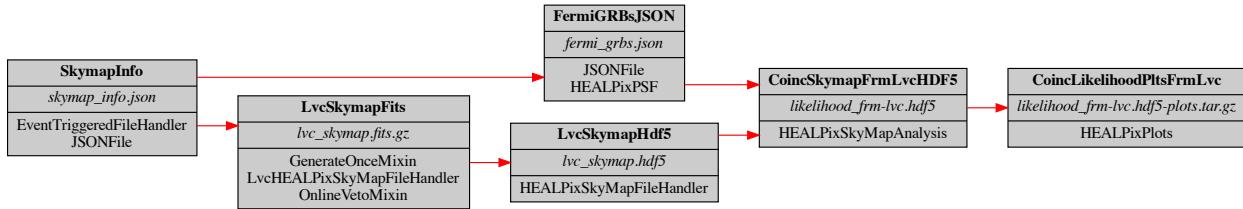


Fig. 68: Required input files for `llama.files.coinc_analyses.CoincLikelihoodPltsFrmLvc` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_analyses.CoincLikelihoodPltsFrmLvc` to be generated.

```

DEPENDENCIES = (<class 'llama.files.coinc_analyses.CoincSkymapFrmLvcHDF5'>,)
FILENAME = 'likelihood_frm-lvc.hdf5-plots.tar.gz'
MANIFEST_TYPES = (<class 'llama.files.coinc_analyses.CoincLikelihoodPltsFrmLvc'>,)
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.fermi_grb.FermiGRBsJSON'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.coinc_analyses.CoincSkymapFrmLvcHDF5'>)
UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_analyses.CoincSkymapFrmLvcHDF5'>})
class llama.files.coinc_analyses.CoincLikelihoodPltsI3Lvc(eventid_or_fh, rundir=None)
Bases: llama.files.healpix.HEALPixPlots
    
```

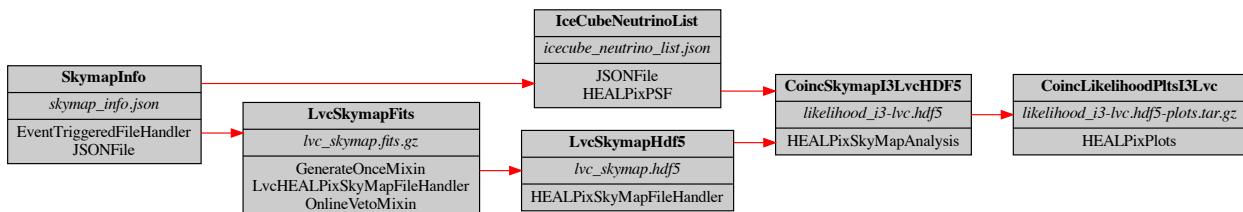


Fig. 69: Required input files for `llama.files.coinc_analyses.CoincLikelihoodPltsI3Lvc` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_analyses.CoincLikelihoodPltsI3Lvc` to be generated.

Skymap plots of analysis products of `LvcSkymapHdf5` and `IceCubeNeutrinoList` skymaps generated using the `HEALPixSkyMapAnalysis.analysis_kernel` algorithm.

```

DEPENDENCIES = (<class 'llama.files.coinc_analyses.CoincSkymapI3LvcHDF5'>,)
FILENAME = 'likelihood_i3-lvc.hdf5-plots.tar.gz'
MANIFEST_TYPES = (<class 'llama.files.coinc_analyses.CoincLikelihoodPltsI3Lvc'>,)
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_analyses.CoincSkymapI3LvcHDF5'>)
    
```

```

UR_DEPENDENCY_TREE = frozenset({<class
    'llama.files.coinc_analyses.CoincSkymapI3LvcHDF5'>})

class llama.files.coinc_analyses.CoincSkymapFrmI3HDF5(eventid_or_fh, rundir=None)
Bases: llama.files.healpix.HEALPixSkyMapAnalysis

```

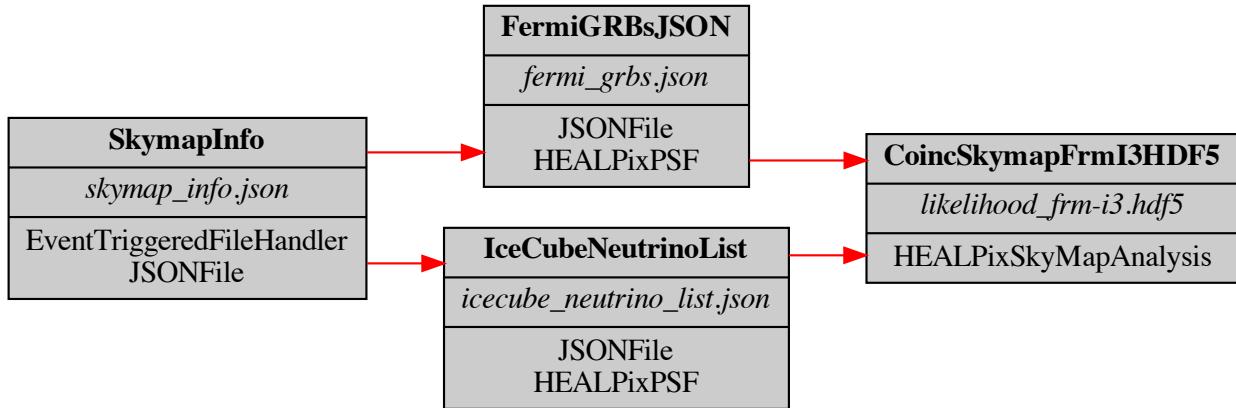


Fig. 70: Required input files for `llama.files.coinc_analyses.CoincSkymapFrmI3HDF5` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_analyses.CoincSkymapFrmI3HDF5` to be generated.

Skymap analysis products of FermiGRBsJSON and IceCubeNeutrinoList skymaps generated using the `HEALPixSkyMapAnalysis.analysis_kernel` algorithm.

```

DEPENDENCIES = (<class 'llama.files.fermi_grb.FermiGRBsJSON'>, <class
    'llama.files.i3.json.IceCubeNeutrinoList'>)

DETECTORS = (Detector(name='Fermi', abbrev='frm', fullname='Fermi', url=None,
    summary='Fermi', description='', citations=ImmutableDict({})),
    Detector(name='IceCube', abbrev='i3', fullname='IceCube',
    url='http://wiki.icecube.wisc.edu', summary='IceCube', description='',
    citations=ImmutableDict({})))

FILENAME = 'likelihood_frm-i3.hdf5'
FILENAME_PREFIX = 'likelihood'

MANIFEST_TYPES = (<class 'llama.files.coinc_analyses.CoincSkymapFrmI3HDF5'>)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
    'llama.files.fermi_grb.FermiGRBsJSON'>, <class
    'llama.files.i3.json.IceCubeNeutrinoList'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class
    'llama.files.i3.json.IceCubeNeutrinoList'>: ImmutableDict({<class
        'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}), <class
        'llama.files.fermi_grb.FermiGRBsJSON'>: ImmutableDict({<class
            'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})})})

class llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDF5(eventid_or_fh, rundir=None)
Bases: llama.files.healpix.HEALPixSkyMapAnalysis

```

Skymap analysis products of LvcSkymapHdf5, FermiGRBsJSON, and IceCubeNeutrinoList skymaps generated using the `HEALPixSkyMapAnalysis.analysis_kernel` algorithm.

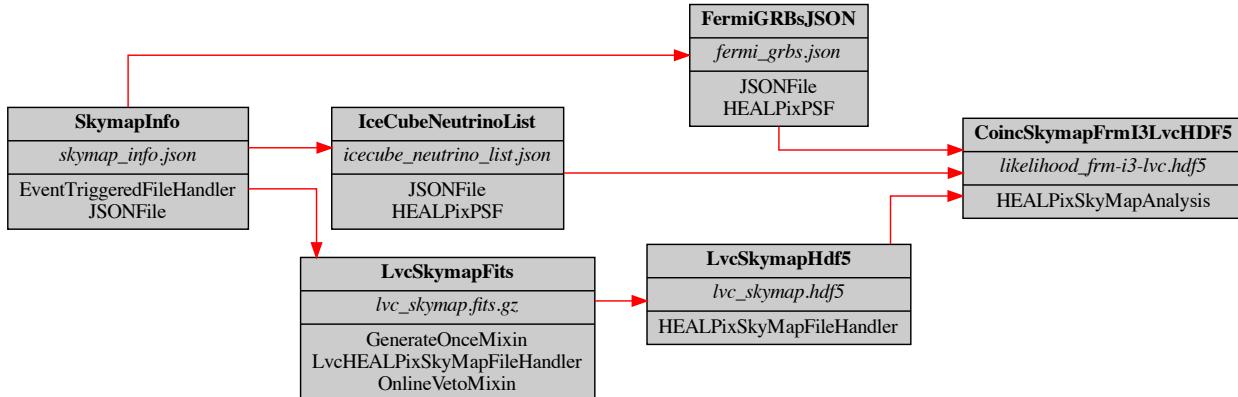


Fig. 71: Required input files for `llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDF5` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDF5` to be generated.

```

DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.fermi_grb.FermiGRBsJSON'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)

DETECTORS = (Detector(name='Fermi', abbrev='frm', fullname='Fermi', url=None,
summary='Fermi', description='', citations=ImmutableDict({})),
Detector(name='IceCube', abbrev='i3', fullname='IceCube',
url='http://wiki.icecube.wisc.edu', summary='IceCube', description='',
citations=ImmutableDict({})), Detector(name='LVC', abbrev='lvc', fullname='LVC',
url='http://wiki.ligo.org', summary='LVC', description='',
citations=ImmutableDict({})))

FILENAME = 'likelihood_frm-i3-lvc.hdf5'

FILENAME_PREFIX = 'likelihood'

MANIFEST_TYPES = (<class 'llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDF5'>)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.fermi_grb.FermiGRBsJSON'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}), <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>: ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}), <class
'llama.files.fermi_grb.FermiGRBsJSON'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})})}})

class llama.files.coinc_analyses.CoincSkymapFrmLvcHDF5(eventid_or_fh, rundir=None)
Bases: llama.files.healpix.HEALPixSkyMapAnalysis

Skymap analysis products of LvcSkymapHdf5 and FermiGRBsJSON skymaps generated using the
HEALPixSkyMapAnalysis.analysis_kernel algorithm.

```

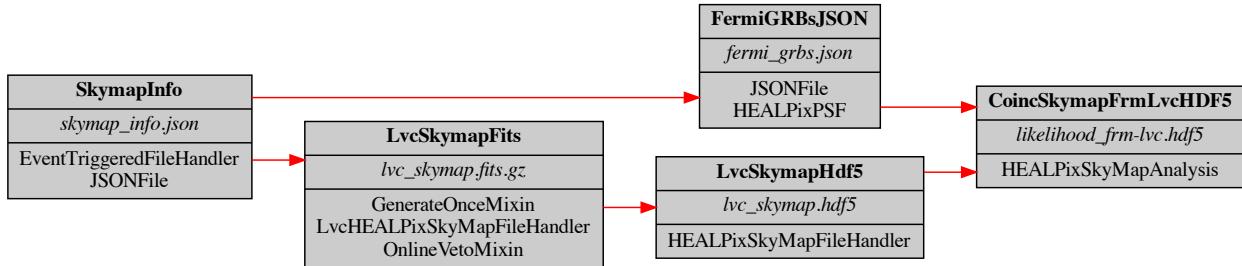


Fig. 72: Required input files for `llama.files.coinc_analyses.CoincSkymapFrmLvcHDF5` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_analyses.CoincSkymapFrmLvcHDF5` to be generated.

```

DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.fermi_grb.FermiGRBsJSON'>)

DETECTORS = (Detector(name='Fermi', abbrev='frm', fullname='Fermi', url=None,
summary='Fermi', description='', citations=ImmutableDict({})), Detector(name='LVC',
abbrev='lvc', fullname='LVC', url='http://wiki.ligo.org', summary='LVC',
description='', citations=ImmutableDict({})))

FILENAME = 'likelihood_frm-lvc.hdf5'

FILENAME_PREFIX = 'likelihood'

MANIFEST_TYPES = (<class 'llama.files.coinc_analyses.CoincSkymapFrmLvcHDF5'>)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.fermi_grb.FermiGRBsJSON'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>:
ImmutableDict({<class 'llama.files.lvc_skymap.LvcSkymapFits'>:
ImmutableDict({<class 'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}), <class
'llama.files.fermi_grb.FermiGRBsJSON'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})})}})

class llama.files.coinc_analyses.CoincSkymapI3LvcHDF5(eventid_or_fh, rundir=None)
Bases: llama.files.healpix.HEALPixSkyMapAnalysis
    
```

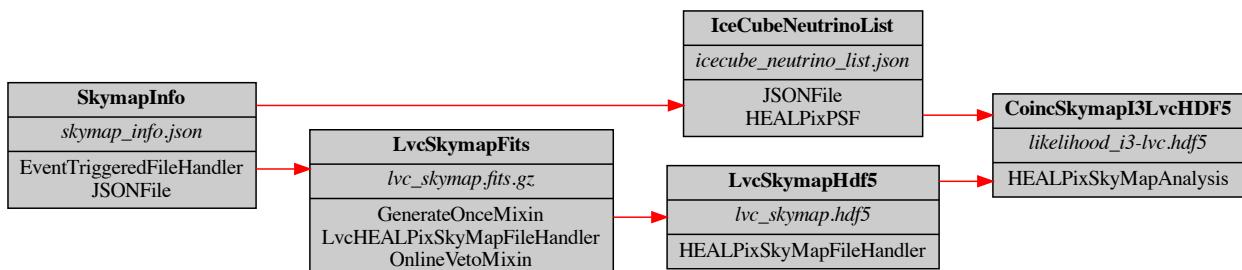


Fig. 73: Required input files for `llama.files.coinc_analyses.CoincSkymapI3LvcHDF5` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_analyses.CoincSkymapI3LvcHDF5` to be generated.

Skymap analysis products of LvcSkymapHdf5 and IceCubeNeutrinoList skymaps generated using the HEALPixSkyMapAnalysis.analysis_kernel algorithm.

```
DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class  
'llama.files.i3.json.IceCubeNeutrinoList'>)  
  
DETECTORS = (Detector(name='IceCube', abbrev='i3', fullname='IceCube',  
url='http://wiki.icecube.wisc.edu', summary='IceCube', description='',  
citations=ImmutableDict({})), Detector(name='LVC', abbrev='lvc', fullname='LVC',  
url='http://wiki.ligo.org', summary='LVC', description='',  
citations=ImmutableDict({})))  
  
FILENAME = 'likelihood_i3-lvc.hdf5'  
  
FILENAME_PREFIX = 'likelihood'  
  
MANIFEST_TYPES = (<class 'llama.files.coinc_analyses.CoincSkymapI3LvcHDF5'>,)  
  
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class  
'llama.files.lvc_skymap.LvcSkymapFits'>, <class  
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class  
'llama.files.i3.json.IceCubeNeutrinoList'>)  
  
UR_DEPENDENCY_TREE = ImmutableDict({<class  
'llama.files.i3.json.IceCubeNeutrinoList'>: ImmutableDict({<class  
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}}), <class  
'llama.files.lvc_skymap.LvcSkymapHdf5'>: ImmutableDict({<class  
'llama.files.lvc_skymap.LvcSkymapFits'>: ImmutableDict({<class  
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}})})})})})
```

28.9 `llama.files.coinc_o2` module

FileHandlers for the MATLAB coincident analysis code. Sees whether neutrinos and gravitational waves overlap and plots joint skymaps. Deprecated.

```
class llama.files.coinc_o2.CoincAnalysis(eventid_or_fh, rundir=None)  
Bases: llama.files.matlab.MATLABCallingFileHandler  
  
An abstract FileHandler class for files generated as part of the SingleGWENAnalysis pipeline.  
  
DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class  
'llama.files.lvc_skymap_mat.LvcSkymapMat'>, <class  
'llama.files.i3.json.IceCubeNeutrinoList'>)  
  
MANIFEST_TYPES = (<class 'llama.files.coinc_o2.HENlistONSOURCEIceCubeMat'>, <class  
'llama.files.coinc_o2.IceCubeNeutrinoListMat'>, <class  
'llama.files.coinc_o2.CoincAnalysis'>, <class  
'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson'>)  
  
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class  
'llama.files.lvc_skymap.LvcSkymapFits'>, <class  
'llama.files.lvc_skymap_mat.LvcSkymapMat'>, <class  
'llama.files.i3.json.IceCubeNeutrinoList'>)
```

```

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.lvc_skymap_mat.LvcSkymapMat': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}})}))})
property matlab_command
    Use MATLAB to run Single_GWHEN_Analysis for this event.
matlab_options = ['-nojvm']

class llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson(eventid_or_fh, rundir=None)
Bases: llama.files.coinc_o2.CoincAnalysis, llama.filehandler.JSONFile

```

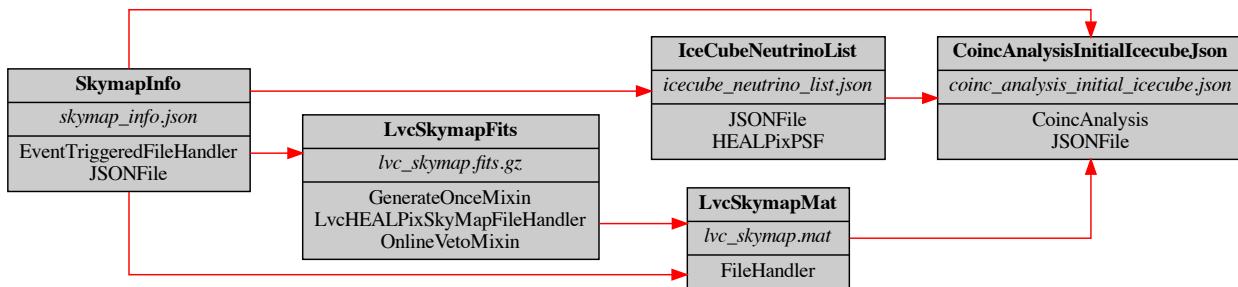


Fig. 74: Required input files for `llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson` to be generated.

A JSON document containing the quantitative results of the LLAMA coincident analysis, as documented in arXiv:1112.1140. Uses the old O2 MATLAB implementation.

```

FILENAME = 'coinc_analysis_initial_icecube.json'

MANIFEST_TYPES = (<class 'llama.files.coinc_o2.HENlistONSOURCEIceCubeMat', <class
'llama.files.coinc_o2.IceCubeNeutrinoListMat', <class
'llama.files.coinc_o2.CoincAnalysis', <class
'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson'>)

property best_neutrino
    Read the list of reconstructed neutrinos from the JSON file output by the pipeline and pick the best reconstructed neutrino. If there are no neutrinos, this property will equal None.

property reconstructed_neutrinos
    Read a list of reconstructed neutrinos from the JSON file output by the pipeline. If there are no coincident neutrinos, the list will be empty.

```

```

class llama.files.coinc_o2.CoincSkymapInitialIcecube(eventid_or_fh, rundir=None)
Bases: llama.files.matlab.MATLABCallingFileHandler

```

Plots of the LVC GW skymap with neutrinos provided by ICECUBE superimposed as point-sources. You can manually override the latitude and longitude limits by changing those variables' values before generating.

```

DEPENDENCIES = (<class 'llama.files.coinc_o2.IceCubeNeutrinoListMat', <class
'llama.files.lvc_skymap_mat.LvcSkymapMat'>)

```

```

latitude_limits = [-90, 90]
longitude_limits = [-180, 180]
property matlab_command
    Use MATLAB to run PlotGWHEN_5 for this event.

class llama.files.coinc_o2.CoincSkymap02Large(eventid_or_fh, rundir=None)
Bases: llama.files.coinc_o2.CoincSkymapInitialIcecube
    
```

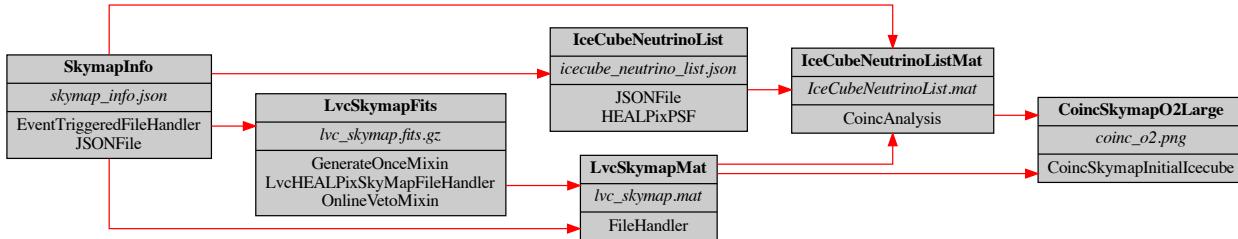


Fig. 75: Required input files for `llama.files.coinc_o2.CoincSkymap02Large` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_o2.CoincSkymap02Large` to be generated.

A larger version of the MATLAB-generated GW/HEN Coincident skymap.

```

FILENAME = 'coinc_o2.png'

MANIFEST_TYPES = (<class 'llama.files.coinc_o2.CoincSkymap02Large'>,)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.lvc_skymap_mat.LvcSkymapMat'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_o2.IceCubeNeutrinoListMat'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.coinc_o2.IceCubeNeutrinoListMat': ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}), <class
'llama.files.lvc_skymap_mat.LvcSkymapMat': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}), <class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'llama.files.lvc_skymap_mat.LvcSkymapMat': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({}), <class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})})})})})
    
```

```

class llama.files.coinc_o2.HENlistONSOURCEIceCubeMat(eventid_or_fh, rundir=None)
Bases: llama.files.coinc_o2.CoincAnalysis
    
```

A PDF plot of the LVC GW skymap with neutrinos provided by ICECUBE superimposed as point-sources. Uses the environmental variable `MATLAB_EXECUTABLE_PATH` to find a usable MATLAB binary for that part of the analysis. Default path subject to change.

```
FILENAME = 'HENlist_ONSOURCE_icecube.mat'
```

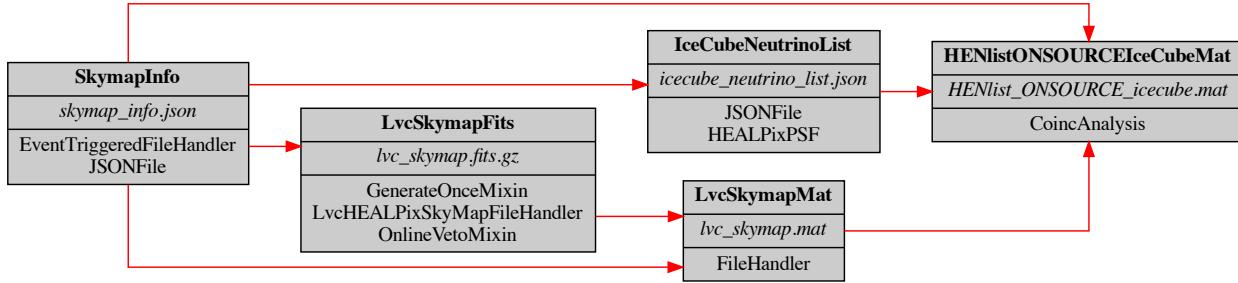


Fig. 76: Required input files for `llama.files.coinc_o2.HENlistONSOURCEIceCubeMat` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_o2.HENlistONSOURCEIceCubeMat` to be generated.

```

MANIFEST_TYPES = (<class 'llama.files.coinc_o2.HENlistONSOURCEIceCubeMat'>, <class
'llama.files.coinc_o2.IceCubeNeutrinoListMat'>, <class
'llama.files.coinc_o2.CoincAnalysis'>, <class
'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson'>)

class llama.files.coinc_o2.IceCubeNeutrinoListMat(eventid_or_fh, rundir=None)
Bases: llama.files.coinc_o2.CoincAnalysis
  
```

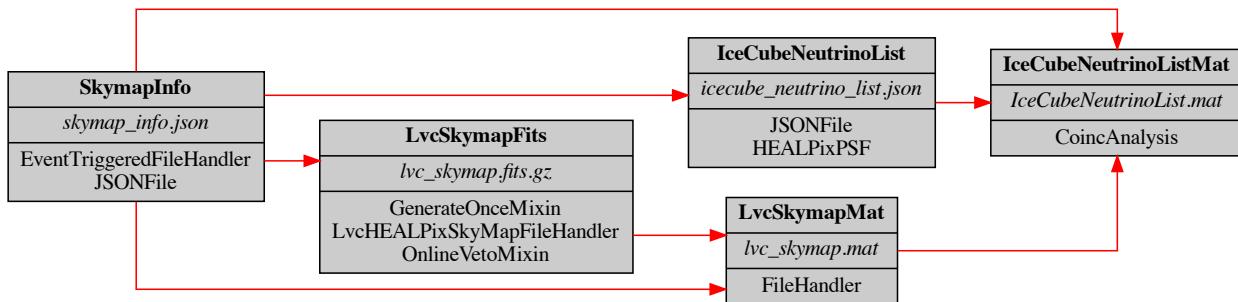


Fig. 77: Required input files for `llama.files.coinc_o2.IceCubeNeutrinoListMat` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_o2.IceCubeNeutrinoListMat` to be generated.

A MATLAB-formatted file containing Neutrino data. Used for running analysis and for generating coincidence plots.

```

FILENAME = 'IceCubeNeutrinoList.mat'

MANIFEST_TYPES = (<class 'llama.files.coinc_o2.HENlistONSOURCEIceCubeMat'>, <class
'llama.files.coinc_o2.IceCubeNeutrinoListMat'>, <class
'llama.files.coinc_o2.CoincAnalysis'>, <class
'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson'>)

class llama.files.coinc_o2.RctGdbCoincSkymap02Large(eventid_or_fh, rundir=None)
Bases: llama.files.gracedb.GraceDBReceipt
DEPENDENCIES = (<class 'llama.files.coinc_o2.CoincSkymap02Large'>,)
FILENAME = 'rct_gdb_coinc_o2.png.log'
MANIFEST_TYPES = (<class 'llama.files.coinc_o2.RctGdbCoincSkymap02Large'>,)
  
```

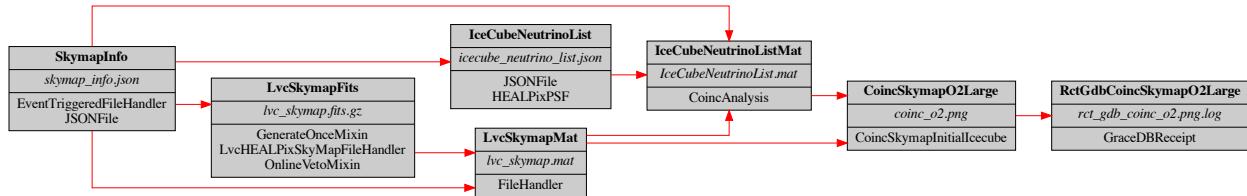


Fig. 78: Required input files for `llama.files.coinc_o2.RctGdbCoincSkymap02Large` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_o2.RctGdbCoincSkymap02Large` to be generated.

UPLOAD

alias of `llama.files.coinc_o2.CoincSkymap02Large`

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.lvc_skymap_mat.LvcSkymapMat'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_o2.IceCubeNeutrinoListMat'>, <class
'llama.files.coinc_o2.CoincSkymap02Large'>)

UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.coinc_o2.CoincSkymap02Large'>})
    
```

property log_message

GraceDB upload log message for <class 'llama.files.coinc_o2.CoincSkymap02Large'>

```

class llama.files.coinc_o2.RctGwaCoincAnalysisInitialIcecubeJson(eventid_or_fh, rundir=None)
Bases: llama.files.gwastro.GWAstroReceipt
    
```

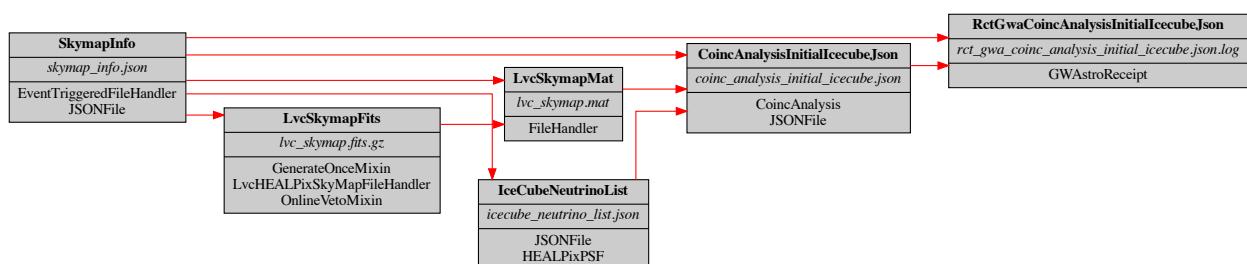


Fig. 79: Required input files for `llama.files.coinc_o2.RctGwaCoincAnalysisInitialIcecubeJson` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_o2.RctGwaCoincAnalysisInitialIcecubeJson` to be generated.

```

DEPENDENCIES = (<class 'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson'>,
<class 'llama.files.skymap_info.SkymapInfo'>)
    
```

```

FILENAME = 'rct_gwa_coinc_analysis_initial_icecube.json.log'
    
```

```

MANIFEST_TYPES = (<class
'llama.files.coinc_o2.RctGwaCoincAnalysisInitialIcecubeJson'>,)
    
```

```

RSYNC_DEST = 'gwhen@gw-astronomy.org:/home/gwhen/content/events/'
    
```

UPLOAD

alias of `llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson`

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.lvc_skymap_mat.LvcSkymapMat'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson'>})

class llama.files.coinc_o2.RctGwaCoincSkymap02Large(eventid_or_fh, rundir=None)
Bases: llama.files.gwastro.GWAstroReceipt
    
```

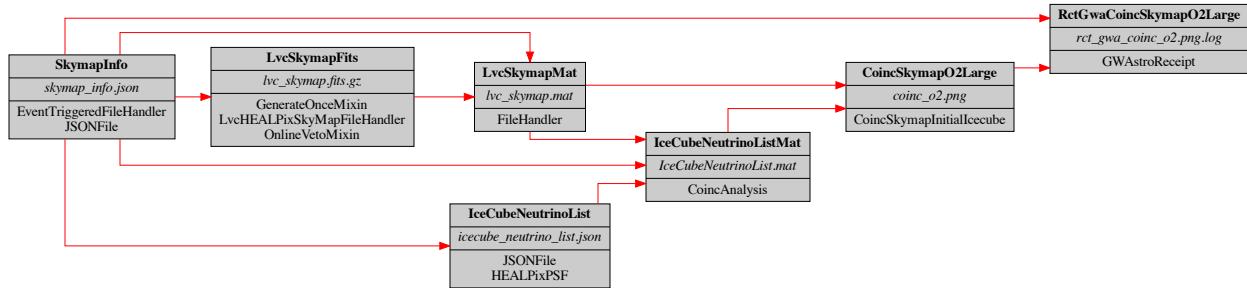


Fig. 80: Required input files for `llama.files.coinc_o2.RctGwaCoincSkymap02Large` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_o2.RctGwaCoincSkymap02Large` to be generated.

```

DEPENDENCIES = (<class 'llama.files.coinc_o2.CoincSkymap02Large'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_gwa_coinc_o2.png.log'

MANIFEST_TYPES = (<class 'llama.files.coinc_o2.RctGwaCoincSkymap02Large'>,)

RSYNC_DEST = 'gwhen@gw-astronomy.org:/home/gwhen/content/events/'

UPLOAD
    alias of llama.files.coinc_o2.CoincSkymap02Large

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.lvc_skymap_mat.LvcSkymapMat'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_o2.IceCubeNeutrinoListMat'>, <class
'llama.files.coinc_o2.CoincSkymap02Large'>)

UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.coinc_o2.CoincSkymap02Large'>})

class llama.files.coinc_o2.RctTeamCoincAnalysisInitialIcecubeJson(eventid_or_fh, rundir=None)
Bases: llama.files.team_receipts.TeamEmailReceipt

DEPENDENCIES = (<class 'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson'>,)

FILENAME = 'rct_team_coinc_analysis_initial_icecube.json.log'

MANIFEST_TYPES = (<class
'llama.files.coinc_o2.RctTeamCoincAnalysisInitialIcecubeJson'>,)

UPLOAD
    alias of llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson
    
```

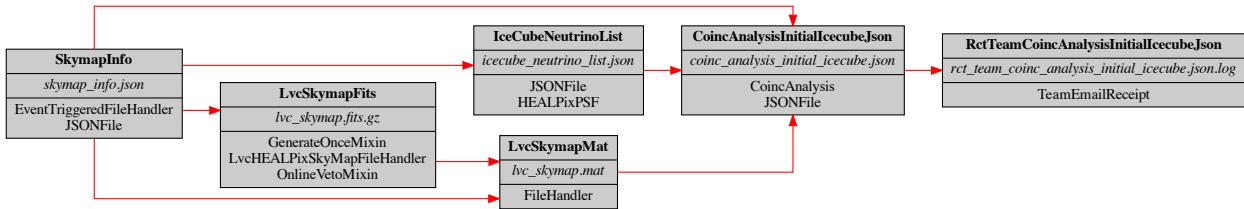


Fig. 81: Required input files for `llama.files.coinc_o2.RctTeamCincAnalysisInitialIcecubeJson` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_o2.RctTeamCincAnalysisInitialIcecubeJson` to be generated.

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.lvc_skymap_mat.LvcSkymapMat'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson'>)})

property subject
    Email subject for <class 'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson'> email upload.

```

28.10 `llama.files.coinc_plots` module

Human-readable plotters for coincident skymaps showing overlapping events

```

class llama.files.coinc_plots.CoincScatterI3Lvc(eventid_or_fh, rundir=None)
    Bases: llama.files.healpix.plotters.JointSkymapScatter

Abstract class for a human-viewable joint Neutrino/Gravitational Wave skymap plot. Subclass this with an output file extension to make a working FileHandler.

BACKGROUND_SKYMAP
    alias of llama.files.lvc\_skymap.LvcSkymapHdf5

POINT_SOURCES = frozenset({<class 'llama.files.i3.json.IceCubeNeutrinoList'>)})

class llama.files.coinc_plots.CoincScatterI3LvcPdf(eventid_or_fh, rundir=None)
    Bases: llama.files.coinc\_plots.CoincScatterI3Lvc

Human-readable joint Neutrino/Gravitational Wave skymap plot in PDF format.

DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)

FILEEXT = 'pdf'

FILENAME = 'scatterplot_lvc-i3.pdf'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincScatterI3LvcPdf'>,)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)

```

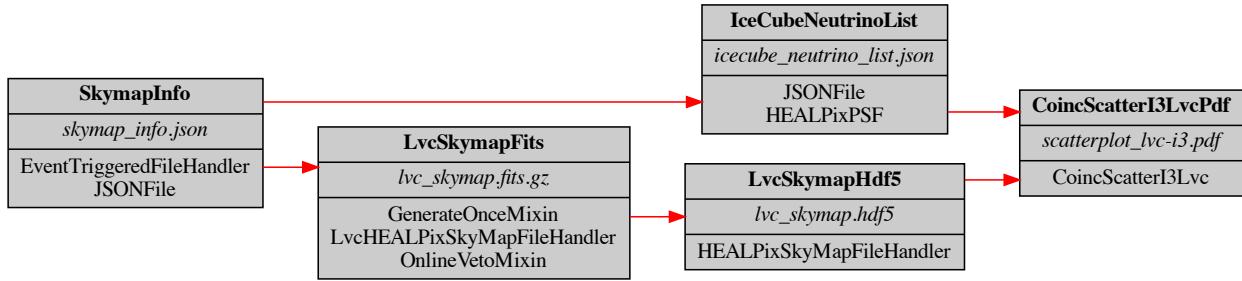


Fig. 82: Required input files for `llama.files.coinc_plots.CoincScatterI3LvcPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincScatterI3LvcPdf` to be generated.

```

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})}), <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({})})})})
  
```

```

class llama.files.coinc_plots.CoincScatterI3LvcPng(eventid_or_fh, rundir=None)
  
```

Bases: `llama.files.coinc_plots.CoincScatterI3Lvc`

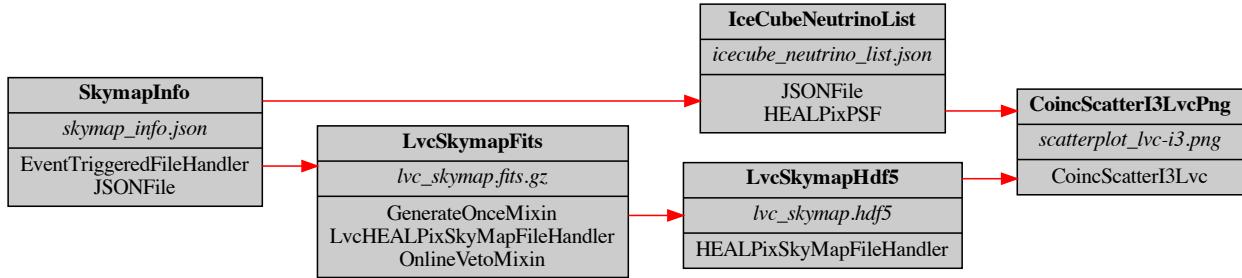


Fig. 83: Required input files for `llama.files.coinc_plots.CoincScatterI3LvcPng` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincScatterI3LvcPng` to be generated.

Human-readable joint Neutrino/Gravitational Wave skymap plot in PNG format.

```

DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'	llama.files.i3.json.IceCubeNeutrinoList'>)

FILEEXT = 'png'

FILENAME = 'scatterplot_lvc-i3.png'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincScatterI3LvcPng'>,)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'	llama.files.lvc_skymap.LvcSkymapFits'>, <class
'	llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'	llama.files.i3.json.IceCubeNeutrinoList'>)
  
```

```

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}), <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}})})})})
})

class llama.files.coinc_plots.CoincScatterZtfI3Lvc(eventid_or_fh, rundir=None)
Bases: llama.files.healpix.plotters.JointSkymapScatter

Abstract class for a human-viewable joint ZTF/Gravitational Wave/High Energy Neutrino joint skymap plot.
Subclass this with an output file extension to make a working FileHandler.

BACKGROUND_SKYMAP
    alias of llama.files.lvc_skymap.LvcSkymapHdf5

POINT_SOURCES = ImmutableDict({<class 'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({'marker': '$\\times$', 'color': 'green'}), <class
'llama.files.ztf_trigger_list.ZtfTriggerList': ImmutableDict({'marker': '$\\times$', 'color': 'red'})})

class llama.files.coinc_plots.CoincScatterZtfI3LvcPdf(eventid_or_fh, rundir=None)
Bases: llama.files.coinc_plots.CoincScatterZtfI3Lvc

```

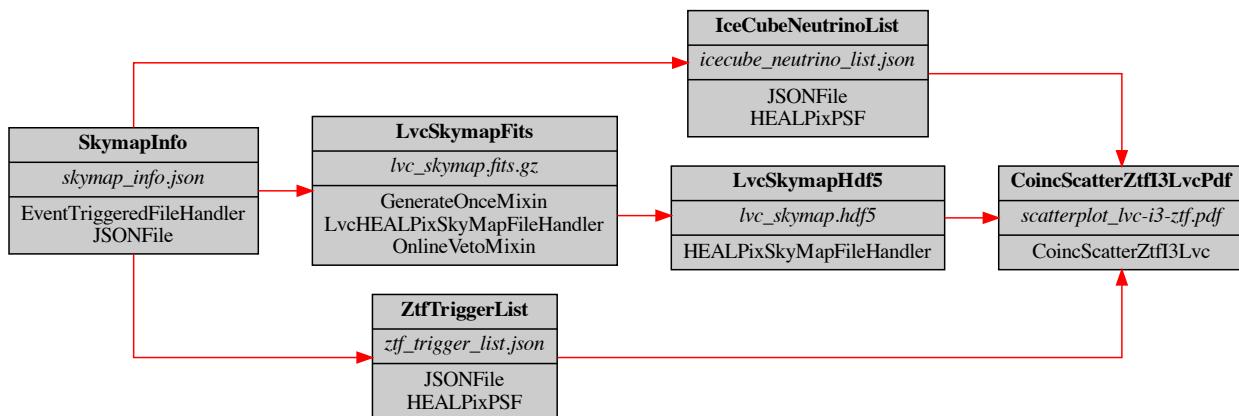


Fig. 84: Required input files for `llama.files.coinc_plots.CoincScatterZtfI3LvcPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincScatterZtfI3LvcPdf` to be generated.

Human-readable joint ZTF/Gravitational Wave/High Energy Neutrino skymap plot in PDF format.

```

DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5', <class
'	llama.files.i3.json.IceCubeNeutrinoList', <class
'	llama.files.ztf_trigger_list.ZtfTriggerList')

FILEEXT = 'pdf'
FILENAME = 'scatterplot_lvc-i3-ztf.pdf'
MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincScatterZtfI3LvcPdf',)

```

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{})), <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>: ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}))}), <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}))})})

class llama.files.coinc_plots.CoincScatterZtfI3LvcPng(eventid_or_fh, rundir=None)
Bases: llama.files.coinc_plots.CoincScatterZtfI3Lvc

```

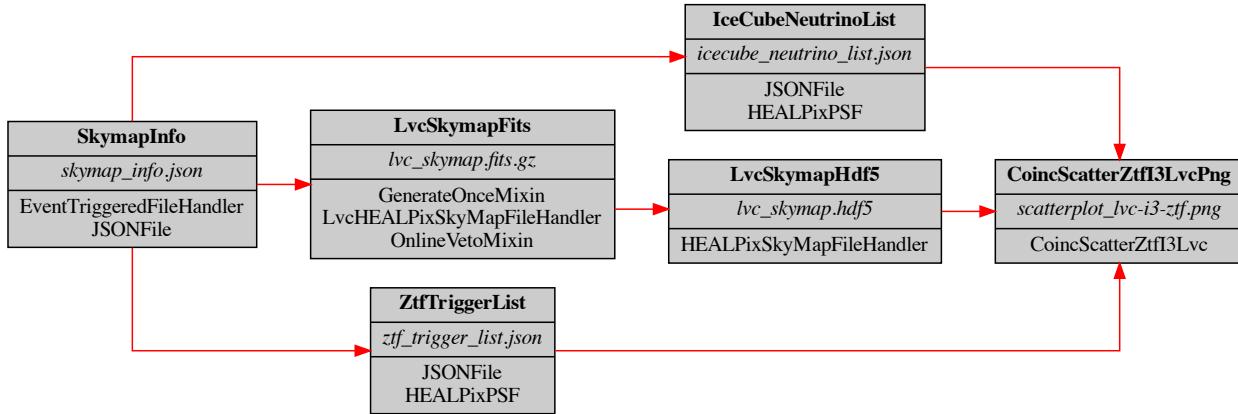


Fig. 85: Required input files for `llama.files.coinc_plots.CoincScatterZtfI3LvcPng` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincScatterZtfI3LvcPng` to be generated.

Human-readable joint ZTF/Gravitational Wave/High Energy Neutrino skymap plot in PNG format.

```

DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>)

FILEEXT = 'png'

FILENAME = 'scatterplot_lvc-i3-ztf.png'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincScatterZtfI3LvcPng'>)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)

```

```

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}})}), <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}})}), <class
'llama.files.ztf_trigger_list.ZtfTriggerList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}})})})

class llama.files.coinc_plots.CoincScatterZtfLVC(eventid_or_fh, rundir=None)
Bases: llama.files.healpix.plotters.JointSkymapScatter

Abstract class for a human-viewable joint ZTF/Gravitational Wave skymap plot. Subclass this with an output file extension to make a working FileHandler.

BACKGROUND_SKYMAP
alias of llama.files.lvc_skymap.LvcSkymapHdf5

POINT_SOURCES = frozenset({<class 'llama.files.ztf_trigger_list.ZtfTriggerList'>)}

class llama.files.coinc_plots.CoincScatterZtfLVCPdf(eventid_or_fh, rundir=None)
Bases: llama.files.coinc_plots.CoincScatterZtfLVC

```

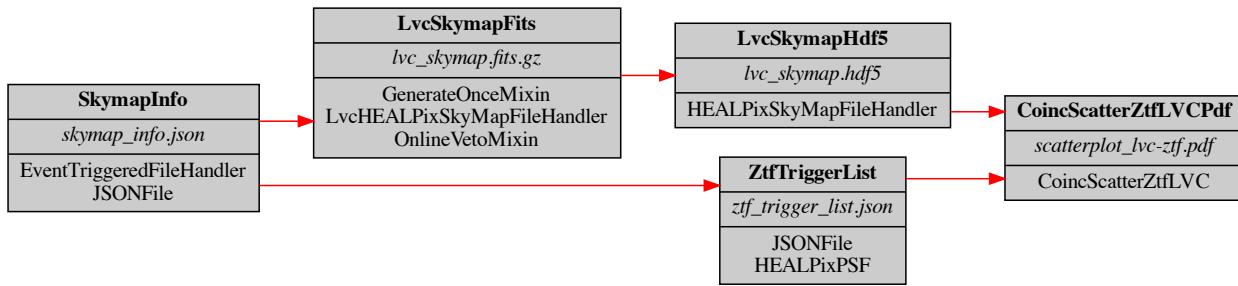


Fig. 86: Required input files for `llama.files.coinc_plots.CoincScatterZtfLVCPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincScatterZtfLVCPdf` to be generated.

Human-readable joint ZTF/Gravitational Wave skymap plot in PDF format.

```

DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'	llama.files.ztf_trigger_list.ZtfTriggerList'>)

FILEEXT = 'pdf'

FILENAME = 'scatterplot_lvc-ztf.pdf'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincScatterZtfLVCPdf'>)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'	llama.files.lvc_skymap.LvcSkymapFits'>, <class
'	llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'	llama.files.lvc_skymap.LvcSkymapHdf5'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>:
ImmutableDict({<class 'llama.files.lvc_skymap.LvcSkymapFits'>:
ImmutableDict({<class 'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}})}),
<class 'llama.files.ztf_trigger_list.ZtfTriggerList'>: ImmutableDict({<class
'	llama.files.skymap_info.SkymapInfo'>: ImmutableDict({{}})})})

```

```
class llama.files.coinc_plots.CoincScatterZtfLVCPng(eventid_or_fh, rundir=None)
Bases: llama.files.coinc_plots.CoincScatterZtfLVC
```

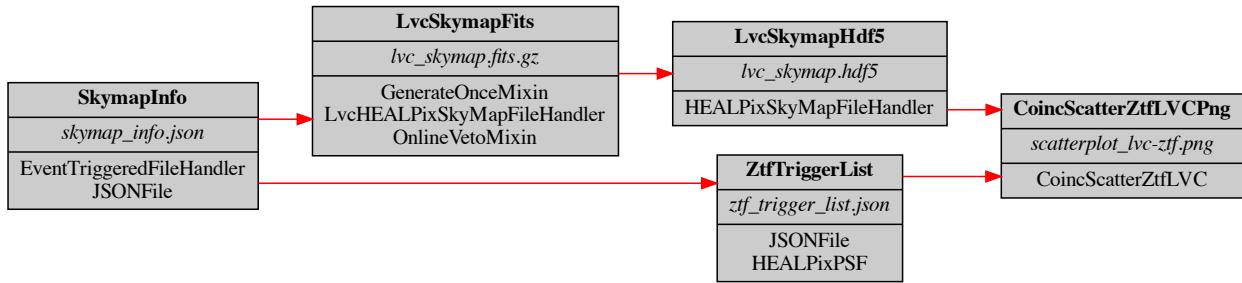


Fig. 87: Required input files for `llama.files.coinc_plots.CoincScatterZtfLVCPng` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincScatterZtfLVCPng` to be generated.

Human-readable joint ZTF/Gravitational Wave skymap plot in PNG format.

```
DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>)

FILEEXT = 'png'

FILENAME = 'scatterplot_lvc-ztf.png'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincScatterZtfLVCPng'>,)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class 'llama.files.lvc_skymap.LvcSkymapHdf5'>:
ImmutableDict({<class 'llama.files.lvc_skymap.LvcSkymapFits'>:
ImmutableDict({<class 'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})}),},
<class 'llama.files.ztf_trigger_list.ZtfTriggerList'>: ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})})})
```

```
class llama.files.coinc_plots.CoincSummaryI3LvcPdf(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler
```

Human-viewable joint Neutrino/Gravitational Wave skymap plot complete with a list of neutrinos and their properties in PDF form. Useful for quick human checks of the output of the pipeline on an event.

```
DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincSummaryI3LvcTex'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>)

FILENAME = 'summary_lvc-I3.pdf'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincSummaryI3LvcPdf'>,)
```

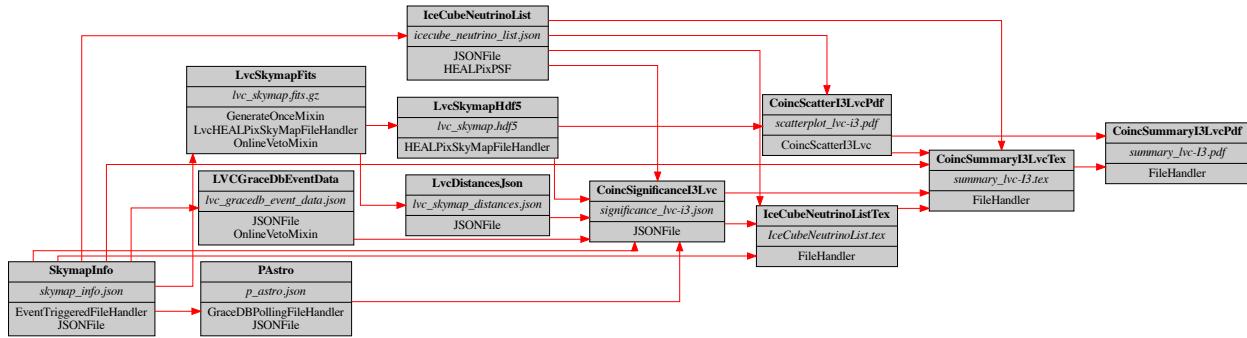


Fig. 88: Required input files for `llama.files.coinc_plots.CoincSummaryI3LvcPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincSummaryI3LvcPdf` to be generated.

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>, <class
'llama.files.i3.tex.IceCubeNeutrinoListTex'>, <class
'llama.files.coinc_plots.CoincSummaryI3LvcTex'>)
  
```

```
class llama.files.coinc_plots.CoincSummaryI3LvcTex(eventid_or_fh, rundir=None)
    Bases: llama\_filehandler\_FileHandler
```

LaTeX file for a human-viewable joint Neutrino/Gravitational Wave skymap plot complete with a list of neutrinos and their properties. Gets compiled to a PDF file.

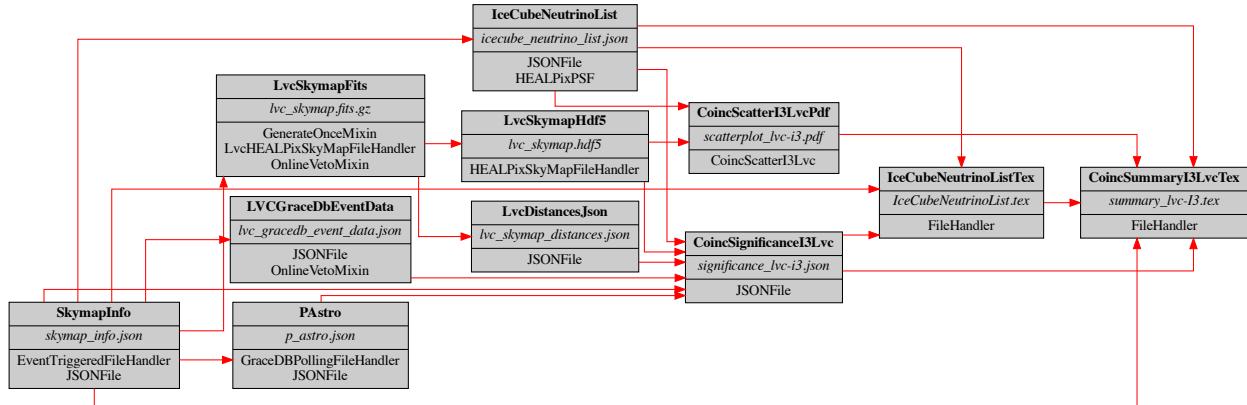


Fig. 89: Required input files for `llama.files.coinc_plots.CoincSummaryI3LvcTex` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.CoincSummaryI3LvcTex` to be generated.

```

DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>, <class
'llama.files.i3.tex.IceCubeNeutrinoListTex'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>)

FILENAME = 'summary_lvc-I3.tex'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.CoincSummaryI3LvcTex'>)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>, <class
'llama.files.i3.tex.IceCubeNeutrinoListTex'>)
  
```

```

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.i3.tex.IceCubeNeutrinoListTex': ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc': ImmutableDict({<class
'llama.files.gracedb.LVCGraceDbEventData': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}), <class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}), <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.lvc_skymap.LvcDistancesJson': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.gracedb.PAstro': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf': ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc': ImmutableDict({<class
'llama.files.gracedb.LVCGraceDbEventData': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.lvc_skymap.LvcDistancesJson': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class
'llama.files.gracedb.PAstro': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}}, <class

class llama.files.coinc_plots.RctSlkI3CoincSummaryI3LvcPdf(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptIcecube

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincSummaryI3LvcPdf', <class
'	llama.files.skymap_info.SkymapInfo')

FILENAME = 'rct_slk_i3_summary_lvc-I3.pdf.json'
FILENAME_FMT = 'rct_slk_i3_{}.json'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.RctSlkI3CoincSummaryI3LvcPdf',)

UPLOAD
alias of llama.files.coinc_plots.CoincSummaryI3LvcPdf

```

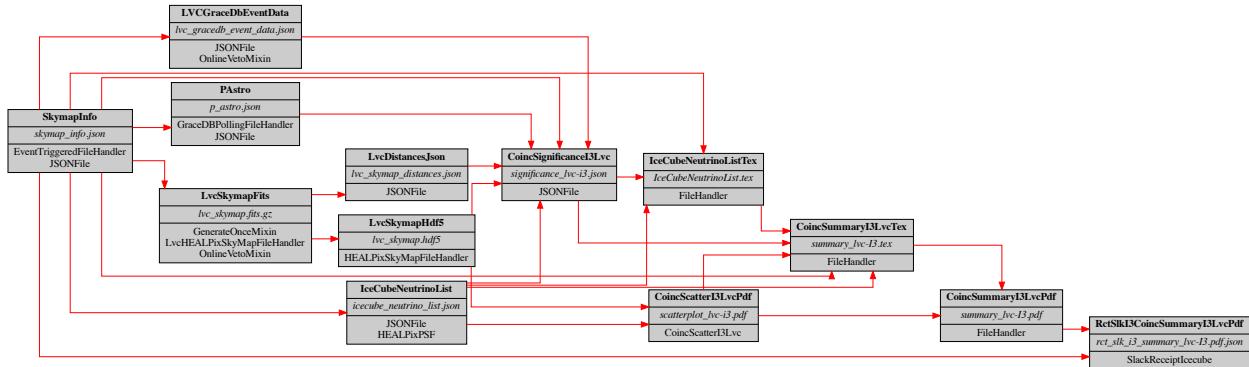


Fig. 90: Required input files for `llama.files.coinc_plots.RctSlkI3CoincSummaryI3LvcPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkI3CoincSummaryI3LvcPdf` to be generated.

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>, <class
'llama.files.i3.tex.IceCubeNeutrinoListTex'>, <class
'llama.files.coinc_plots.CoincSummaryI3LvcTex'>, <class
'llama.files.coinc_plots.CoincSummaryI3LvcPdf'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincSummaryI3LvcPdf'>)})

class_vetoes = ((<function icecube_upload_flag_false>, None),)

class llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPdf(eventid_or_fh, rundir=None)
    Bases: llama.files.slack.SlackReceiptLlama
    
```

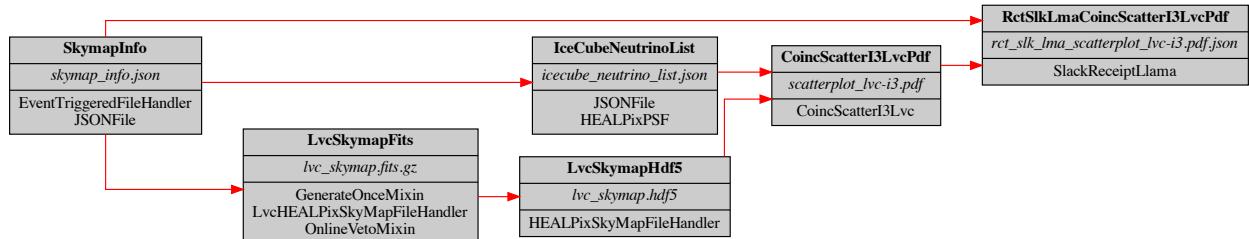


Fig. 91: Required input files for `llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPdf` to be generated.

```

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincScatterI3LvcPdf'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_scatterplot_lvc-i3.pdf.json'
    
```

```

FILENAME_FMT = 'rct_slk_lma_{}.json'
MANIFEST_TYPES = (<class 'llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPdf'>,)
UPLOAD
alias of llama.files.coinc\_plots.CoincScatterI3LvcPdf

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>)})

class_vetoies = ()

class llama.files.coinc\_plots.RctSlkLmaCoincScatterI3LvcPng(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

```

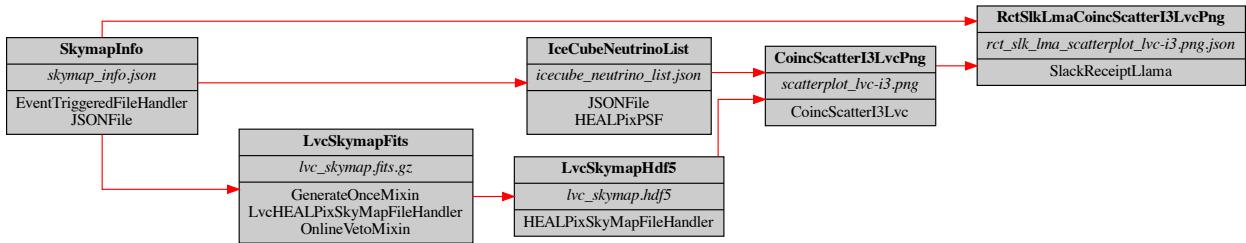


Fig. 92: Required input files for `llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPng` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPng` to be generated.

```

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincScatterI3LvcPng'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_scatterplot_lvc-i3.png'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPng'>,)

UPLOAD
alias of llama.files.coinc\_plots.CoincScatterI3LvcPng

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPng'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincScatterI3LvcPng'>)})

class_vetoies = ()

class llama.files.coinc\_plots.RctSlkLmaCoincScatterZtfI3LvcPdf(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

```

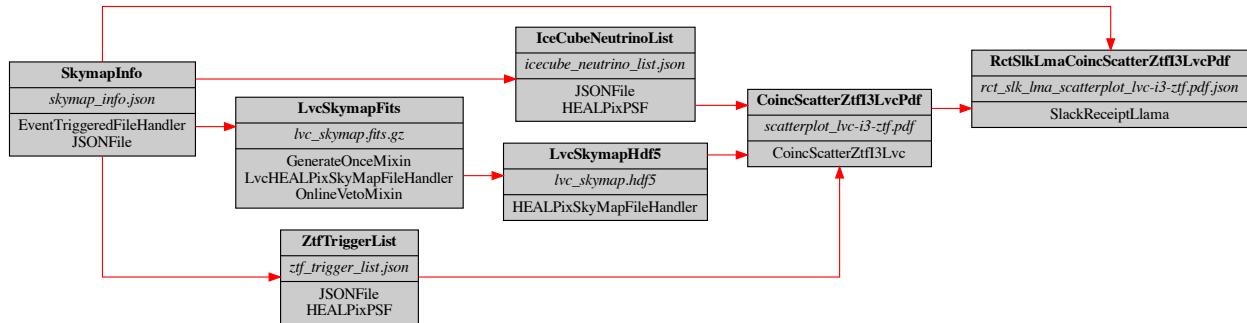


Fig. 93: Required input files for `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPdf` to be generated.

```

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincScatterZtfI3LvcPdf'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_scatterplot_lvc-i3-ztf.pdf.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class
'llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPdf'>,)

UPLOAD
    alias of llama.files.coinc\_plots.CoincScatterZtfI3LvcPdf

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_plots.CoincScatterZtfI3LvcPdf'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincScatterZtfI3LvcPdf'>)})

class_vetoes = ()

class llama.files.coinc\_plots.RctSlkLmaCoincScatterZtfI3LvcPng(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincScatterZtfI3LvcPng'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_scatterplot_lvc-i3-ztf.png.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class
'llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng'>,)

UPLOAD
    alias of llama.files.coinc\_plots.CoincScatterZtfI3LvcPng

```

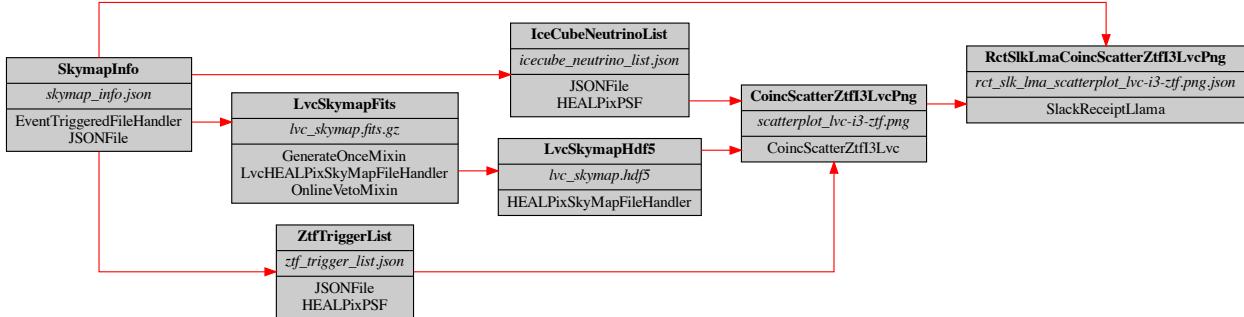


Fig. 94: Required input files for `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng` to be generated.

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.coinc_plots.CoincScatterZtfI3LvcPng'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincScatterZtfI3LvcPng'>)})

class_vetoes = ()

class llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPdf(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama
  
```

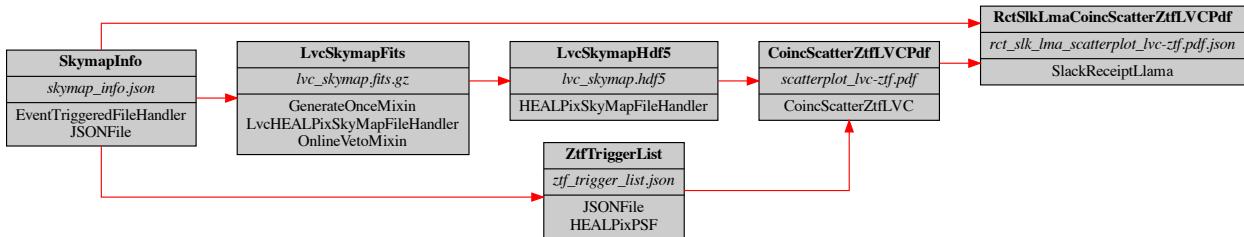


Fig. 95: Required input files for `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPdf` to be generated.

```

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincScatterZtfLVCPdf'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_scatterplot_lvc-ztf.pdf.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPdf'>,)

UPLOAD
alias of llama.files.coinc_plots.CoincScatterZtfLVCPdf
  
```

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.coinc_plots.CoincScatterZtfLVCPdf'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincScatterZtfLVCPdf'>)})

class_veto = ()

class llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPng(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

```

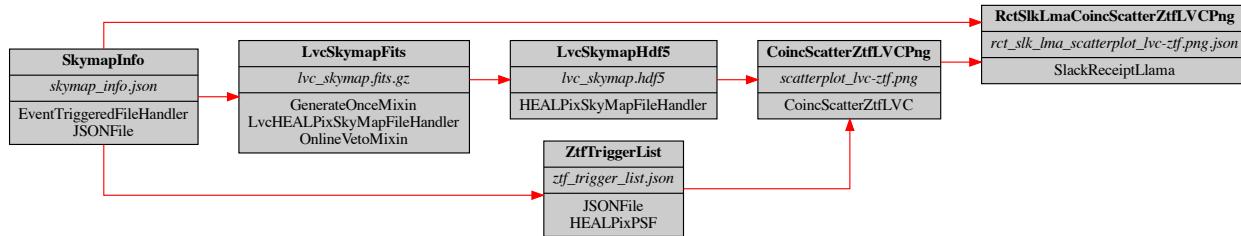


Fig. 96: Required input files for `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPng` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPng` to be generated.

```

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincScatterZtfLVCPng'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_scatterplot_lvc-ztf.png.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPng'>,) 

UPLOAD
    alias of llama.files.coinc\_plots.CoincScatterZtfLVCPng

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.ztf_trigger_list.ZtfTriggerList'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.coinc_plots.CoincScatterZtfLVCPng'>)

UR_DEPENDENCY_TREE = frozenset({<class
'llama.files.coinc_plots.CoincScatterZtfLVCPng'>)})

class_veto = ()

class llama.files.coinc_plots.RctSlkLmaCoincSummaryI3LvcPdf(eventid_or_fh, rundir=None)
Bases: llama.files.slack.SlackReceiptLlama

DEPENDENCIES = (<class 'llama.files.coinc_plots.CoincSummaryI3LvcPdf'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_summary_lvc-I3.pdf.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class 'llama.files.coinc_plots.RctSlkLmaCoincSummaryI3LvcPdf'>,) 

```

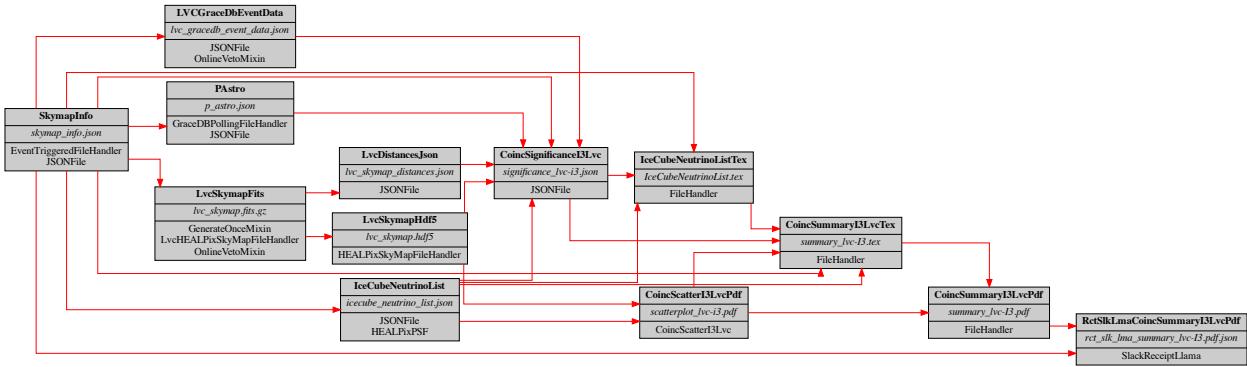


Fig. 97: Required input files for `llama.files.coinc_plots.RctSlkLmaCoincSummaryI3LvcPdf` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.coinc_plots.RctSlkLmaCoincSummaryI3LvcPdf` to be generated.

UPLOAD

alias of `llama.files.coinc_plots.CoincSummaryI3LvcPdf`

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skytmap.LvcSkymapFits'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skytmap.LvcDistancesJson'>, <class
'llama.files.lvc_skytmap.LvcSkymapHdf5'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>, <class
'llama.files.coinc_plots.CoincScatterI3LvcPdf'>, <class
'llama.files.i3.tex.IceCubeNeutrinoListTex'>, <class
'llama.files.coinc_plots.CoincSummaryI3LvcTex'>, <class
'llama.files.coinc_plots.CoincSummaryI3LvcPdf'>)

UR_DEPENDENCY_TREE = frozenset({<class
'	llama.files.coinc_plots.CoincSummaryI3LvcPdf'>)})

class_vetoes = ()

```

28.11 `llama.files.fermi_grb` module

Defines a FileHandler for generating lists of GRBs for an LVC GW trigger (or another trigger at a given time) as well as methods for obtaining historical GRB events from stored lists.

```

class llama.files.fermi_grb.FermiGRBsJSON(eventid_or_fh, rundir=None)
Bases: llama.filehandler.JSONFile, llama.files.healpix.skymap.HEALPixPSF

```

New and historical Fermi GRB triggers saved in a JSON file. Triggers are in a list of JSON objects, where each trigger will have (at least) the following properties:

- ra** [float] Right ascension in degrees.
- dec** [float] Declination in degrees.
- sigma** [float] Angular uncertainty in degrees.
- gps_time** [float] GPS time of the trigger.

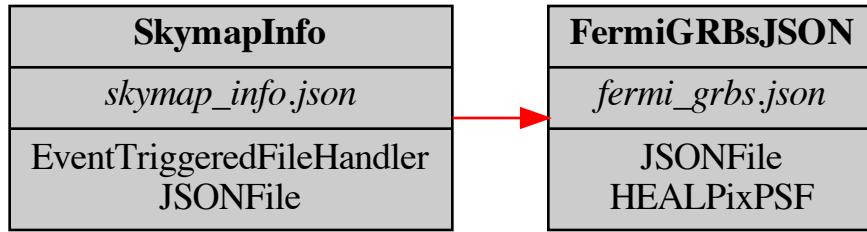


Fig. 98: Required input files for `llama.files.fermi_grb.FermiGRBsJSON` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.fermi_grb.FermiGRBsJSON` to be generated.

Additional properties might include:

`graceid` [str] GraceID of the event, if available.

`skymap_url` [str] URL of a FITS skymap, if available.

`DEPENDENCIES` = (<class '`llama.files.skymap_info.SkymapInfo`'>,)

`DETECTORS` = (`Detector(name='Fermi', abbrev='frm', fullname='Fermi', url=None, summary='Fermi', description='', citations=ImmutableDict({}))`,)

`FILENAME` = '`fermi_grbs.json`'

`MANIFEST_TYPES` = (<class '`llama.files.fermi_grb.FermiGRBsJSON`'>,)

`UR_DEPENDENCIES` = (<class '`llama.files.skymap_info.SkymapInfo`'>,)

`UR_DEPENDENCY_TREE` = `frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})`

`property num_triggers`

The number of triggers described by this file. Useful mostly for quickly determining if this trigger list is empty.

`source_locations()`

Returns a list of tuples of (RA, Dec, sigma) for all point-like sources, where (RA, Dec) is the central sky position and sigma is the standard deviation of the Gaussian PSF. Since this depends on the structure of data in the relevant file handler, it must be specified for each subclass.

`property template_skymap_filehandler`

The HEALPixSkyMapFileHandler whose HEALPix parameters should be used as a template for the HEALPixSkyMap output by this FileHandler. It is assumed that this HEALPixSkyMapFileHandler only returns one skymap in its list; consequently, only the first skymap loaded by this file handler will be used. Note that the template skymap file handler is likely not a dependency of this file handler; it is only used as a default for formatting generated skymaps from this file handler's data.

`llama.files.fermi_grb.get_grbs_from_csv(start_time, end_time)`

Use the `get_grbs_from_csv(start_time, end_time)` function to obtain a list of dictionaries of past Gamma-Ray Bursts (GRB)s. The contents of each GRB dictionary are:

`Parameters`

- `start_time` (int or float) – The start time in GPS seconds of the interval in which to search for GRBs.
- `end_time` (int or float) – The end time of the search interval, also in GPS seconds.

`Returns`

`grbs` – GRB events in the given time window, where each GRB is a dictionary:

```
{
    'ra': ra, # Right Ascension of the GRB in [deg]
    'dec': dec, # Declination of the GRB [deg]
    'sigma': sigma, # Standard deviation of the GRB location [deg]
    'time': gps_time, # GPS time of the GRB event [deg]
}
```

Return type list

Examples

```
>>> grbs = get_grbs_from_csv(1126089475.0, 1126200000.0)
>>> sorted(grbs[0])
['dec', 'ra', 'sigma', 'time']
>>> [g['dec'] for g in grbs]
[-21.0339, 73.26, -53.79]
>>> [g['ra'] for g in grbs]
[248.4429, 321.36, 241.05]
>>> [g['sigma'] for g in grbs]
[0.033, 6.38, 1.95]
>>> [g['time'] for g in grbs]
[1126089476.0, 1126103089.0, 1126151534.0]
```

28.12 llama.files.gcn_draft_o2 module

Define a FileHandler that makes draft human-readable GCN Circulars for events.

```
class llama.files.gcn_draft_o2.I3LvcGcnDraft(eventid_or_fh, rundir=None)
Bases: llama.filehandler.GenerateOnceMixin, llama.filehandler.FileHandler
```

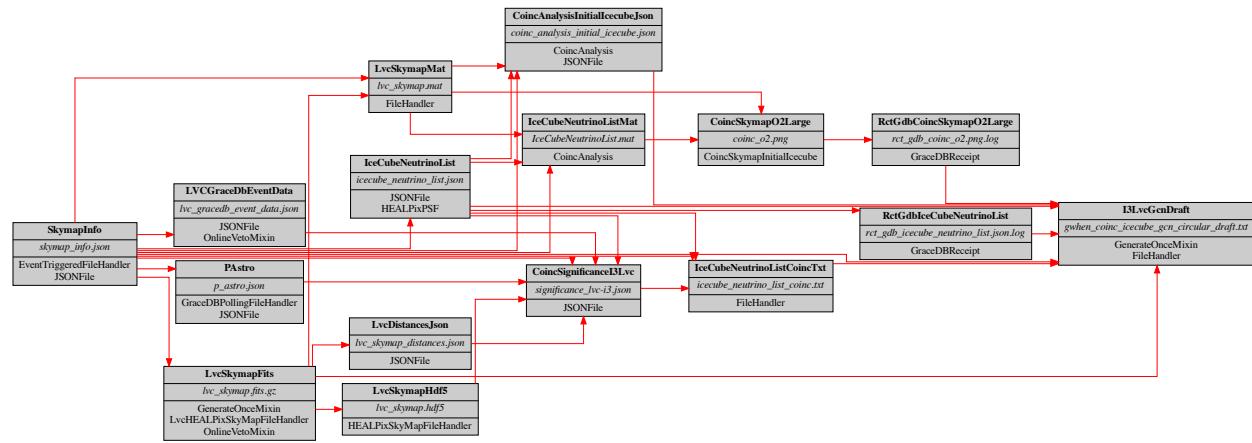


Fig. 99: Required input files for `llama.files.gcn_draft_o2.I3LvcGcnDraft` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.gcn_draft_o2.I3LvcGcnDraft` to be generated.

A draft version of the email sent to lvccirc@caeplla2.gsfc.nasa.gov for distribution to the LVC GCN Circular list. This message is meant to be sent to a LLAMA member for verification and modifications before being manually

submitted to GCN.

```
DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class  
'llama.files.i3.json.IceCubeNeutrinoList'>, <class  
'llama.files.i3.txt.IceCubeNeutrinoListCincTxt'>, <class  
'llama.files.coinc_o2.RctGdbCoincSkymap02Large'>, <class  
'llama.files.i3.json.RctGdbIceCubeNeutrinoList'>, <class  
'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson'>, <class  
'llama.files.lvc_skymap.LvcSkymapFits'>)  
  
FILENAME = 'gwhen_coinc_icecube_gcn_circular_draft.txt'  
  
MANIFEST_TYPES = (<class 'llama.files.gcn_draft_o2.I3LvcGcnDraft'>,)  
  
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class  
'llama.files.lvc_skymap.LvcSkymapFits'>, <class  
'llama.files.lvc_skymap_mat.LvcSkymapMat'>, <class  
'llama.files.i3.json.IceCubeNeutrinoList'>, <class 'llama.files.gracedb.PAstro'>,  
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class  
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class  
'llama.files.gracedb.LVCGraceDbEventData'>, <class  
'llama.files.coinc_o2.IceCubeNeutrinoListMat'>, <class  
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>, <class  
'llama.files.coinc_o2.CoincSkymap02Large'>, <class  
'llama.files.i3.json.RctGdbIceCubeNeutrinoList'>, <class  
'llama.files.i3.txt.IceCubeNeutrinoListCincTxt'>, <class  
'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson'>, <class  
'llama.files.coinc_o2.RctGdbCoincSkymap02Large'>)
```

```

UR_DEPENDENCY_TREE = ImmutableDict({<class
'llama.files.coinc_o2.RctGdbCoincSkymapO2Large': ImmutableDict({<class
'llama.files.coinc_o2.CoincSkymapO2Large': ImmutableDict({<class
'llama.files.coinc_o2.IceCubeNeutrinoListMat': ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{})), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}), <class
'llama.files.lvc_skymap_mat.LvcSkymapMat': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}), <class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}))}), <class
'llama.files.lvc_skymap_mat.LvcSkymapMat': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}), <class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}))}), <class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{})), <class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{})), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}), <class
'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson': ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{})), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}), <class
'llama.files.lvc_skymap_mat.LvcSkymapMat': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}), <class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}))}), <class
'llama.files.i3.txt.IceCubeNeutrinoListCoincTxt': ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{})), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}), <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc': ImmutableDict({<class
'llama.files.gracedb.LVCGraceDbEventData': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{})), <class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{})), <class
'llama.files.lvc_skymap.LvcSkymapHdf5': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}))}), <class
'llama.files.lvc_skymap.LvcDistancesJson': ImmutableDict({<class
'llama.files.lvc_skymap.LvcSkymapFits': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}))}), <class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}), <class
'llama.files.gracedb.PAstro': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}))}), <class
'llama.files.i3.json.RctGdbIceCubeNeutrinoList': ImmutableDict({<class
'llama.files.i3.json.IceCubeNeutrinoList': ImmutableDict({<class
'llama.files.skymap_info.SkymapInfo': ImmutableDict({{}))})

```

property authors

Return a list of authors for this Circular.

property conclusion

A conclusion paragraph for this Circular.

property description

A description of IceCube.

property introduction

An introductory paragraph for this Circular.

property neutrino_table

Return the human-readable neutrino table that will go in this Circular.

property postscript

A postscript with some extra info about the LLAMA project.

```
property reconstructed_neutrino_table
```

A table showing the value of the reconstructed neutrino.

property reconstructed neutrinos

Return the list of reconstructed neutrino directions generated by the MATLAB analysis pipeline.

property subject

Get a highly-descriptive subject line for the Circular email.

property table legend

A legend explaining the contents of the neutrino table.

property temporally coincident neutrinos

Return the list of temporally-coincident neutrinos fetched from IceCube.

property uploaded file links

Don't mention GraceDB uploaded files if we haven't uploaded there for whatever reason

```
class llama.files.gcn.draft.o2.RctTeam13LvcGcnDraft(eventid or fh, rundir=None)
```

Bases: *llama_files.team.receipts*, *TeamEmailReceipt*

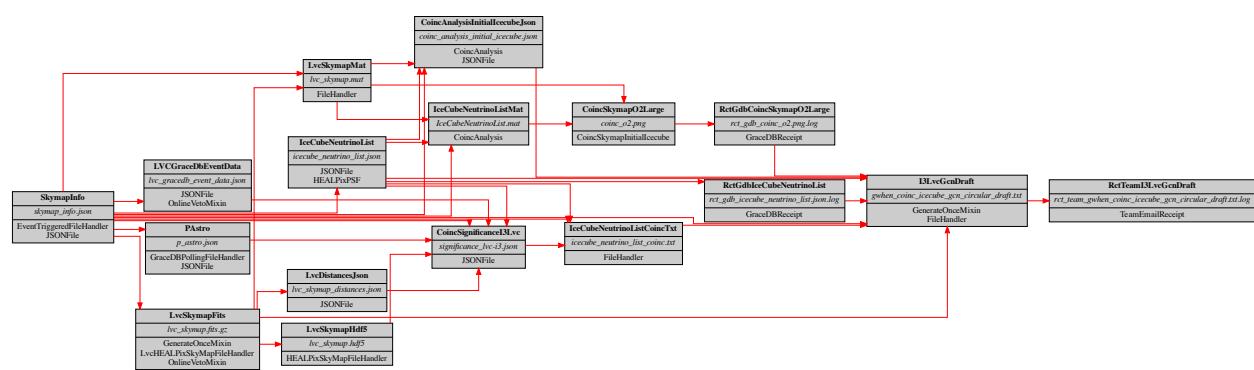


Fig. 100: Required input files for `llama.files.gcn_draft_o2.RctTeamI3LvcGcnDraft` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.gcn_draft_o2.RctTeamI3LvcGcnDraft` to be generated.

```
DEPENDENCIES = (<class 'llama.files.gcn_draft_o2.I3LvcGcnDraft'>,)
```

FILENAME = 'rct_team_qwhen_coinc_icecube_gcn_circular_draft.txt.log'

MANIFEST_TYPES = (<class 'llama.files.gcn_draft_o2.RctTeamI3LvcGcnDraft'>,)

UPLOAD

alias of `llama.files.qcn` draft 02. `I3LvcGcnDraft`

```

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.lvc_skymap_mat.LvcSkymapMat'>, <class
'llama.files.i3.json.IceCubeNeutrinoList'>, <class 'llama.files.gracedb.PAstro'>,
<class 'llama.files.lvc_skymap.LvcDistancesJson'>, <class
'llama.files.lvc_skymap.LvcSkymapHdf5'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.coinc_o2.IceCubeNeutrinoListMat'>, <class
'llama.files.coinc_significance.opa.CoincSignificanceI3Lvc'>, <class
'llama.files.coinc_o2.CoincSkymapO2Large'>, <class
'llama.files.i3.json.RctGdbIceCubeNeutrinoList'>, <class
'llama.files.i3.txt.IceCubeNeutrinoListCoincTxt'>, <class
'llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson'>, <class
'llama.files.coinc_o2.RctGdbCoincSkymapO2Large'>, <class
'llama.files.gcn_draft_o2.I3LvcGcnDraft'>)

UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.gcn_draft_o2.I3LvcGcnDraft'>})

property subject
    Email subject for <class 'llama.files.gcn_draft_o2.I3LvcGcnDraft'> email upload.

```

28.13 llama.files.gracedb module

FileHandlers for uploading and downloading files from GraceDB plus utilities for interacting with GraceDB.

```

class llama.files.gracedb.GraceDBPollingFileHandler(eventid_or_fh, rundir=None)
    Bases: llama.filehandler.FileHandler, llama.filehandler.mixins.OnlineVetoMixin

A data file that can be downloaded from GraceDB as soon as it is ready. SkymapInfo is necessarily a dependency for all subclasses (since we need to extract the GraceID from it). In most cases, this should be the only dependency.

DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)

abstract property remote_filename
    The name of the required data in a format recognizable by the remote resource's API. This is quite possibly dynamically generated, and so it must be considered separate from the file handler's name for this file.

class llama.files.gracedb.GraceDBReceipt(eventid_or_fh, rundir=None)
    Bases: llama.com.utils.UploadReceipt

A log file created when a file is uploaded to GraceDB.

CLASSNAME_FMT = 'RctGdb{}'
FILENAME_FMT = 'rct_gdb_{}.log'

classmethod decorator_dict(upload, log_message='Upload from LLAMA pipeline.')
    See UploadReceipt.upload_this and UploadReceipt.decorator_dict.

```

Parameters

- **upload** – The decorated FileHandler class that is being registered for upload.
- **log_message** (*function or str, optional*) – Either a string whose format function will be called with the new GraceDBReceipt subclass as its `self` keyword argument or a function taking the new GraceDBReceipt as its only argument (will become the `log_message` property for the new GraceDBReceipt).

Returns newclassdict – A dictionary that can be passed to `type` to specify the attributes of a new class.

Return type dict

Raises TypeError – If `log_message` is not a formattable string or callable object.

property gracedb_tags

Specify tags (which affect display and access permissions) for the uploaded GraceDB file. Can be overridden for files that don't fit the default topics:

[u'lvem', u'em_follow', u'sky_loc']

abstract property log_message

A message to accompany this file on the GraceDB log entry for this event. The last line of this log file is a JSON object describing the result of the upload operation.

classmethod set_class_attributes(subclass)

See `UploadReceipt.set_class_attributes`; this method additionally adds `SkymapInfo` to the `DEPENDENCIES` list for `subclass` (if it is not implicitly there by virtue of being the uploaded file).

property upload_dict

When uploading to GraceDB via the REST API, the GraceDB client produces a dict with useful information about the uploaded file. This method returns that dict. An example instance of this dict:

```
{u'N': 46, u'comment': u'Testing json upload ONE MORE TIME!', u'created': u'2017-03-19T04:49:43.515566+00:00', u'file': (u'https://gracedb.ligo.org/api/events/M278200/'  
u'files/icecube_neutrino_list.json',4'),  
u'file_version': 4, u'filename': u'icecube_neutrino_list.json', u'issuer': {u'display_name':  
u'Stefan Countryman',  
u'username': u'stefan.countryman@LIGO.ORG'109,},  
u'self': u'https://gracedb.ligo.org/api/events/M278200/log/46', u'tag_names':  
[u'em_follow'], u'tags': u'https://gracedb.ligo.org/api/events/M278200/log/46/tag/'}
```

If the log file doesn't exist or the upload dict cannot be read, this function returns None.

```
class llama.files.gracedb.LVCGraceDbEventData(eventid_or_fh, rundir=None)  
Bases: llama.filehandler.JSONFile, llama.filehandler.mixins.OnlineVetoMixin
```

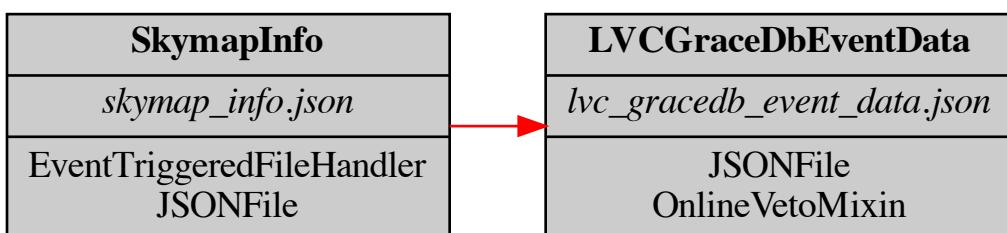


Fig. 101: Required input files for `llama.files.gracedb.LVCGraceDbEventData` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.gracedb.LVCGraceDbEventData` to be generated.

The event dictionary as returned by the GraceDb API. Includes richer information on the single-detector and multi-detector inspiral parameters than is provided in LVAerts or GCN Notices. Convenience methods are provided for certain commonly-used reconstructed parameters, but full data can always be accessed by running `read_json`.

¹⁰⁹ u'stefan.countryman@LIGO.ORG

```

DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
FILENAME = 'lvc_gracedb_event_data.json'
GW_DETECTORS = {'H1', 'L1', 'V1'}
MANIFEST_TYPES = (<class 'llama.files.gracedb.LVCGraceDbEventData'>,)
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})

property chirp_mass
    The chirp mass. Raises a KeyError if it is not defined.

property instruments
    A sorted list of the interferometers that participated in creating this trigger, e.g. ['H1', 'L1']. Raises a ValueError if unexpected instruments are found. Raises a KeyError if it is not defined.

property offline
    Whether this trigger was generated offline.

property snr
    The coincident signal-to-noise ratio (SNR). Raises a KeyError if it is not defined.

property total_mass
    The total binary mass. Raises a KeyError if it is not defined.

class llama.files.gracedb.PAstro(eventid_or_fh, rundir=None)
    Bases: llama.files.gracedb GraceDBPollingFileHandler, llama.filehandler.JSONFile

```

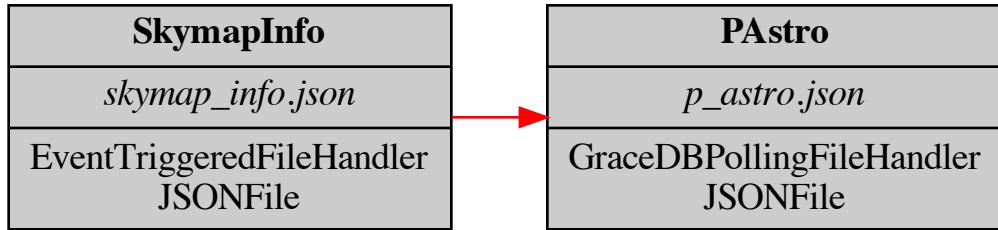


Fig. 102: Required input files for `llama.files.gracedb.PAstro` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.gracedb.PAstro` to be generated.

A classification of an event's possible sources by probability.

```

FILENAME = 'p_astro.json'

MANIFEST_TYPES = (<class 'llama.files.gracedb.PAstro'>,)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)

UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})

property most_likely_population
    Get the population that this event most likely belongs too by seeing whether it is more likely than not to contain NS matter. More precisely, compares p_bbh to p_bns + p_nsbh + p_mass_gap, returning 'bbh' if the former is larger and 'bns' if the latter is larger. p_terr is ignored in this calculation.

property p_bbh
    The probability that this was a BBH (binary black hole) merger.

```

property p_bns

The probability that this was a BNS (binary neutron star) merger.

property p_mass_gap

The probability that at least one of the compact objects in the merger was in the ambiguous mass gap between definite BH and definite NS.

property p_nsbh

The probability that this was a NSBH (neutron star/black hole) merger.

property p_terr

The probability that this event was terrestrial noise.

property remote_filename

The name of the required data in a format recognizable by the remote resource's API. This is quite possibly dynamically generated, and so it must be considered separate from the file handler's name for this file.

```
class llama.files.gracedb.RctSlkLmaLVCGraceDbEventData(eventid_or_fh, rundir=None)
```

Bases: [llama.files.slack.SlackReceiptLlama](#)

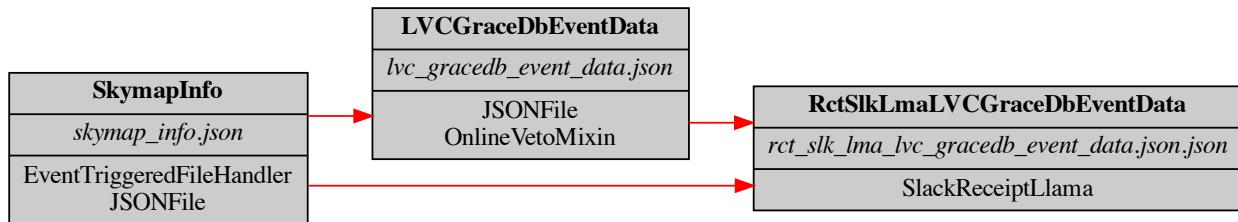


Fig. 103: Required input files for `llama.files.gracedb.RctSlkLmaLVCGraceDbEventData` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.gracedb.RctSlkLmaLVCGraceDbEventData` to be generated.

```

DEPENDENCIES = (<class 'llama.files.gracedb.LVCGraceDbEventData'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_slk_lma_lvc_gracedb_event_data.json.json'
FILENAME_FMT = 'rct_slk_lma_{}.json'

MANIFEST_TYPES = (<class 'llama.files.gracedb.RctSlkLmaLVCGraceDbEventData'>,)

UPLOAD
    alias of llama.files.gracedb.LVCGraceDbEventData

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.gracedb.LVCGraceDbEventData'>)

UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.gracedb.LVCGraceDbEventData'>})

class_veto = ()
  
```

28.14 llama.files.gwastro module

FileHandlers that upload files to gw-astronomy.org/gwhen and record successful uploads with logfile receipts.

```
class llama.files.gwastro.GWAstroReceipt(eventid_or_fh, rundir=None)
Bases: llama.com.utils.UploadReceipt
```

A log file created indicating successful upload of some other original file to gw-astronomy.org.

```
CLASSNAME_FMT = 'RctGwa{}'
FILENAME_FMT = 'rct_gwa_{}.log'
REMOTE_RUNDIR = '/home/gwhen/content/events/'
REMOTE_URL = 'gw-astronomy.org'
REMOTE_USER = 'gwhen'
```

property rsync_source

Get the source path for an rsync upload to a remote server (including the proper --relative dot in the path to ensure subdirectory creation serverside).

classmethod set_class_attributes(subclass)

See UploadReceipt.set_class_attributes; this method adds SkymapInfo to subclass.DEPENDENCIES if it is not already a member and RSYNC_DEST, the destination directory for uploaded events.

```
class llama.files.gwastro.RctGwaLVAAlertJSON(eventid_or_fh, rundir=None)
```

Bases: llama.files.gwastro.GWAstroReceipt

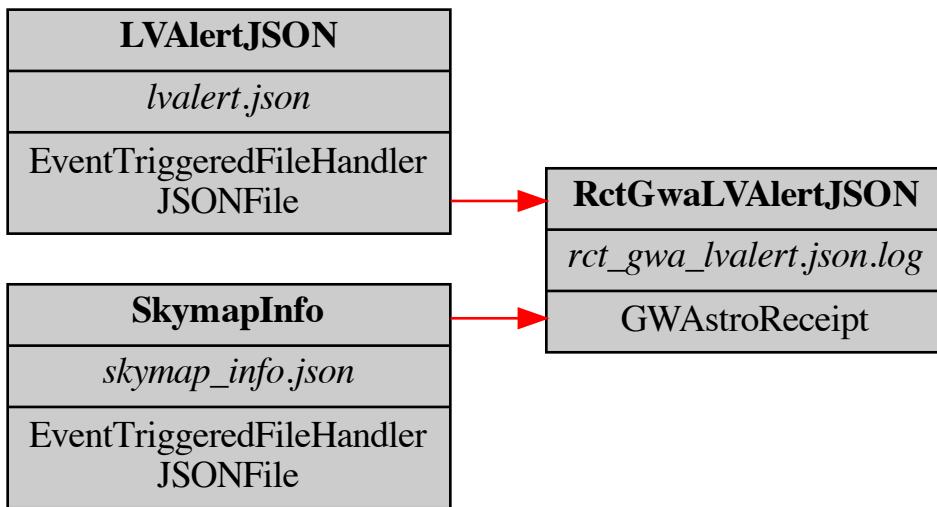


Fig. 104: Required input files for `llama.files.gwastro.RctGwaLVAAlertJSON` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.gwastro.RctGwaLVAAlertJSON` to be generated.

```
DEPENDENCIES = (<class 'llama.files.lvalert_json.LVAAlertJSON'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_gwa_lvalert.json.log'

MANIFEST_TYPES = (<class 'llama.files.gwastro.RctGwaLVAAlertJSON'>,)
```

```
RSYNC_DEST = 'gwhen@gw-astronomy.org:/home/gwhen/content/events/'

UPLOAD
    alias of llama.files.lvalert\_json.LVAlertJSON

UR_DEPENDENCIES = (<class 'llama.files.lvalert_json.LVAlertJSON'>,)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvalert_json.LVAlertJSON'>})

class llama.files.gwastro.RctGwaLvcGcnXml(eventid_or_fh, rundir=None)
Bases: llama.files.gwastro.GWAstroReceipt
```

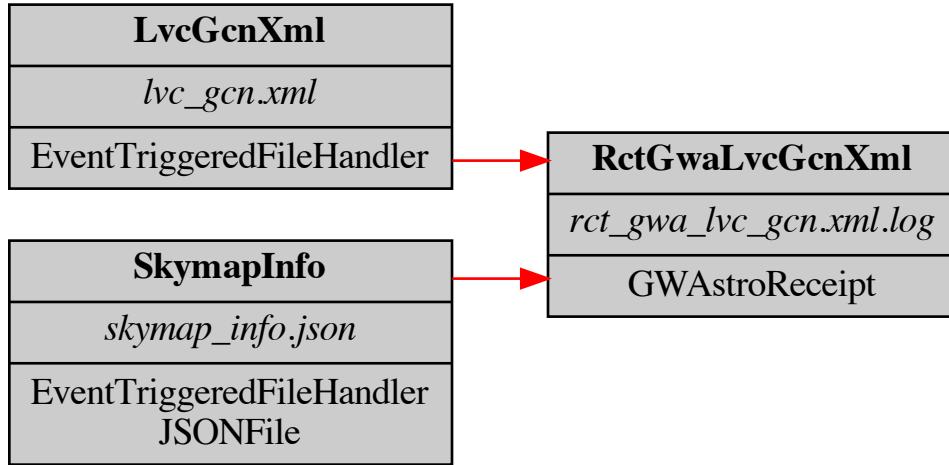


Fig. 105: Required input files for `llama.files.gwastro.RctGwaLvcGcnXml` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.gwastro.RctGwaLvcGcnXml` to be generated.

```
DEPENDENCIES = (<class 'llama.files.lvc_gcn_xml.LvcGcnXml'>, <class
    'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_gwa_lvc_gcn.xml.log'

MANIFEST_TYPES = (<class 'llama.files.gwastro.RctGwaLvcGcnXml'>,)

RSYNC_DEST = 'gwhen@gw-astronomy.org:/home/gwhen/content/events/'

UPLOAD
    alias of llama.files.lvc\_gcn\_xml.LvcGcnXml

UR_DEPENDENCIES = (<class 'llama.files.lvc_gcn_xml.LvcGcnXml'>,)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvc_gcn_xml.LvcGcnXml'>})

class llama.files.gwastro.RctGwaLvcRetractionXml(eventid_or_fh, rundir=None)
Bases: llama.files.gwastro.GWAstroReceipt

DEPENDENCIES = (<class 'llama.files.lvc_gcn_xml.LvcRetractionXml'>, <class
    'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'rct_gwa_lvc_gcn_retraction.xml.log'

MANIFEST_TYPES = (<class 'llama.files.gwastro.RctGwaLvcRetractionXml'>,)

RSYNC_DEST = 'gwhen@gw-astronomy.org:/home/gwhen/content/events/'

UPLOAD
    alias of llama.files.lvc\_gcn\_xml.LvcRetractionXml
```

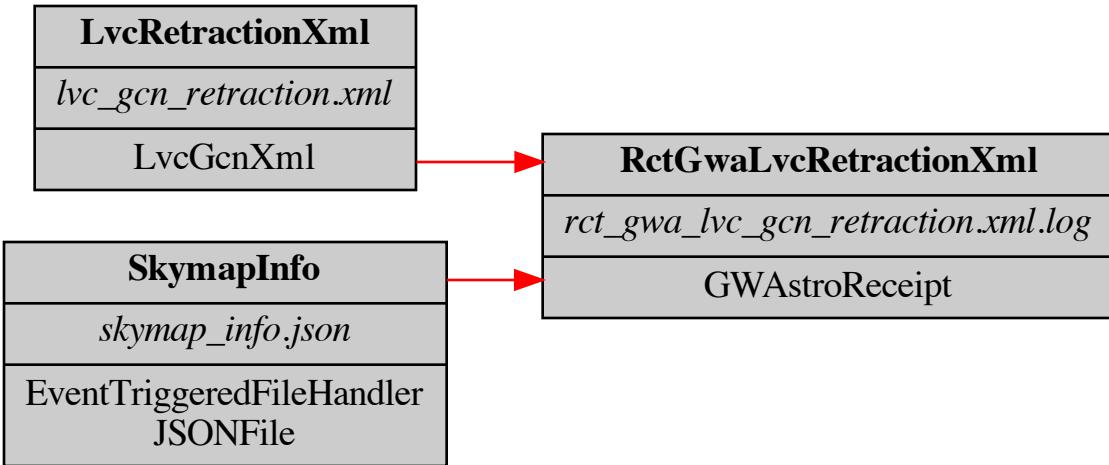


Fig. 106: Required input files for `llama.files.gwastro.RctGwaLvcRetractionXml` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.gwastro.RctGwaLvcRetractionXml` to be generated.

```

UR_DEPENDENCIES = (<class 'llama.files.lvc_gcn_xml.LvcRetractionXml'>,)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvc_gcn_xml.LvcRetractionXml'>})

class llama.files.gwastro.RctGwaSkymapInfo(eventid_or_fh, rundir=None)
Bases: llama.files.gwastro.GWAstroReceipt

```

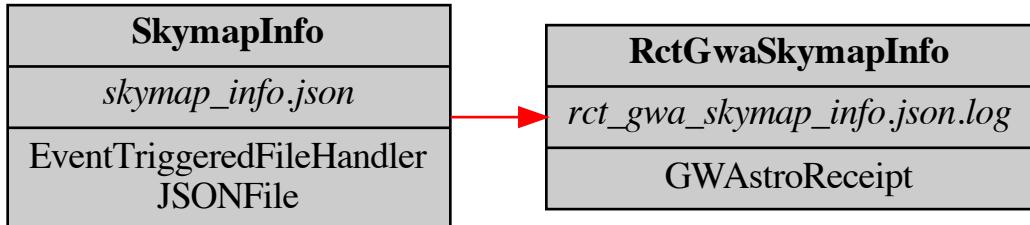


Fig. 107: Required input files for `llama.files.gwastro.RctGwaSkymapInfo` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.gwastro.RctGwaSkymapInfo` to be generated.

```

DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
FILENAME = 'rct_gwa_skymap_info.json.log'
MANIFEST_TYPES = (<class 'llama.files.gwastro.RctGwaSkymapInfo'>,)
RSYNC_DEST = 'gwhen@gw-astronomy.org:/home/gwhen/content/events/'
UPLOAD
alias of llama.files.skymap_info.SkymapInfo
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})

```

28.15 `llama.files.lvalert_advok` module

A FileHandler that indicates whether an event was tagged ADVOK via LVAAlert.

```
class llama.files.lvalert_advok.LVAlertAdvok(eventid_or_fh, rundir=None)
    Bases: llama.filehandler.EventTriggeredFileHandler, llama.filehandler.JSONFile
```



Fig. 108: `llama.files.lvalert_advok.LVAlertAdvok` is created from external triggers. It therefore has no LLAMA-representable input dependencies, but instead acts as initial input for other `FileHandler` classes.

If the event was tagged ADVOK via an LVAAlert, this file will contain the LVAAlert JSON data.

```
FILENAME = 'lvalert_advok.json'
MANIFEST_TYPES = (<class 'llama.files.lvalert_advok.LVAlertAdvok'>,)
UR_DEPENDENCIES = ()
UR_DEPENDENCY_TREE = frozenset({})
```

28.16 `llama.files.lvalert_json` module

FileHandler for a JSON file holding data about an LVAAlert. Generated when an LVAAlert comes in.

```
class llama.files.lvalert_json.LVAlertJSON(eventid_or_fh, rundir=None)
    Bases: llama.filehandler.EventTriggeredFileHandler, llama.filehandler.JSONFile
```



Fig. 109: `llama.files.lvalert_json.LVAlertJSON` is created from external triggers. It therefore has no LLAMA-representable input dependencies, but instead acts as initial input for other `FileHandler` classes.

An LVAAlert JSON object containing skymap info.

```
FILENAME = 'lvalert.json'
MANIFEST_TYPES = (<class 'llama.files.lvalert_json.LVAlertJSON'>,)
UR_DEPENDENCIES = ()
UR_DEPENDENCY_TREE = frozenset({})
```

property alert_type

Get the alert_type of this LVAAlert.

property base_filename

Get the unversioned filename for this file. This only works for LVAAlerts corresponding to uploads.

Returns `base_filename` – The skymap filename as a `SkymapFilename` instance.

Return type `files.lvc_skymap.utils.SkymapFilename`

Raises `KeyError` – If this LVAAlert does not contain a filename.

property far

The False Alarm Rate of this trigger.

property graceid

The GraceID, a unique ID used on GraceDB to register gravitational wave triggers.

property notice_time_str

Get the time at which the LVAAlert was created.

property skymap_filename

Get the exact filename, including version information, for this file. If this LVAAlert does not correspond to a skymap upload, try to pull down the most recently generated skymap.

Returns `filename` – The full canonical filename as a `SkymapFilename` instance.

Return type `files.lvc_skymap.utils.SkymapFilename`

property skymap_version

Get the version of the skymap with this particular filename. This only works for LVAAlerts corresponding to uploads.

Returns `version` – File version corresponding to this upload.

Return type `int`

28.17 `llama.files.lvc_gcn_xml` module

Class for saving, parsing, and conveniently retrieving data from VOEvent XML files distributed by the GCN network and created by LVC describing GW triggers. Includes functionality work with multiple schemas of LVC VOEvents from the Advanced LIGO era.

```
class llama.files.lvc_gcn_xml.LvcGcnXml(eventid_or_fh, rundir=None)
Bases: llama.filehandler.EventTriggeredFileHandler
```

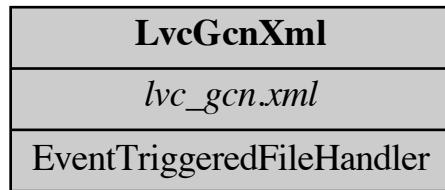


Fig. 110: `llama.files.lvc_gcn_xml.LvcGcnXml` is created from external triggers. It therefore has no LLAMA-representable input dependencies, but instead acts as initial input for other `FileHandler` classes.

A VOEvent XML file received from GCN corresponding to an LVC event notice.

```
FILENAME = 'lvc_gcn.xml'
```

```
MANIFEST_TYPES = (<class 'llama.files.lvc_gcn_xml.LvcGcnXml'>,)
UR_DEPENDENCIES = ()
UR_DEPENDENCY_TREE = frozenset({})

property alert_type
    Get the alert type for this VOEvent.

property etree
    Return an lxml.etree for this VOEvent.

property event_time_gps
    Get the time of the observed event up to nanosecond precision (less accurate than this in practice) in GPS time format.

property event_time_gps_nanoseconds
    Get the number of nanoseconds past the last GPS second at the time of the observed event. Ostensibly provides nanosecond precision, but is less accurate than this in practice.

property event_time_gps_seconds
    Get the time of the observed event in GPS seconds (truncated to the nearest second).

property event_time_mjd
    Get the time of the observed event in UTC MJD format.

property event_time_str
    Get a unicode string with the ISO date and time of the observed event straight from the VOEvent file. UTC time.

property far
    Get the false alarm rate of this VOEvent as a float value.

get_alert_type()
    Read the alert type of this GCN Notice.

get_param(param)
    Get this VOEventParam for this event.

property graceid
    Get the GraceID corresponding to this VOEvent.

property ivorn
    Get the IVORN, a unique identifier for this GCN Notice.

property notice_time_str
    Get a unicode string with the date and time of creation of the notification associated with this VOEvent, rather than the time of detection of the event itself. Should be in ISO format.

property pipeline
    Return an ALL-CAPS name of the pipeline used to generate this event, e.g. GSTLAL or CWB.

property role
    Get the role of this VOEvent as specified in the header.

property skymap_filename
    Get the filename for this skymap as it appears on GraceDB.

class llama.files.lvc_gcn_xml.LvcRetractionXml(eventid_or_fh, rundir=None)
    Bases: llama.files.lvc\_gcn\_xml.LvcGcnXml

    A VOEvent XML file retracting a GCN notice.

    FILENAME = 'lvc_gcn_retraction.xml'
```

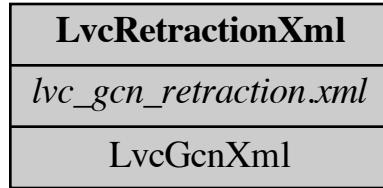


Fig. 111: `llama.files.lvc_gcn_xml.LvcRetractionXml` is created from external triggers. It therefore has no LLAMA-representable input dependencies, but instead acts as initial input for other `FileHandler` classes.

`llama.files.lvc_gcn_xml.get_filename_from_group(filename, paramname)`

old VOEvent format had the skymap URL buried in a Param which itself is contained in a group:

VOEvent/What/Group/Param

This is mostly the same as get VOEventParam in utils and is rewritten here only to add support for an edge case.

`llama.files.lvc_gcn_xml.get_filename_from_group_name(filename, grouptypename, paramname)`

some stupid VOEvent format has things nested under filter(lambda a: a['type'] == 'GW_SKYMAP', event.voe_VOEvent.What.Group)[0] so go ahead and look under different group type names to find the right skymap URL.

`llama.files.lvc_gcn_xml.get_filename_from_param(filename, paramname)`

get the skymap URL for newer style VOEvents, which have the skymap param directly under the What element:

VOEvent/What/Param

`llama.files.lvc_gcn_xml.get_skymap_filename(filename)`

Get the skymap URL regardless of the current VOEvent format from the VOEvent file with the given filename.

`llama.files.lvc_gcn_xml.parse_ivorn(payload)`

Get identifying information about an LVC VOEvent, raising a `ValueError` if the IVORN cannot be parsed. Provides a safe way of handling errors for the GCN listener, which should to the greatest extent possible handle errors due to malformed data from partners.

Parameters `payload (str)` – The LVC VOEvent in XML string format.

Returns

- **ivorn (str)** – The event's unique ID, called an “IVORN”.
- **eventid (str)** – A combination of GraceID, serial GCN notice number for the event, and notice type for this event that will be used for OPA events in the pipeline.
- **graceid (str)** – The GraceID of the gravitational wave event.
- **serial_no (int)** – The serial number of this GCN Notice, e.g. 2 for the second notice related to this particular GraceID.
- **notice_type (str)** – The LVC GCN Notice type for this notice; expected to be one of Preliminary, Initial, Update, or Retraction, though this is not checked.

Raises `ValueError` – A descriptive error raised if root is not a `lxml.etree._Element` describing a valid LVC GCN Notice. The message will describe what part of parsing failed.

Examples

Get the IVORN, eventid, and GraceID from a test event file:

```
>>> from llama.utils import get_test_file
>>> with open(get_test_file('lvc_gcn.xml',
...                         'MS181101ab-2-Initial')) as xmlfile:
...     payload = xmlfile.read()
>>> parse_ivorn(payload)
('ivo://gwnet/LVC#MS181101ab-2-Initial', 'MS181101ab-2-Initial', 'MS181101ab', 2,
 'Initial')
```

28.18 `llama.files.lvc_skymap_mat` module

Make a MAT file that can be read in by LoadCWbData.m, in the style of the FormatToCWbData.m function written by Rainer. Adapted from code and information provided by Rainer Corley and Imre Bartos.

```
class llama.files.lvc_skymap_mat.LvcSkymapMat(eventid_or_fh, rundir=None)
    Bases: llama.filehandler.FileHandler
```

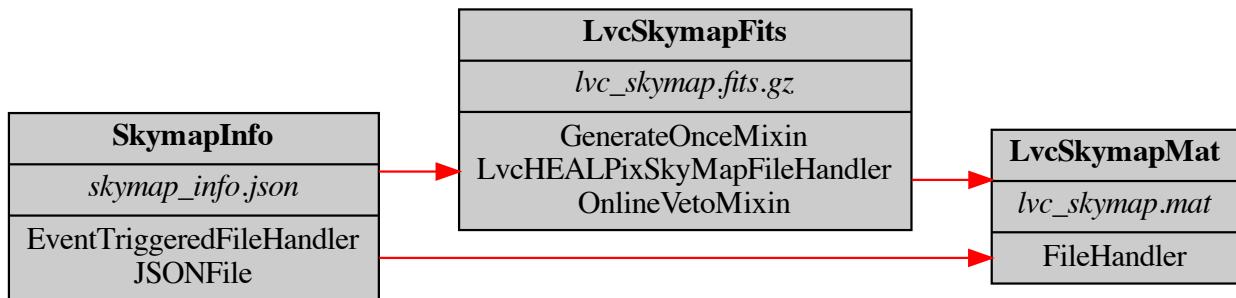


Fig. 112: Required input files for `llama.files.lvc_skymap_mat.LvcSkymapMat` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.lvc_skymap_mat.LvcSkymapMat` to be generated.

A list of sky positions and probability densities in text form generated from the original Healpix-encoded LVC skymap.

```
DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapFits'>, <class
'llama.files.skymap_info.SkymapInfo'>)

FILENAME = 'lvc_skymap.mat'

MANIFEST_TYPES = (<class 'llama.files.lvc_skymap_mat.LvcSkymapMat'>,)

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.lvc_skymap.LvcSkymapFits'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class 'llama.files.skymap_info.SkymapInfo'>:
ImmutableDict({}), <class 'llama.files.lvc_skymap.LvcSkymapFits'>:
ImmutableDict({<class 'llama.files.skymap_info.SkymapInfo'>: ImmutableDict({})})})

llama.files.lvc_skymap_mat.argsort_end_of_cr(pixel_values, total_probability=0.9)
```

Take input values, sort them in descending order, and perform a binary search to find how many values must be included such that the sum of those top values gives the desired total_probability. Return a list of indices which,

when used to get a slice from the original list of pixel values, will give the cut off list. This can be used to sort multiple array fields (while cutting unneeded values) as if they were rows in a spreadsheet.

28.19 `llama.files.lvc_skymap_txt` module

A FileHandler that generates a list of pixel positions and values from a HEALPix skymap to be used in the old O2 MATLAB pipeline.

```
class llama.files.lvc_skymap_txt.LvcSkymapTxt(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler
```

A list of sky positions and probability densities in text form generated from the original Healpix-encoded skymap. The values form a space-delimited table, with each row corresponding to a pixel and the columns corresponding to theta, phi, and probability density (**Header is not included in text file**):

theta [deg]	phi [deg]	probability density
...

```
DEPENDENCIES = (<class 'llama.files.lvc_skymap.LvcSkymapFits'>,)
FILENAME = 'lvc_skymap.txt'
```

28.20 `llama.files.matlab` module

FileHandlers that make calls to MATLAB in order to generate their data. You will need to specify the directory containing MATLAB functions to call with the environmental variable MATLABDIR.

```
class llama.files.matlab.MATLABCallingFileHandler(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler
```

A class that is generated by calling a MATLAB function. The generator just calls the matlab function specified by the `matlab_command` property using MATLAB command-line options specified by `matlab_options`.

abstract property `matlab_command`

Return the command that MATLAB will eval in order to generate this file.

`matlab_options = []`

```
llama.files.matlab.matlab_eval(matcmd, cli_flags=None)
```

Run a command in a headless instance of MATLAB. By default, changes to the Neutrinos directory of the repo directory, which is where all LLAMA MATLAB code is currently stored. NOTE that your command can have multiple statements, but it CANNOT have newlines; it must have semicolons separating valid statements (See example below).

You can also provide a list of command line flags like “-nojvm” to modify MATLAB’s behavior. *This implementation will be removed if we switch to MATLAB Engine, since MATLAB is not called through the command line interface when using the MATLAB Engine interface.*

Examples

Print “foo” and “bar” on separate lines, checking first if MATLAB is installed: >>> try: ... matpath = matlabpath() ... matlab_eval(“disp(‘foo’);disp(‘bar’)”) ... except OSError: ... # put your cleanup code here ... pass

```
llama.files.matlab.matlabpath()
```

Get a path to a `matlab` executable (if it exists). If something called `matlab` is in your `$PATH`, this will be used. Otherwise, if you have specified `$MATLAB_EXECUTABLE_PATH` as an environmental variable, that path will be used. If both of these checks fail, the most recent version of MATLAB matching the pattern `/Applications/MATLAB_R20*.app/bin/matlab` will be used (only applicable on MacOS).

28.21 `llama.files.sms_receipts` module

FileHandlers that send SMS messages for high-priority human-in-the-loop alerts.

You will need to configure certain environmental variables to let LLAMA know how to send messages to the outside world.

If you have not specified your twilio credentials, you will see the following message in your warnings (which are usually suppressed for optional features like this), and if you try to send out SMS uploads, you’ll see the following as the error message explaining why file generation failed:

```
Twilio API credentials are not configured;  
you will not be able to submit text message notifications until this  
is fixed.
```

```
Please fill in your Twilio API credentials in e.g. your .bashrc, which should  
look something like the following (where ``PLACEHOLDER`` has obviously been  
replaced with your actual authentication information):
```

```
export TWILIO_ACCOUNT_SID=PLACEHOLDER  
export TWILIO_AUTH_TOKEN=PLACEHOLDER  
export TWILIO_NUMBER=2407432233
```

If you have not specified phone numbers with which to contact team members, you will see the following message in your warnings and errors:

```
You must specify phone numbers in a JSON format in  
the environmental variable LLAMA_PHONE_NUMBERS in order to send SMS  
receipts. Your phonebook should be defined in your .bashrc or somewhere similar  
like:
```

```
export LLAMA_PHONE_NUMBERS='[  
    {  
        "phone_number": "+395555555555",  
        "name": "John Smith"  
    },  
    {  
        "phone_number": "+12345678900",  
        "name": "Jane Doe"  
    }  
'
```

You will need to specify environmental variables with the missing configuration data as described above in order to use SMS receipts.

```
class llama.files.sms_receipts.MessageClient
Bases: object
```

A client for actually sending messages using Twilio's api, modified from their example app.

```
send_message(body, recipient)
```

Send a text message.

Parameters

- **body** (*str*) – The body of the text message.
- **recipient** (*str*) – The cell phone number of the recipient.

```
class llama.files.sms_receipts.RctSMSAdvokJSON(eventid_or_fh, rundir=None)
```

Bases: *llama.files.sms_receipts.SMSReceipt*

A log file created when alerting LLAMA team members of a new event that requires immediate attention.

```
DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>, <class
'llama.files.advok.Advok'>)
```

```
FILENAME = 'rct_sms_skymap_info.json.log'
```

```
MANIFEST_TYPES = (<class 'llama.files.sms_receipts.RctSMSAdvokJSON'>,)
```

UPLOAD

alias of *llama.files.skymap_info.SkymapInfo*

```
UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
```

```
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})
```

property message_for_team

Must specify a formula for generating the message that the team receives.

```
class llama.files.sms_receipts.SMSReceipt(eventid_or_fh, rundir=None)
```

Bases: *llama.com.utils.UploadReceipt*

A log file created when a text message is sent out to team members.

```
FILENAME_FMT = 'rct_sms_{}.log'
```

abstract property message_for_team

Must specify a formula for generating the message that the team receives.

```
classmethod send_message_to_team(body)
```

Send a text message to all team members.

```
classmethod set_class_attributes(subclass)
```

See *UploadReceipt.set_class_attributes*; this method additionally adds Advok to the DEPENDENCIES list for subclass (if it is not already there).

28.22 llama.files.team_receipts module

FileHandlers that send out information to the LLAMA team via email and which create log files to record the successful sending of those warning emails.

```
class llama.files.team_receipts.TeamEmailReceipt(eventid_or_fh, rundir=None)
```

Bases: [llama.com.email.EmailReceipt](#)

Emails with important files should be sent to LLAMA team members for review. These receipts record successful sending of those emails.

```
CLASSNAME_FMT = 'RctTeam{}'
```

```
FILENAME_FMT = 'rct_team_{}.log'
```

```
classmethod decorator_dict(upload, subject=None)
```

See `UploadReceipt.upload_this` and `UploadReceipt.decorator_dict`.

Parameters

- **upload** – The decorated FileHandler class that is being registered for upload.
- **subject** (*function or str, optional*) – Either a string whose `format` function will be called with the new `TeamEmailReceipt` as its `self` keyword argument or a function taking the new `TeamEmailReceipt` as its only argument (will become the `subject` property for the new `TeamEmailReceipt`). Uses the default `TeamEmailReceipt.subject` by default.

Returns `newclassdict` – A dictionary that can be passed to `type` to specify the attributes of a new class.

Return type

Raises `TypeError` – If `subject` is not a formattable string or callable object.

property recipients

The quick response team will be the same for most alert emails.

property subject

By default, just alert the team with an email saying which file is ready (defined as `self.UPLOAD`) and the `eventid`.

28.23 llama.files.timing_checks module

FileHandlers that initialize timing checks for an event as part of LLAMA analysis. This is a kludge to do work that's not actually related to LLAMA.

```
class llama.files.timing_checks.TimingChecks(eventid_or_fh, rundir=None)
```

Bases: [llama.filehandler.FileHandler](#), [llama.filehandler.mixins.OnlineVetoMixin](#), [llama.filehandler.mixins.ObservingVetoMixin](#)

Initialize GECO timing checks on ldas-pcdev2.ligo.caltech.edu to help confirm validity of GW observation.

```
DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
```

```
FILENAME = 'timing_checks.log'
```

28.24 `llama.files.uw_summary` module

Download the UW Madison GW/High Energy Neutrino analysis summary results for a given event.

```
class llama.files.uw_summary.UwSummary(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler, llama.filehandler.mixins.OnlineVetoMixin
```

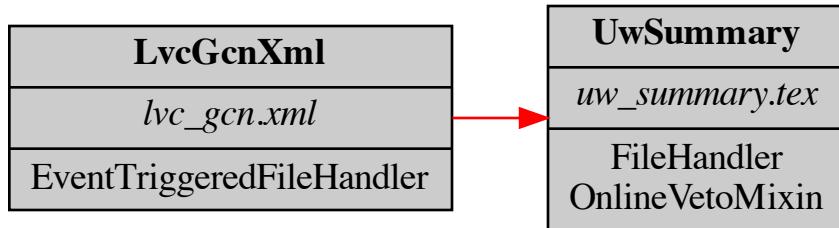


Fig. 113: Required input files for `llama.files.uw_summary.UwSummary` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.uw_summary.UwSummary` to be generated.

The UW Madison group's LaTeX summary of a GW/HEN joint trigger.

```
COOLDOWN_PARAMS = CoolDownParams(base=20, increment=5, maximum=14400)
DEPENDENCIES = (<class 'llama.files.lvc_gcn_xml.LvcGcnXml'>,)
FILENAME = 'uw_summary.tex'
MANIFEST_TYPES = (<class 'llama.files.uw_summary.UwSummary'>,)
UR_DEPENDENCIES = (<class 'llama.files.lvc_gcn_xml.LvcGcnXml'>,)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.lvc_gcn_xml.LvcGcnXml'>})
property p_value
Extract the UW Madison p-value from their summary TeX file.
```

28.25 `llama.files.ztf_trigger_list` module

Methods and `FileHandler` classes associated with fetching ZTF triggers.

```
class llama.files.ztf_trigger_list.ZtfTriggerList(eventid_or_fh, rundir=None)
Bases: llama.filehandler.JSONFile, llama.files.healpix.skymap.HEALPixPSF
```

Takes a list of dictionaries describing neutrinos and saves it to a text file. Validates the neutrino data to make sure each neutrino contains (minimally) the following properties:

- gps
- ra
- dec

The sigma value is always expected to be negligible and is hence given as a small value (relative to GW skymap pixel sizes).

```
DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)
```

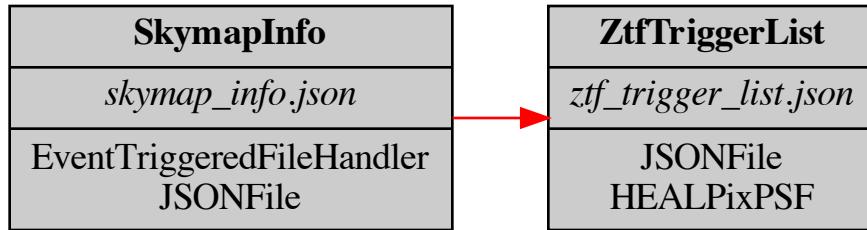


Fig. 114: Required input files for `llama.files.ztf_trigger_list.ZtfTriggerList` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.files.ztf_trigger_list.ZtfTriggerList` to be generated.

```
DETECTORS = (Detector(name='ZTF', abbrev='ztf', fullname='Zwicky Transient Facility', url='https://www.ptf.caltech.edu/page/ztf_technical', summary='ZTF', description='', citations=ImmutableDict({'The unblinking eye on the sky (Bellm and Kulkarni 2017)': 'http://adsabs.harvard.edu/abs/2017NatAs...1E..71B', 'The Zwicky Transient Facility (Bellm 2014)': 'http://adsabs.harvard.edu/abs/2014arXiv1410.8185B', 'The Zwicky transient facility observing system (Smith et al. 2014)': 'http://adsabs.harvard.edu/abs/2014SPIE.9147E..79S', 'The Zwicky Transient Facility Camera (Dekany et al. 2016)': 'http://adsabs.harvard.edu/abs/2016SPIE.9908E..5MD'})),)
```

FILENAME = 'ztf_trigger_list.json'

MANIFEST_TYPES = (<class 'llama.files.ztf_trigger_list.ZtfTriggerList'>,)

SIGMA = 0.1

UR_DEPENDENCIES = (<class 'llama.files.skymap_info.SkymapInfo'>,)

UR_DEPENDENCY_TREE = frozenset({<class 'llama.files.skymap_info.SkymapInfo'>})

class_vetoes = ((<function always_veto>, None),)

property num_triggers
The number of triggers described by this file. Useful mostly for quickly determining if this trigger list is empty.

source_locations()
Returns a list of tuples of (RA, Dec, sigma) for all point-like sources, where (RA, Dec) is the central sky position and sigma is the standard deviation of the Gaussian PSF. Since this depends on the structure of data in the relevant file handler, it must be specified for each subclass.

property template_skymap_filehandler
The HEALPixSkyMapFileHandler whose HEALPix parameters should be used as a template for the HEALPixSkyMap output by this FileHandler. It is assumed that this HEALPixSkyMapFileHandler only returns one skymap in its list; consequently, only the first skymap loaded by this file handler will be used. Note that the template skymap file handler is likely not a dependency of this file handler; it is only used as a default for formatting generated skymaps from this file handler's data.

property triggers
Return a list containing a `ZtfTrigger` object for each trigger; more convenient to work with when writing formulae than dealing directly with the neutrino dictionaries.

`llama.files.ztf_trigger_list.always_veto(eventdir)`
Always veto this FileHandler so that it has to be manually added.

CHAPTER
TWENTYNINE

LLAMA.FLAGS PACKAGE

A mixin for keeping track of the state of flags applied to an event, e.g. whether an event is in ‘test’ or ‘observation’ mode.

```
class llama.flags.FlagDict(eventdir)
    Bases: object
```

A dict-like interface to flags that sets and gets values from an on-disk file containing flags for a specific event directory. *Note that values and keys must both be strings.* For flags in FlagDict.ALLOWED_VALUES, the provided value can only be set to one of the allowed values. Either of these dictionaries can be extended as necessary to provide extra defaults and restrictions across all FlagDict instances.

Parameters `eventdir (str)` – The path to the event directory that contains these flags.

```
ALLOWED_VALUES = frozenset({'BLINDED_NEUTRINOS', 'ICECUBE_UPLOAD', 'MANUAL',
    'ONLINE', 'ROLE', 'UPLOAD', 'VETOED'})

DEFAULT_FLAGS = frozenset({'BLINDED_NEUTRINOS', 'ICECUBE_UPLOAD', 'MANUAL',
    'ONLINE', 'ROLE', 'UPLOAD', 'VETOED'})

PRESETS = FlagPresets(DEFAULT=FlagPreset({'BLINDED_NEUTRINOS': 'false', 'MANUAL':
    'false', 'VETOED': 'false', 'ONLINE': 'true', 'ICECUBE_UPLOAD': 'false', 'UPLOAD':
    'false', 'ROLE': 'test'}), TRIGGERED_INTERNAL=FlagPreset({'UPLOAD': 'true',
    'BLINDED_NEUTRINOS': 'false', 'MANUAL': 'false', 'VETOED': 'false', 'ONLINE':
    'true', 'ICECUBE_UPLOAD': 'false', 'ROLE': 'observation'}), TRIGGERED_PUBLIC=FlagPreset({'ICECUBE_UPLOAD': 'true',
    'UPLOAD': 'true', 'BLINDED_NEUTRINOS': 'false', 'MANUAL': 'false', 'VETOED': 'false', 'ONLINE':
    'true', 'ROLE': 'observation'}, TRIGGERED_TEST=FlagPreset({'MANUAL': 'false',
    'VETOED': 'false', 'ONLINE': 'true', 'ICECUBE_UPLOAD': 'false', 'UPLOAD': 'false',
    'BLINDED_NEUTRINOS': 'true', 'ROLE': 'test'}), RERUN=FlagPreset({'BLINDED_NEUTRINOS': 'false',
    'MANUAL': 'false', 'VETOED': 'false', 'ONLINE': 'true', 'ICECUBE_UPLOAD': 'false',
    'ROLE': 'observation', 'UPLOAD': 'false'}), MANUAL=FlagPreset({'ICECUBE_UPLOAD': 'true',
    'UPLOAD': 'true', 'BLINDED_NEUTRINOS': 'false', 'VETOED': 'false', 'ONLINE': 'true',
    'ROLE': 'observation', 'MANUAL': 'true'}))
```

`items()` → list of D’s (key, value) pairs, as 2-tuples.

`keys()` → list of D’s keys

`update(*E, **F)`

(Same as dict.update.) `self.update([E,]**F)` -> None. Update self from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: `self[k] = E[k]`

If E is present and lacks a .keys() method, then does: for k, v in E: `self[k] = v`

In either case, this is followed by: for k in F: `self[k] = F[k]`

values() → list of D's values

class llama.flags.FlagPreset(*args, description=None, **kwargs)
Bases: [llama.classes.ImmutableDict](#)

A set of key, value pairs for llama flags to serve common use cases. Adds an optional extra `description` field to `ImmutableDict` that can be used to describe the intended use case for a `FlagPreset`.

class llama.flags.FlagsMixin
Bases: `object`

A mixin that can read from and write to flags the 'FLAGS.json' file of any object with an `eventdir` property.

property flags

A list of flags that apply to this instance, e.g. whether this is a testing or production event. See `llama.flags.FlagDict` for details.

Returns flags – A collection of flags for this event directory with a dictionary-like interface. Flags are read from file using standard dictionary syntax, e.g. `self.flags['foo']`, with default flags added from `FlagDict.DEFAULT_FLAGS`. Setting new flags with something like `self.flags['foo'] = bar` will write the new flags to file. Changes to flags are not automatically version-controlled because they are committed as part of event directory state prior to any file generation attempts.

Return type [llama.flags.FlagDict](#)

llama.flags.flag_table(flagdict, color=True, emphasize=())

Print flags into a nice table for terminals. If `color` is True, add terminal color escape codes for emphasis. Specify flags that should be emphasized by listing the flag names in `emphasize` (does not apply if `color` is False).

29.1 llama.flags.cli module

CLI tools for Flags.

class llama.flags.cli.FlagAction(option_strings, dest, nargs=None, const=None, default=None, type=None, choices=None, required=False, help=None, metavar=None)
Bases: `argparse.Action`

Parses flags provided at the command line. Sets the returned argument to an `ImmutableDict` containing the desired flags (either specified by preset or explicit deviations from defaults) after validating that the provided values are valid and quitting with a helpful message if they are not.

class llama.flags.cli.Parsers
Bases: `object`

CLI flags related to flags. Same idea as `llama.cli.Parsers` but with narrower scope.

```
flags = CliParser(prog='sphinx-build', usage=None, description=None,
formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error',
add_help=False)
```

class llama.flags.cli.PrintFlagsAction(option_strings, dest, nargs=None, const=None, default=None, type=None, choices=None, required=False, help=None, metavar=None)

Bases: `argparse.Action`

Action to show flag help and quit.

`llama.flags.cli.postprocess_flags(self: llama.cli.CliParser, namespace: argparse.Namespace)`
Check whether flags have been specified under `--flags`. If not, and if this is an interactive session, prompt the user to manually input flags. If not specified and not interactive, raise an error.

`llama.flags.cli.postprocess_flags_dry_run(_self: llama.cli.CliParser, namespace: argparse.Namespace)`
Print the flags selected by `namespace.flags` and exit without taking further action if `namespace.dry_run_flags`.

LLAMA.INSTALL PACKAGE

Scripts for setting environmental and configuration settings and fetching data files that were too large to be included with the package distribution (or, alternatively, were access-controlled separately from the package distribution).

This method of file storage makes it possible to install a much smaller distribution of pure code and only download pipeline data files as necessary. It also makes it possible to lazy-download the correct file version as-needed.

The `DESTINATIONS` module-level variable maps semantic names of file download destinations (as used in the command-line interface) to tuples of local root directories for storing downloaded files and manifests specifying which files to download and where to situate them relative to those root directories.

`llama.install.install_manifest(root: str, manifest: dict)`

Download files as specified in the `manifest` dict (as returned by `llama.dev.upload.upload_and_get_manifest`) under the `root` directory. Existing files with the expected hash will not be redownloaded, but **existing files will be overwritten**, so exercise caution when calling this function. Files in `root` not mentioned in the manifest will be ignored; delete them manually if they are no longer needed. Also places the `manifest` in the `root` directory, making it possible to check whether post-install downloads are up-to-date.

30.1 `llama.install.manifest` module

Data files to fetch after package installation. These are to be maintained separately from source files. See `llama.dev.upload` for information on how to upload these files and generate a file `MANIFEST` like the one specified here.

CHAPTER
THIRTYONE

LLAMA.INTENT MODULE

Classes for tracking whether files are cooling down after a failed generation attempt or are currently being generated.

class `llama.intent.CoolDown(eventdir, manifest_filehandlers)`
Bases: `llama.classes.RiderFile, llama.classes.JsonRiderMixin`

An object for keeping track of whether a particular file failed to generate recently and determining how long the pipeline should wait before trying again to generate it. Instances of this class are meant to be calculated as properties from FileHandlers and similar classes.

Parameters

- **manifest** (`List`) – List of names of the files that need to be generated (and which might fail, necessitating a cooldown countdown).
- **eventdir** (`str`) – Path to the directory containing that file.

`in_progress()`

Determine whether a cooldown is still in progress for generating this file, i.e. whether this file generation routine failed too recently for another generation attempt.

`rider_fmt = '.{}.cooldown.json'`

`write()`

Write a json file with the name .<FILENAME>.cooldown.json for each file in the manifest containing the time of the last attempt at generating the file as well as the number of consecutive failed attempts at generating the file. Format is

```
{ "num_attempts": 3, "last_attempt": "2016-09-12T17:16:20.337867"  
}
```

Where last_attempt in particular is an ISO formatted time string.

class `llama.intent.CoolDownParams(base, increment, maximum)`
Bases: `tuple`

`base`

Alias for field number 0

`increment`

Alias for field number 1

`maximum`

Alias for field number 2

class `llama.intent.Intent(eventdir, manifest_filehandlers)`
Bases: `llama.classes.RiderFile, llama.classes.JsonRiderMixin`

An object tracking intent to make a file. Tracks whether another process should start making this file. Saves the timestamp at which intent expires to a rider file for each file in a `FileHandler`'s manifest.

`in_progress()`

Check whether this file is already being generated and has not timed out yet.

`rider_fmt = '.{}.intent.json'`

`write(timeout)`

Write the `timeout` (as a UNIX timestamp) at which this file has timed out and another file can attempt to generate it.

`llama.intent.utcnow()`

Return a new datetime representing UTC day and time.

CHAPTER
THIRTYTWO

LLAMA.LISTEN PACKAGE

Tools for handling incoming alerts.

class llama.listen.Parsers

Bases: object

Parsers for new trigger listeners.

```
rundir = CliParser(prog='sphinx-build', usage=None, description=None,
formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error',
add_help=False)
```

llama.listen.flag_preset_freeze_veto(flagdict: llama.flags.FlagDict, presetname: str)

Exactly like flag_update_freeze_veto, but instead of providing the dictionary of the flag value changes you want to apply, you provide the name of the llama.flags.FLAG_PRESETS attribute you want to use. Just a shortcut for getting that attribute manually that also logs the preset choice.

llama.listen.flag_update_freeze_veto(flagdict: llama.flags.FlagDict, updatedict: Dict[str, str])

Update a flagdict for some event with the given dictionary of new flag values, but IGNORE any updates to the VETOED flag, since this flag should usually be set manually (and this function is meant to be a way to procedurally apply llama.flags.FLAG_PRESETS to an existing llama.event.Event without accidentally vetoing or un-vetoing the event).

32.1 llama.listen.gcn package

Tools for receiving, parsing, and archiving GCN Notices. When run as a script, starts a GCN VOEvent listener process that listens for GCN triggers and runs all_llama_handlers on those triggers. When run at the command line, starts a GCN listener that reacts to incoming GCN triggers.

class llama.listen.gcn.LlamaHandlers(rundir='/Users/s/.local/share/llama/current_run/', gcn_archive='/Users/s/.local/share/llama/gcn/archive', included=None)

Bases: llama.listen.gcn.LlamaHandlersTuple

A gcn handler (see pygcn documentation) that runs all gcn handlers specified in LlamaHandlers.included on the given payload (the VOEvent XML string) and root (the lxml VOEvent object for the payload). Add handlers either manually or with the llama_handler decorator. All handlers are run in succession, so it's up to you to make sure they don't interfere with each other. Set up as a callable class so that parameters for handlers can be stored in instances (e.g. output directories).

Parameters

- **rundir (str, optional)** – Path to the run directory (i.e. where new LLAMA triggers are saved by the GCN handler). Defaults to DEFAULT_RUN_DIR. This directory is created on instantiation if it does not already exist.

- **gcn_archive** (*str, optional*) – Path to the GCN Notice archive directory (i.e. where ALL new GCN Notices are archived). Defaults to GCN_ARCHIVE. This directory is created on instantiation if it does not already exist.
- **included** (*array-like, optional*) – A tuple of handlers to call when this instance is called; these functions must take `payload` and `root` as their only arguments. Defaults to a list of all methods and callable properties of `LlamaHandlers` whose names start with “handle”, but this can be overridden. Note that you can include other valid GCN handlers that are not `LlamaHandlers` methods if you choose (see the `pygcn` documentation for details on valid GCN handlers).

handle_archive(*payload, root*)

Save a GCN Notice VOEvent to the archive specified by this `LlamaHandlers` instance. Applies to all GCN Notices including ones processed by the pipeline. Will not overwrite existing VOEvent files (since VOEvent filenames are based on IVORNs, which are supposed to be unique). If the filename is already in use, the xml will be logged at the `info` level (useful for production to catch unexpected behavior).

property handle_lvc_initial_or_update

Get a `handle_lvc_gcn` function for GCN Initial or Update notices that saves to `self.rundir` and adds an Advok marker.

property handle_lvc_preliminary

Get a `handle_lvc_gcn` function for GCN Preliminary notices that saves to `self.rundir`.

handle_lvc_retraction(*payload, root*)

Mark all auto-generated events associated with this graceid as vetoed and save the retraction.

property included

Get a list of GCN handlers to include in this overall handler. If this was not specified when `LlamaHandlers` was instantiated, return all GCN handlers defined in `LlamaHandlers` whose names start with “handle”.

class llama.listen.gcn.LlamaHandlersTuple(rundir, gcn_archive, manually_included)

Bases: tuple

gcn_archive

Alias for field number 1

manually_included

Alias for field number 2

rundir

Alias for field number 0

llama.listen.gcn.get_handle_lvc_gcn(filehandler, notice_types, rundir='/Users/s/.local/share/llama/current_run/')

Get a `handle_lvc_gcn` function that outputs triggers to directories in `rundir` using the specified filehandler.

llama.listen.gcn.get_subdir(archive_dir, receipt_time)

Get the subdirectory where a VOEvent would be archived. Since VOEvents are archived in subdirectories (based on date) of the main `archive_directory`, this path depends on the `receipt_time` of the VOEvent.

Parameters

- **archive_dir** (*str*) – The root directory for all GCN Notices.
- **receipt_time** (*datetime.datetime*) – The time of receipt of the GCN Notice.

llama.listen.gcn.run_on_voevent_file(files='*.xml', handler=LlamaHandlers(rundir='/Users/s/.local/share/llama/current_run', gcn_archive='/Users/s/.local/share/llama/gcn/archive', manually_included=None))

Run a handler on VOEvent XML files (as if they had just been received via GCN). By default, runs all LLAMA

handlers (marked with `@llama_handler` decorator) on the given VOEvent.

Parameters

- **`files`** (*str or list, optional*) – A list of filenames to operate on or else a string that can be interpreted as a UNIX-like glob pattern for matching files as handled by `fnmatch` or `glob` (DEFAULT: “`*.xml`”, matching XML files in the current directory). Note that this means you can usually just provide the exact filename (as long as it can’t be interpreted as a UNIX glob pattern).
- **`handler`** (*function, optional*) – The GCN Handler, of the same sort used by `gcn.listen` (DEFAULT: `all_llama_handlers`).

Raises `XMLSyntaxError` – If the files provided are not valid VOEvents, raises an exception once it gets to that VOEvent file.

32.2 `llama.listen.lvalert` package

Tools for listening to LVAAlerts, LIGO/Virgo’s internal GW trigger alert system.

```
class llama.listen.lvalert.Alert(*args, **kwargs)
```

Bases: `dict`

A JSON alert dictionary from LVAAlert. Parses the alert and reads various properties.

`property advok`

Check whether the alert describes an ADVOK label being applied.

`property advreq`

Check whether the alert has received the advocate requested label (ADVREQ) and is therefore a promising trigger.

`property eventid`

Return the preferred event GraceID if this is a superevent or else return this event’s GraceID if it is just a normal event.

`property graceid`

Return the GraceID associated with this trigger; works for both events and superevents.

`property is_superevent`

Check whether the alert describes a superevent.

`property json`

Get the JSON value. If this was originally constructed as JSON, return that; otherwise, try to serialize it. Raises a `TypeError` if this `Alert` cannot be represented as a JSON string.

`property new_skymap`

Check whether the alert describes the availability of a new skymap.

`property production`

Return whether this LVAAlert object is marked as “production”.

`property superid`

Return the superevent GraceID if this is a superevent. Raises a `KeyError` if not available.

```
class llama.listen.lvalert.Client(processor: function, server: str = 'lvalert.cgca.uwm.edu', nodes:
    Tuple[str] = ('cbc_gstlal', 'cbc_mbtaonline', 'cbc_lowmass', 'cbc_pycbc',
    'test_superevent', 'superevent', 'burst_cwb', 'cbc_spiir', 'stc-testnode'),
    heartbeat_interval: float = 180)
```

Bases: `object`

A wrapper around LVAAlertClient that provides an interface for subscribing to LVAAlert nodes and starting a listener. Specify the new alert processor as processor and the URL of the LVAAlert server to use as server.

HEARTBEAT_NODE = 'stc-testnode'

check_heartbeat(*timeout*=20)

Publish a test message to `self.HEARTBEAT_NODE` and wait for a that same message to be parroted back by this instance's `listen` function to `self.heartbeat_queue`. If, after waiting for `timeout` seconds, nothing has been received, or if the received message does not match the random string sent out, return `False`; otherwise, return `True`. You should probably restart your subscriptions if `check_heartbeat` fails.

This heartbeat check creates a new LVAAlertClient and reconnects each time it is called. It is therefore fairly slow and should only be called at most every few minutes or so.

heartbeat_interval: float

heartbeat_message()

Get a random message with which to run a heartbeat check.

heartbeat_queue: multiprocessing.context.BaseContext.Queue

listen(*sleep*=3)

Connect to `self.server`, subscribe to desired `self.nodes`, and start listening for new LVAAlerts, reacting with `self.processor`. Sleep for `sleep` seconds between checks for `KeyboardInterrupt` or `SystemExit`. Will keep checking to make sure all original subscriptions are still active. Will only quit on keyboard interrupt or system exit call.

nodes: Tuple[str]

processor: function

server: str

`llama.listen.lvalert.get_client(server: str)`

Get an LVAAlertClient at the specified `server` URL using the login credentials stored in the `LVALERT_USERNAME` and `LVALERT_PASSWORD` environmental variables (if available).

`llama.listen.lvalert.process_alert_json(node, alert_json, rundir)`

Process an LVAAlert JSON string as read from `stdin` when called by `lvalert-listen`.

CHAPTER
THIRTYTHREE

LLAMA.LOCK MODULE

A mechanism for marking a file as locked, i.e. not eligible for automatic regeneration, or as obsolete, i.e. eligible for automatic regeneration. It can still manually be changed in either case.

class `llama.lock.LockHandler(eventdir, manifest)`
Bases: `llama.lock.LockHandlerTuple`

A class for marking a set of paths as locked, i.e. not eligible for automated obsolescence and regeneration. At time of writing, this is accomplished by creating an empty lockfile that `llama.filehandler.FileHandler.is_obsolete` will check for. The `manifest` should contain filenames (as returned by `basename` or `llama.filehandler.FileHandler.filename`). Is also used for precomputing obsolescence values and storing the results in `obsolescence_files`.

property `is_locked`
Check whether the files in this manifest are locked.

lock()
Mark the files in this manifest as locked. Since locking prevents obsolescence, marking a file as locked will also write cache the obsolescence value as `False`.

property `lockfiles`
A list of lockfiles associated with this manifest. If any of these exists, all files in the manifest are considered locked. Create these files to mark them as locked.

property `lockpaths`
Full paths to lockfiles for this manifest.

property `obsolescence`
Get the precomputed obsolescence value for this manifest (if it exists); raise an `NotFoundError` if none is available and a `ValueError` if a valid value could not be read from the obsolescence file.

property `obsolescence_files`
A list of obsolescence file names associated with this manifest. If any of these exists, the obsolescence status of the manifest will be read from it. Create these files to store precomputed or manual obsolescence values.

property `obsolescence_paths`
Full paths to obsolescence files for this manifest.

record_obsolescence(*obsolete*)
Store the file's obsolescence values. Will ignore any locks, which separately veto obsolescence.

remove_all_obsolescence()
Delete all obsolescence cache files for this event.

remove_obsolescence()
Delete obsolescence files recording obsolescence state.

unlock()

Mark the files in this manifest as unlocked. Will remove any cached obsolescence values (since obsolescence must now be recalculated).

class llama.lock.LockMixin

Bases: object

A mixin that lets `FileHandler`-like objects (producing a manifest and eventdir) mark files as locked, i.e. ineligible for automatic obsolescence checks and subsequent regeneration. This functionality is added via a `lock` property on the `FileHandler`-like object with sub-methods implementing its functionality.

property lock

Return a `LockHandler` instance corresponding to this `FileHandler` and all other file paths in its manifest.

llama.lock.cache_obsolescence(func)

Decorator for a `LockMixin` instance's obsolescence check method that marks all files in the manifest with the returned obsolescence value of the decorated method. Immediately returns a cached obsolescence value if stored.

llama.lock.trash_obsolescence(func)

Decorator for a `LockMixin` instance's generation method that removes all obsolescence files for the instance's manifest. Used for cache invalidation.

CHAPTER
THIRTYFOUR

LLAMA.META MODULE

Tools for dealing with metadata of FileHandlers.

class `llama.meta.MetaData(eventdir, manifest_filehandlers)`

Bases: `llama.classes.RiderFile, llama.classes.JsonRiderMixin`

A class for getting metadata about the output file of a `FileHandler` and determining things like file obsolescence.

property `checksums`

Read the stored sha256 checksums of the contents of this `MetaData` instance's `manifest_filehandlers`. The keys of this `dict` are `FileHandler.clsname` values and the values are the corresponding checksums.

Raises

- **FileNotFoundException** – If the metadata file does not exist.
- **KeyError** – If the file format of the metadata files has changed and they do not contain the hash data.
- **IOError** – If the checksums have not been computed for all files in the manifest.

property `dep_checksums`

Read the stored sha256 sums of the contents of the input files (i.e. `DEPENDENCIES`) for this `MetaData` instance's `manifest_filehandlers`. The keys of this `dict` are `FileHandler.clsname` values and the values are the corresponding sha sums. See `FileHandler.dep_checksums` return values for details and use cases.

Raises

- **FileNotFoundException** – If the metadata file does not exist.
- **KeyError** – If the file format of the metadata files has changed and they do not contain the hash data.

property `dep_subset_checksums`

Read the stored sha256 sums of the `subset` of the contents of the input files (i.e. `DEPENDENCIES`) for this `MetaData` instance's `manifest_filehandlers`. The keys of this `dict` are `FileHandler.clsname` values and the values are the corresponding sha sums computed from the `subset` of the input data that is actually used by the `generate` methods of each `FileHandler` in `manifest_filehandlers`. See `FileHandler.dep_checksums` return values for details and use cases.

Raises

- **FileNotFoundException** – If the metadata file does not exist.
- **KeyError** – If the file format of the metadata files has changed and they do not contain the hash data.

property flags

Return a dictionary of the flags defined for this event at the time when the file was generated.

property generation_time

Get the `datetime` object representing the UTC time at which this file was generated according to its rider metadata file. Raises an `IOError` if the metadata file does not exist and a `KeyError` if the `generation_time` information is not present in the metadata file.

property generation_uuid

Get a UUID for this generation event. Raises an `IOError` if the metadata file does not exist and a `KeyError` if the `generation_uuid` information is not present in the metadata file.

rider_fmt = '.{}metadata.json'

property wall_time

Time in seconds spent running `_generate` to make this file. Raises an `IOError` if the metadata file does not exist and a `KeyError` if the `wall_time` information is not present in the metadata file.

write(wall_time)

Write a metadata file for this `FileHandler`. Almost all metadata about a generation attempt can be determined from either the LLAMA execution context or the calling `FileHandler`; the exception is the `wall_time`, i.e. time spent generating this output file; this value must be provided as an argument.

LLAMA.PIPELINE MODULE

Analysis Pipeline instances are Directed Acyclic Graphs (DAGs) specifying sets of FileHandler classes that together form an analysis pipeline. Use Pipeline instances to specify exactly which analysis steps you want to run; select a subset of the maximal DEFAULT_PIPELINE if you only need certain analysis outputs.

35.1 Pipelines

llama.pipeline.DEFAULT_PIPELINE : Pipeline

llama.pipeline.LLAMA2 REVIEW_PIPELINE : Pipeline

llama.pipeline.SUBTHRESHOLD_PIPELINE : Pipeline

```
class llama.pipeline.FileHandlerSubgraphAction(option_strings, dest, nargs=None, const=None,
                                              default=None, type=None, choices=None,
                                              required=False, help=None, metavar=None)
```

Bases: argparse.Action

Downselect for a subgraph on the specified filehandlers, returning a `llama.pipeline.Pipeline` containing the specified filehandlers and all of their ancestors if the option string starts with + or just the specified filehandlers if the option_string starts with -; combine results so that both types of flags can be used in the same command-line invocation.

class llama.pipeline.Parsers

Bases: object

A collection of CLI parsers implementing pipeline related functionality. Note that you will need to specify `'prefix_chars="-+'`` to use the ``+filehandlers`` functionality. See `llama.cli.Parsers` for more information.

```
pipeline = CliParser(prog='sphinx-build', usage=None, description=None,
formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error',
add_help=False)
```

class llama.pipeline.Pipeline(*args, **kwargs)

Bases: `llama.classes.ImmutableDict`, `llama.classes.NamespaceMappable`

A pipeline specifies a specific set of data inputs and the functions that act on them in terms of intermediate data products and the functions used to generate them in a Directed Acyclic Graph (DAG); these products are bundled into FileHandlers. FileHandlers are graph nodes with DEPENDENCIES (edges) specified. A Pipeline DAG can be built purely by specifying the specific FileHandlers which can be done trivially and clearly at the file-system level by putting the FileHandler code into a single directory for each pipeline.

Parameters

- **kwargs** (`dict`) – Names of FileHandler classes mapped to the classes themselves.

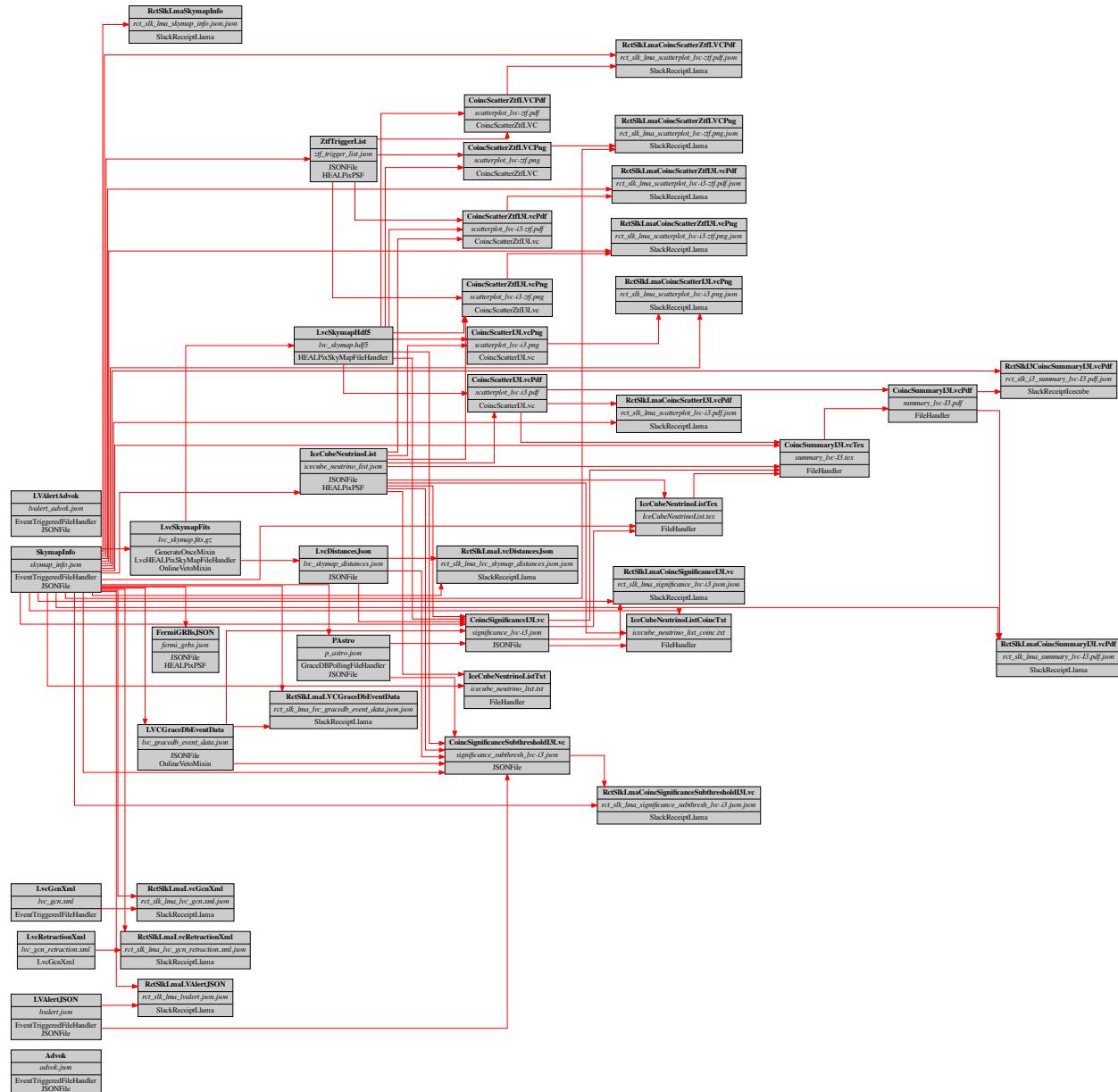
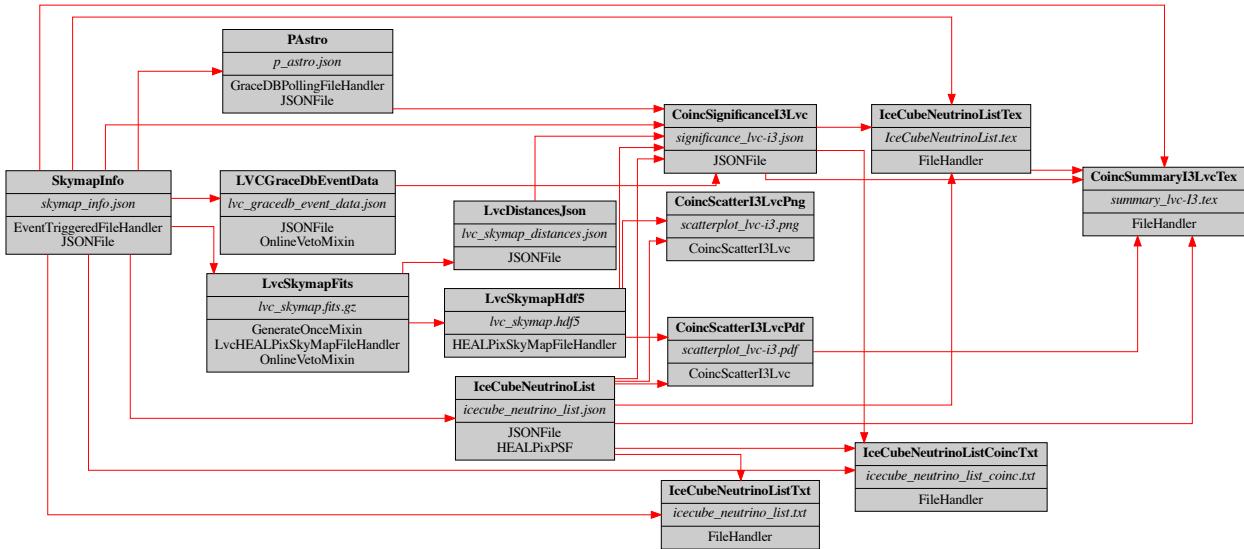
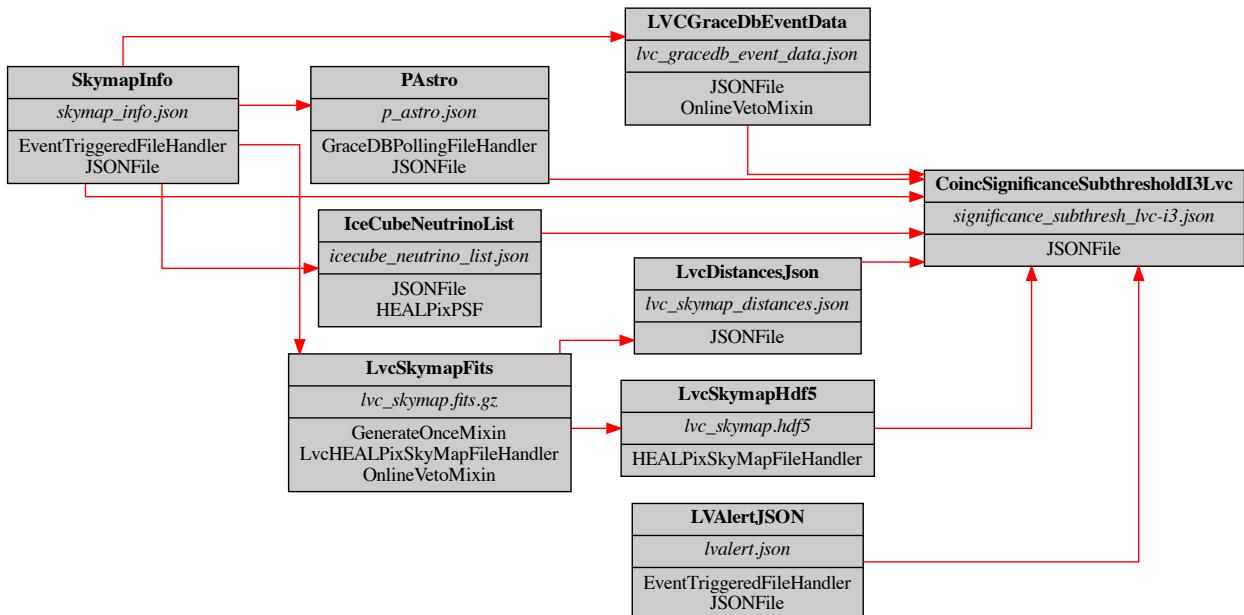


Fig. 1: FileHandler dependency relations in `llama.pipeline.DEFAULT_PIPELINE`.


 Fig. 2: FileHandler dependency relations in `llama.pipeline.LLAMA2 REVIEW PIPELINE`.

 Fig. 3: FileHandler dependency relations in `llama.pipeline.SUBTHRESHOLD PIPELINE`.

- **args** (*array-like*) – FileHandler classes. The `__name__` of each FileHandler will be used as the key.

Returns `pipeline` – A new Pipeline instance containing all of the FileHandler classes specified in `args` and `kwargs`.

Return type `Pipeline`

Raises `TypeError` – If there are any name collisions between classes in the input `args` and `kwargs`, if any of the FileHandler classes it contains are abstract (non-implemented) classes, or if any of the FileHandler classes it contains have missing `required_attributes`.

check_consistency(*other*)

Check whether two Pipeline instances use the same keys to describe the same FileHandler classes, raising a `ValueError` if they don't.

dependency_graph(*outfile: Optional[str] = None*, *title: str = 'Pipeline'*, *url: Optional[function] = None*, *bcolor: str = 'black'*)

Return a graphviz .dot graph of DEPENDENCIES between file handlers in this pipeline. Optionally plot the graph to an output image file visualizing the graph.

Optional file extensions for *outfile*:

- *dot*: just save the dotfile in .dot format.
- *png*: save the image in PNG format.
- *pdf*: save the image in PDF format.
- *svg*: save the image in svg format.

Parameters

- **outfile** (*str, optional*) – If not provided, return a string in .dot file format specifying graph relationsIf an output file is specified, infer the filetype and write to that file.
- **title** (*str, optional*) – The title of the pipeline graph plot.
- **url** (*FunctionType, optional*) – A function taking FileHandler classes as input and returning a URL that will be added to each FileHandler class's node in the output graph. Allows you to add links. If not included, URLs will not be included.
- **bcolor** (*str, optional*) – The background color to use for the generated plot.

Returns `dot` – The dependency graph in .dot format (can be used as input to `dot` at the command line). This is returned regardless of whether an *outfile* is specified.

Return type `str`

downselect(*invert=False*, *reducer=<built-in function all>*, ***kwargs*)

Return a Pipeline instance whose FileHandler classes match ALL the given query parameters.

Parameters

- **invert** (*bool, optional*) – Invert results. (Default: False)
- **reducer** (*function, optional*) – Specify any builtin to match if any check passes. Specify `all` to match only when every check passes. (Default: `all`)
- **type** (*type, optional*) – The type of the FileHandler must exactly match the given FileHandler.
- **typename** (*str, optional*) – The FileHandler type's name must match this string.

- **subclass** (*type, optional*) – The FileHandler must be a subclass of this FileHandler.
- **subgraph** (*type, optional*) – The FileHandler must be either this FileHandler or one of its UR_DEPENDENCIES; use this to make a Pipeline that only generates the subgraph leading to this FileHandler.

file_handler_instances(*args, **kwargs)

Return a FileHandlerMap with FileHandler instances sharing the same initialization arguments, e.g. for FileHandler instances that all refer to the same event.

classmethod from_module(*module*)

Create a pipeline by extracting all FileHandler objects from a given submodule.

class llama.pipeline.PipelineAction(*option_strings, dest, nargs=None, const=None, default=None, type=None, choices=None, required=False, help=None, metavar=None*)

Bases: argparse.Action

Set the pipeline from the available pipelines defined in llama.pipeline.

class llama.pipeline.PrintDefaultPipeline(*option_strings, dest, nargs=None, const=None, default=None, type=None, choices=None, required=False, help=None, metavar=None*)

Bases: argparse.Action

Print DEFAULT_PIPELINE in a readable format and quit.

llama.pipeline.postprocess_pipeline_dry_run(*self: llama.cli.CliParser, namespace: argparse.Namespace*)

Print the filehandlers in namespace.pipeline and quit without evaluating further if namespace.dry_run_pipeline is True.

llama.pipeline.postprocess_pipeline_selection(*self: llama.cli.CliParser, namespace: argparse.Namespace*)

Add any filehandlers defined in namespace.filehandlers to namespace.pipeline. If namespace.pipeline is not defined, set it to the DEFAULT_PIPELINE.

LLAMA.POLL PACKAGE

Periodically check for data from APIs.

36.1 `llama.poll.gracedb` package

Poll GraceDB for new superevents that have been marked EM_READY and have had a GCN Notice VOEvent generated. Download and handle those GCN Notices. Runs as a script checking for new VOEvents when called from the command line.

`llama.poll.gracedb.fetch_recent(rundir='/Users/s/.local/share/llama/current_run/')`

Fetch and store recent superevents.

`llama.poll.gracedb.get_skymap_name(superevent)`

Get candidate skymap filenames for this superevent dictionary.

`llama.poll.gracedb.gpsnow()`

Get the current GPS time (rounded up to the nearest integer for GraceDB's querying interface).

`llama.poll.gracedb.gracedb_query(gps, lookback=200000)`

Get a GraceDB-compatible query string for finding recent superevents that happened in the last `lookback` seconds before the GPS time `gps`.

`llama.poll.gracedb.superevents()`

Get a generator of superevents with skymaps that stops iterating once we've gone through all the events in the last LOOKBACK seconds.

CHAPTER
THIRTYSEVEN

LLAMA.RUN PACKAGE

Utilities for working with a Run Directory, i.e. a directory containing multiple subdirectories, one per event. Includes tools for automatically finding events that can be updated and keeping them updated. Useful for running a pipeline or batch job.

```
class llama.run.ParseRunsAction(option_strings, dest, nargs=None, const=None, default=None, type=None,
                                 choices=None, required=False, help=None, metavar=None)
```

Bases: argparse.Action

Take a bunch of pathnames and parse them into Run instances with associated `eventid` glob filters. See Parsers docstring for details.

```
class llama.run.Parsers(downselect=None)
```

Bases: object

Each LLAMA trigger gets its own directory. The name of this directory is called the `eventid` and the trigger itself is a LLAMA Event (see: `llama.event`). For a given LLAMA run, all event directories should go in a command directory called a “run directory”; the collection of events is called a Run (see: `llama.run`). Most things the pipeline does work on a single Run and are meant to affect one or more matching Event instances. When you specify directories, you are implicitly specifying the Run (i.e. collection of triggers) as well as a UNIX-style glob (like the asterisk matching all files, `*`) which describes the `eventid` pattern you want to match. For example, matching all event IDs that start with “S” (corresponding to O3 LIGO/Virgo superevents) would require using `S*` as your event glob.

If you want to explicitly print which currently-existing Event directories will be impacted by the arguments you provide, you can use `--dry-run-dirs` to print the impacted directories and exit without taking further action. This is good practice while getting used to this interface.

The syntax for specifying the Run and Event glob is the path of the run directory followed by a slash followed by the event glob with **no slash at the end** (be sure to escape the `*` so the shell doesn’t expand it):

```
'/run/directory/event*glob'
```

Specify **only** the event glob by leaving the run directory out but keeping the leading `/` (if for some insane reason your root directory is your run directory, a double-leading `/` will communicate your perverse desire). In this case the default Run directory `/Users/s/.local/share/llama/current_run/` is implied, so the following are equivalent:

```
'/event*glob'
/Users/s/.local/share/llama/current_run/'event*glob'
```

Specify **only** the Run directory by leaving a trailing slash and omitting the event glob; in this case, the default event glob `*` will be used, so the following are equivalent:

```
/run/directory/  
/run/directory/*
```

You can use relative paths for the Run directory, the final part of the path will **not** be expanded and will be treated as the base directory. The only exception to this is if you are using relative paths and don't put *any* / in the specified path, in which case the relative path will be expanded. This allows the common and intuitive behavior of running specific events in the current directory when you pass their name alone, or alternatively to treat the current directory as the only event directory by passing a single . as the run argument. Something like ./, however, will be interpreted as meaning you want the *current* directory to be the run directory only matching Event ids of ..

The following examples assume you are currently in the event directory /some/directory/. Let's say this is the event directory, and you want to update **only** the contents of this directory. You can specify the run as /some/ and the event glob as directory with either of the following commands paths:

```
.
```

```
/some/directory
```

Alternatively, if /some/directory/ is a run directory, and you want to affect the event directories it contains that match the default event glob *, you can run use any of the following (note again that the event glob is in quotes to prevent your shell from expanding it into multiple arguments):

```
./  
./*  
/some/directory/  
/some/directory/*
```

If you want to use the name of the current directory as your event glob (so that only eventids that have the *same* basename as your current directory are used) while **keeping** the default run directory /Users/s/.local/share/llama/current_run/, you would have to place a leading slash followed by the actual name of the run directory; as noted above, /. not work because the dot will be treated literally as the eventid you want to use. (Note that you usually wouldn't want to do this; why would you be in this directory if you want to operate on an event stored in a different run directory?):

```
/directory  
/Users/s/.local/share/llama/current_run/directory
```

You can further specify which types of events should be processed by specifying --downselect followed by a string to be passed as the arguments to Run.downselect (run --print-downselections to see possible options).

See `llama.run` and `llama.event` for more information on Run and Event objects.

property `eventfiltering`

A CliParser to be used for downselecting runs and events.

```
class llama.run.PrintDownselectionsAction(option_strings, dest, nargs=None, const=None,  
                                         default=None, type=None, choices=None, required=False,  
                                         help=None, metavar=None)
```

Bases: argparse.Action

Print a dedented docstring for Run.downselect and exit.

```
class llama.run.Run(rundir='/Users/s/.local/share/llama/current_run/', pipeline=frozenset('Advok',
    'CoincScatterI3LvcPdf', 'CoincScatterI3LvcPng', 'CoincScatterZtfI3LvcPdf',
    'CoincScatterZtfI3LvcPng', 'CoincScatterZtfLVCPdf', 'CoincScatterZtfLVCPng',
    'CoincSignificanceI3Lvc', 'CoincSignificanceSubthresholdI3Lvc',
    'CoincSummaryI3LvcPdf', 'CoincSummaryI3LvcTex', 'FermiGRBsJSON',
    'IceCubeNeutrinoList', 'IceCubeNeutrinoListCoincTxt', 'IceCubeNeutrinoListTex',
    'IceCubeNeutrinoListTxt', 'LVAAlertAdvok', 'LVAAlertJSON', 'LVCGraceDbEventData',
    'LvcDistancesJson', 'LvcGcnXml', 'LvcRetractionXml', 'LvcSkymapFits', 'LvcSkymapHdf5',
    'PAstro', 'RctSlkI3CoincSummaryI3LvcPdf', 'RctSlkLmaCoincScatterI3LvcPdf',
    'RctSlkLmaCoincScatterI3LvcPng', 'RctSlkLmaCoincScatterZtfI3LvcPdf',
    'RctSlkLmaCoincScatterZtfI3LvcPng', 'RctSlkLmaCoincScatterZtfLVCPdf',
    'RctSlkLmaCoincScatterZtfLVCPng', 'RctSlkLmaCoincSignificanceI3Lvc',
    'RctSlkLmaCoincSignificanceSubthresholdI3Lvc', 'RctSlkLmaCoincSummaryI3LvcPdf',
    'RctSlkLmaLVAAlertJSON', 'RctSlkLmaLVCGraceDbEventData',
    'RctSlkLmaLvcDistancesJson', 'RctSlkLmaLvcGcnXml', 'RctSlkLmaLvcRetractionXml',
    'RctSlkLmaSkymapInfo', 'SkymapInfo', 'ZtfTriggerList')), downselection=())
Bases: llama.run.RunTuple
```

A single directory containing multiple event directories combined with a pipeline (i.e. a selection of analysis steps to use) and a set of downselection criteria for picking events:

Run Directory |— Event Directory 1 |— Event Directory 2 |— Event Directory 3

This should ordinarily correspond to a run of some sort (an observing run, engineering run, offline run, test run, etc.) where the events are somehow related. Since this class mostly just provides methods for organizing and selecting Event instances with tailored Pipeline instances, it's up to you to decide how to best organize a run. Run objects are immutable to simplify hashing and uniqueness checks.

These tools allow the user to conveniently check on the status of all events in a given Run. A dictionary of downselection arguments (as fed to `downselect`) can be used to restrict the set of events that will be returned events.

Parameters

- **rundir** (*str*) – The directory where all events are stored. Files for individual events are stored in per-event subdirectories of `rundir`. Will be converted to a canonical path with `os.path.realpath` to help ensure unique Run definitions.
- **pipeline** ([llama.pipeline.Pipeline](#), *optional*) – A Pipeline instance holding FileHandler classes that should be used for this analysis. Defaults to the main pipeline in production use.
- **downselection** (*tuple*, *optional*) – A tuple of dictionaries of keyword arguments of the type passed to `downselect`. The events returned by `events` will match these downselection criteria with each downselection dict applied in the order they appear in this argument (to allow more complex chained downselections). You probably don't want to manually specify this; a more pythonic way to provide downselection arguments is to use the `downselect` method to return a downselection from a starting Run.

`downselect(**kwargs)`

Get another Run instance identical to the current one but with the following downselection criteria applied to the Event instances returned by `self.events`. Can also specify a sorting function and a maximum number of returned values:

Parameters

- **invert** (*bool*, *optional*) – Invert what matches and what doesn't. Default: False
- **eventid_filter** (*str*, *optional*) – A glob (as taken by `fnmatch`) that the `eventid` must match.

- **fileexists** (*str, optional*) – The event directory contains a file with this name.
- **fexists** (`llama.filehandler.FileHandler, optional`) – The eventdir contains the file for this FileHandler.
- **fnameexists** (*str, optional*) – The eventdir contains the file for the FileHandler with this name.
- **fhmeta** (`llama.filehandler.FileHandler, optional`) – The eventdir contains a metadata rider for the file for this FileHandler.
- **fnamemeta** (*str, optional*) – The eventdir contains a metadata rider for the file for the FileHandler with this name.
- **vetoed** (*bool, optional*) – Whether the events have been vetoed by the VETOED flag or not.
- **manual** (*bool, optional*) – Whether the events have been marked as manual by the MANUAL flag.
- **modbefore** (*float, optional*) – Select events whose directory modtimes were before this timestamp.
- **modafter** (*float, optional*) – Select events whose directory modtimes were after this timestamp.
- **sec_since_mod_gt** (*float, optional*) – Select events whose directory modtimes are more than this many seconds ago.
- **sec_since_mod_lt** (*float, optional*) – Select events whose directory modtimes are less than this many seconds ago.
- **v0before** (*float, optional*) – Select events whose first event state version was generated before this timestamp. Will **IGNORE** directories that do not have any versioned files.
- **v0after** (*float, optional*) – Select events whose first event state version was generated after this timestamp. Will **IGNORE** directories that do not have any versioned files.
- **sec_since_v0_gt** (*float, optional*) – Select events whose first event state version was generated more than this many seconds ago. Will **IGNORE** directories that do not have any versioned files.
- **sec_since_v0_lt** (*float, optional*) – Select events whose first event state version was generated less than this many seconds ago. Will **IGNORE** directories that do not have any versioned files.
- **sortkey** (*function, optional*) – A function taking Event instances that can be passed to sorted to sort the downselected Event instances. Default: None (i.e. no sorting)
- **reverse** (*bool, optional*) – Whether to reverse the order of sorting (i.e. put the results in descending order) before applying limit. Default: False
- **limit** (*int, optional*) – Return up to this number of events. Most useful if sortkey has also been provided. Default: None (i.e. no limit)

downselect_pipeline(*invert=False, **kwargs*)

Return a Run instance with a pipeline that has been downselected using Pipeline.downselect.

property events

Return a list of events in this run directory with self.downselection criteria applied (see downselect for a list of possible downselection criteria).

Parameters

- **sortkey** (*function, optional*) – A sorting key (as passed to `sorted`) to use to sort the returned events. If none is provided, the events will be sorted based on astrophysical event time using `Event.gpstime`; beware that an error will be raised if this quantity is ill-defined for ANY of the returned events.
- **reverse** (*bool, optional*) – Whether to reverse the default sort order, i.e. put in descending order. True by default so that the most-recently-occurring events are first in the list.

update()

Get a list of Event instances matching this Run instance’s downselection criteria and update each event directory.

property vis

A collection of visualization methods for this Run instance.

class llama.run.RunTuple(rundir, pipeline, downselection)

Bases: tuple

downselection

Alias for field number 2

pipeline

Alias for field number 1

rundir

Alias for field number 0

class llama.run.RunVisualization(run)

Bases: object

Provide methods for visualizing the status of a run directory.

finished(outfile=None)

Create a bar plot showing the proportion of complete to incomplete files for each FileHandler in this Run instance.

Parameters `outfile` (*str, optional*) – If provided, save the plot to this filename.

Returns `fig` – The bar plot figure.

Return type matplotlib.figure

wall_times(outfile=None)

Create histograms of wall times (i.e. how long each file took to generate) for each FileHandler in this Run instance.

Parameters `outfile` (*str, optional*) – If provided, save all plots as PNG files to a gzipped tarfile with this filename.

Returns `plots` – A dictionary of `matplotlib.figure` instances whose keys are the names of each FileHandler class and whose values are histograms of wall times for each FileHandler class.

Return type dict

llama.run.downselect_events(events, **kwargs)

Take a list of events and downselect them using the checks described in `Run.downselect`.

```
llama.run.past_runs(paths=('/Users/s/.local/share/llama/past_runs'), pipeline=frozenset({'Advok',
    'CoincScatterI3LvcPdf', 'CoincScatterI3LvcPng', 'CoincScatterZtfI3LvcPdf',
    'CoincScatterZtfI3LvcPng', 'CoincScatterZtfLVCPdf', 'CoincScatterZtfLVCPng',
    'CoincSignificanceI3Lvc', 'CoincSignificanceSubthresholdI3Lvc',
    'CoincSummaryI3LvcPdf', 'CoincSummaryI3LvcTex', 'FermiGRBsJSON',
    'IceCubeNeutrinoList', 'IceCubeNeutrinoListCoincTxt', 'IceCubeNeutrinoListTex',
    'IceCubeNeutrinoListTxt', 'LVAAlertAdvok', 'LVAAlertJSON', 'LVCGraceDbEventData',
    'LvcDistancesJson', 'LvcGcnXml', 'LvcRetractionXml', 'LvcSkymapFits', 'LvcSkymapHdf5',
    'PAstro', 'RctSlkI3CoincSummaryI3LvcPdf', 'RctSlkLmaCoincScatterI3LvcPdf',
    'RctSlkLmaCoincScatterI3LvcPng', 'RctSlkLmaCoincScatterZtfI3LvcPdf',
    'RctSlkLmaCoincScatterZtfI3LvcPng', 'RctSlkLmaCoincScatterZtfLVCPdf',
    'RctSlkLmaCoincScatterZtfLVCPng', 'RctSlkLmaCoincSignificanceI3Lvc',
    'RctSlkLmaCoincSignificanceSubthresholdI3Lvc', 'RctSlkLmaCoincSummaryI3LvcPdf',
    'RctSlkLmaLVAAlertJSON', 'RctSlkLmaLVCGraceDbEventData',
    'RctSlkLmaLvcDistancesJson', 'RctSlkLmaLvcGcnXml', 'RctSlkLmaLvcRetractionXml',
    'RctSlkLmaSkymapInfo', 'SkymapInfo', 'ZtfTriggerList'}))
```

Get a dictionary of run names and corresponding Run instances, looking for run directories in the specified paths.

Parameters

- **paths** (*tuple, optional*) – Directories in which to search for past run directories.
- **pipeline** ([llama.pipeline.Pipeline](#)) – The pipeline to use for the returned Run instances in `runs`.

Returns `runs` – A dictionary of past runs whose keys are the name of the rundir and whose values are the corresponding Run. Absolute paths are used as the keys.

Return type dict

```
llama.run.postprocess_downselect(_self: llama.cli.CliParser, namespace: argparse.Namespace)
```

If `namespace.downselect` is not None, parse it as a comma-separated list of key=value pairs, where value will be parsed as a boolean if it equals either True or False and as a string otherwise. Use these arguments to downselect each of the runs specified in `namespace.run`.

```
llama.run.postprocess_dry_run(_self: llama.cli.CliParser, namespace: argparse.Namespace)
```

If `--dry-run-dirs` is true, print the directories that would be affected by the given arguments and quit without taking further action. If you want to extend this, print more dry run information and then call this function to print run/event information before quitting.

```
llama.run.v0ts(event)
```

Return the timestamp of the first file version commit, catching the error if the event does not have versioning initialized/has no versions and returning False.

LLAMA.SERVE PACKAGE

Tools for interacting with LLAMA servers in non-SSH-based ways, e.g. websites etc.

38.1 llama.serve.gui package

Tools related to the interactive status/results website maintained for the current LLAMA run.

`llama.serve.gui.gui_url(obj, linktype=None)`

Get a URL to the LLAMA GUI webpage running on this server. Use this to generate links to LLAMA Run, Event, and FileHandler webpages.

Parameters

- **obj (object)** – A LLAMA object. If this is has a `rundir` attribute, try to find the run name that LLAMA GUI uses. If it also has an `eventid` attribute, include that in the returned URL. If it also has a `FILENAME` attribute, include that as well.
- **linktype (str, optional)** – Optionally manually specify the output type as '`run`', '`event`', or '`filehandler`'. Use this if you want to e.g. get a link to the webpage for an Event when providing a `FileHandler`. Will not work if `obj` does not contain the data required to construct the requested `linktype` (e.g. `linktype='filehandler'` will fail if `obj` is a `Run`).

Returns `url` – A URL linking to the LLAMA summary web page for the provided object.

Return type str

Raises `ValueError` – If the `rundir` for the provided object is not in a directory served by `llama.serve.gui`, if `obj` has no `rundir` attribute, or if the requested `linktype` cannot be constructed from `obj`.

38.1.1 llama.serve.gui.wsgi package

Provide an entrypoint for the WSGI launcher.

38.1.2 `llama.serve.gui.domain` module

Get the domain on which this server is running from the environment.

38.2 `llama.serve.jupyter` package

Launch a Jupyter Notebook server. Specify the domain using environment variable LLAMA_DOMAIN (default: localhost) and the port as LLAMA_JUP_PORT (default: 8080).

`llama.serve.jupyter.load_notebook(remote, local, readonly=True, clobber=True)`

Copy a distribution LLAMA JUPYTER notebook to the output directory (fetching it first from LLAMA S3 if necessary), set it as trusted, and set its permissions to readonly. Deletes the old notebook if present.

Parameters

- `remote` (`RemoteFileCacher`) – A `RemoteFileCacher` instance pointing to the file you want to load.
- `local` (`str or Path`) – Local path *relative* to the `OUTPUT_DIR` that you'd like to save the remote file as.
- `readonly` (`bool, optional`) – If `True`, set the notebook to readonly mode so that user's can't accidentally break it. Set this to `False` to allow editing when making changes to the notebook notebook (and remember to commit those changes with `llama dev upload`).
- `clobber` (`bool, optional`) – If `True`, overwrite any existing notebook files. In production, you usually want this to be `True` since it will ensure you get the latest version of the documentation. While developing, you probably want this to be `False` so that you can keep incrementally modifying your notebook until you're ready to commit the changes to LLAMA S3.

Returns `local` – The path to the local Jupyter notebook.

Return type `pathlib.Path`

`llama.serve.jupyter.running_servers()`

Get a list of running servers as stored in `SERVER_LIST_FILE`. Returns a list of tuples of the form (`server_url`, `mounted_path`) where `server_url` is the URL that can be used to access the server and `mounted_path` is the path on the server that the jupyter notebook has mounted. Returns `None` if `SERVER_LIST_FILE` does not exist.

38.2.1 `llama.serve.jupyter.logs` module

`llama.serve.jupyter.logs.setup_tail_widgets()`

38.2.2 `llama.serve.jupyter.utils` module

Utilities for working with LLAMA in Jupyter notebooks.

`llama.serve.jupyter.utils.create_download_link(filehandler)`

Create a Jupyter notebook compatible download link for the file contents of a filehandler. Only works in an IPython notebook.

Parameters `filehandler` (`llama.filehandler.FileHandler`) – The `FileHandler` whose data you want to download.

Raises `FileNotFoundException` – If the file to be downloaded does not exist.

LLAMA.TEST PACKAGE

Tests for LLAMA, to be run with `pytest`.

39.1 `llama.test.test_files` package

Unit tests for individual pipeline `FileHandler` subclasses and their supporting functions and methods.

Note: do not test expected file generation outputs here; those integration tests should be run as part of the automatic reproducibility tests included in `test_pipeline`. File generation tests in this submodule should cover sanity checks and edge cases.

39.2 `llama.test.test_listeners` package

Test our event listeners.

39.2.1 `llama.test.test_listeners.test_gcn` module

Unit tests for `llama.listen.gcn`.

```
class llama.test.test_listeners.test_gcn.AbstractGcndTester
    Bases: llama.test.test_listeners.test_gcn.AbstractTestLlamaHandlers
```

Test `gcnd`'s ability to listen for parse events. `gcnd` has to be in your “\$PATH” for this to work along with `pygcn-serve`. Tests the same thing as `AbstractFileGenerationComparator`, but does so in a more robust way by using the command line interface to test the script's functionality.

```
SLEEP_INTERVAL = 0.25
```

```
TIMEOUT = 6
```

```
execute()
```

Launch `gcnd` and make it listen to a test server set up with `pygcn-serve`. Let run for 5 seconds before timing out and killing.

```
class llama.test.test_listeners.test_gcn.AbstractTestLlamaHandlers
    Bases: llama.test.classes.AbstractFileGenerationComparator
```

Test `LlamaHandlers` with example GCN Notices to make sure that they are parsed correctly. Implementations of this class must provide an `EVENTID` class variable specifying the `eventid` to use (the `rundir` will be a temporary directory and the data source will be in the test data directory).

```
STARTING_MANIFEST = ()
```

execute()

Test whether the LLAMA handler properly creates the expected event directory and output files by comparing what is created to the expected output.

property gcn_archive

A temporary directory for storing GCN archived files.

property handler

Get a LlamaHandlers instance for this test class.

property pipeline

A Pipeline instance defining the output files generated by this test. These are the output files that will be compared by AbstractFileGenerationComparator.compare to make sure that all outputs are as expected (and, as a sanity check, that no inputs have been mutated).

test()

Run this test with execute and then check whether outputs are as expected with compare.

class llama.test.test_listeners.test_gcn.GcnInitialMixin

Bases: [llama.test.test_listeners.test_gcn.AbstractTestLlamaHandlers](#)

Specifies that this tests for LVC *Initial* GCN Notices; the proper LvcGcnXml, SkymapInfo, and Advok files should be generated.

property pipeline

A Pipeline instance defining the output files generated by this test. These are the output files that will be compared by AbstractFileGenerationComparator.compare to make sure that all outputs are as expected (and, as a sanity check, that no inputs have been mutated).

class llama.test.test_listeners.test_gcn.TestGcndS190425z

Bases: [llama.test.classes.S190425zMixin](#), [llama.test.test_listeners.test_gcn.GcnInitialMixin](#)

Test the daemon on the GCN Initial notice that came out with the first BNS event of O3 on April 25, 2019.

class llama.test.test_listeners.test_gcn.TestGcndS190521g

Bases: [llama.test.classes.S190521gMixin](#), [llama.test.test_listeners.test_gcn.AbstractGcndTester](#)

Test the daemon on the GCN Preliminary notice that came out an early-O3 BBH event from May 21, 2019.

class llama.test.test_listeners.test_gcn.TestLlamaHandlerMS181101ab

Bases: [llama.test.classes.MS181101abMixin](#), [llama.test.test_listeners.test_gcn.GcnInitialMixin](#)

Test LlamaHandlers on a sample GCN Initial notice for ER14.

class llama.test.test_listeners.test_gcn.TestLlamaHandlerS190425z

Bases: [llama.test.classes.S190425zMixin](#), [llama.test.test_listeners.test_gcn.GcnInitialMixin](#)

Test LlamaHandlers on the GCN Initial notice that came out with the first BNS event of O3 on April 25, 2019

class llama.test.test_listeners.test_gcn.TestLlamaHandlerS190521g

Bases: [llama.test.classes.S190521gMixin](#), [llama.test.test_listeners.test_gcn.AbstractTestLlamaHandlers](#)

Test LlamaHandlers parser on the GCN Preliminary notice that came out with an early-O3 BBH event from May 21, 2019.

39.3 llama.test.classes module

Classes used in unit tests for LLAMA.

```
class llama.test.classes.AbstractFileGenerationChecker
Bases: llama.test.classes.AbstractFileGenerationComparator
```

Just like AbstractFileGenerationComparator, but does not compare file contents of outputs to a nominal value; instead, just checks whether the expected output files exist. Use this for files that do not depend deterministically on their inputs (and cannot therefore be compared with a simple diff to expected outputs).

compare()

Check whether running execute in the test eventdir has produced all of the expected output files. Does not check contents. Use this for files whose contents do not deterministically depend on input files.

Returns `events_equal` – True if the output is exactly as expected. False in any other case.

Return type bool

```
class llama.test.classes.AbstractFileGenerationComparator
```

Bases: abc.ABC

A class for running a test that produces some sort of expected output files for a given event and testing whether those output files are as expected by comparing them to reference files that are assumed correct. This class should be used to implement unit tests for the actual file generation steps of each step in the pipeline (including external triggering).

abstract property EVENTID

The eventid (i.e. the directory name) of the scenario that is being tested. Must correspond to an Event in TEST_RUN_INPUTS (i.e. EVENTID must be a directory in run.rundir for some run in TEST_RUN_INPUTS).

```
INPUT_RUN = Run(rundir="/Users/s/.local/share/llama/inputs/tests/event_files",
pipeline="Pipeline(<['Advok', 'CoincScatterI3LvcPdf', 'CoincScatterI3LvcPng',
'CoincScatterZtfI3LvcPdf', 'CoincScatterZtfI3LvcPng', 'CoincScatterZtfLVCPdf',
'CoincScatterZtfLVCPng', 'CoincSignificanceI3Lvc',
'CoincSignificanceSubthresholdI3Lvc', 'CoincSummaryI3LvcPdf',
'CoincSummaryI3LvcTex', 'FermiGRBsJSON', 'IceCubeNeutrinoList',
'IceCubeNeutrinoListCoincTxt', 'IceCubeNeutrinoListTex', 'IceCubeNeutrinoListTxt',
'LVAAlertAdvok', 'LVAAlertJSON', 'LVCGraceDbEventData', 'LvcDistancesJson',
'LvcGcnXml', 'LvcRetractionXml', 'LvcSkymapFits', 'LvcSkymapHdf5', 'PAstro',
'RctSlkI3CoincSummaryI3LvcPdf', 'RctSlkLmaCoincScatterI3LvcPdf',
'RctSlkLmaCoincScatterI3LvcPng', 'RctSlkLmaCoincScatterZtfI3LvcPdf',
'RctSlkLmaCoincScatterZtfI3LvcPng', 'RctSlkLmaCoincScatterZtfLVCPdf',
'RctSlkLmaCoincScatterZtfLVCPng', 'RctSlkLmaCoincSignificanceI3Lvc',
'RctSlkLmaCoincSignificanceSubthresholdI3Lvc', 'RctSlkLmaCoincSummaryI3LvcPdf',
'RctSlkLmaLVAAlertJSON', 'RctSlkLmaLVCGraceDbEventData', 'RctSlkLmaLvcDistancesJson',
'RctSlkLmaLvcGcnXml', 'RctSlkLmaLvcRetractionXml', 'RctSlkLmaSkymapInfo',
'SkymapInfo', 'ZtfTriggerList'])>")
```

abstract property STARTING_MANIFEST

A tuple of specific FileHandler classes whose files should be copied from the input directory in TEST_RUN_INPUTS to the temporary output test directory. This will set the starting state for the test.

compare()

Compare the outputs of execute in the test eventdir to the expected values in the input Event from TEST_RUN_INPUTS. Checks whether the file contents are strictly equal.

Returns `events_equal` – True if the output is exactly as expected. False in any other case.

Return type bool

property event
An Event instance corresponding to the output files of this test. Use this instance to get e.g. file locations if necessary.

abstract execute()
Run the actual test (no need to compare outputs; this will be done in `compare`).

property inputevent
An Event instance corresponding to the input files of this test.

abstract property pipeline
A Pipeline instance defining the output files generated by this test. These are the output files that will be compared by `AbstractFileGenerationComparator.compare` to make sure that all outputs are as expected (and, as a sanity check, that no inputs have been mutated).

provision()
Link or copy input files from `STARTING_MANIFEST` into `event.eventdir` so that they are available for `execute`. The test directory is presumed to exist.

property run
A temporary Run instance for tests (probably using a `pytest` fixture) where the test outputs for this test's `EVENTID` will reside. Feel free to override this with a custom directory.

test()
Run this test with `execute` and then check whether outputs are as expected with `compare`.

class llama.test.classes.MS181101abMixin
Bases: object
Tests on an ER14-era test event from the EM Follow-up userguide.
`EVENTID = 'MS181101ab-2-Initial'`

class llama.test.classes.S190412mMixin
Bases: object
Tests on S190412m, a manually-triggered BBH event in early O3.
`EVENTID = 'S190412m'`

class llama.test.classes.S190425zMixin
Bases: object
Tests on S190425z, the first BNS of O3, triggered by GCN Initial notice in this test.
`EVENTID = 'S190425z-1-Initial'`

class llama.test.classes.S190521gMixin
Bases: object
Tests on S190521g, a BBH event in O3, triggered by GCN Preliminary notice in this test.
`EVENTID = 'S190521g-1-Preliminary'`

39.4 llama.test.test_bin module

Test various LLAMA command line executables/scripts.

```
class llama.test.test_bin.AbstractTestS190412mSkymapInfoCli
    Bases: llama.test.classes.S190412mMixin, llama.test.test\_bin.AbstractTestSkymapInfoCli

    Test our ability to generate the correct skymap_info.json file through the skymap_info CLI for S190412m.
    Subclass this with specific flag combinations to test multiple cases.

    SKYMAP_FILENAME = 'bayestar.fits,0'

class llama.test.test_bin.AbstractTestSkymapInfoCli
    Bases: llama.test.classes.AbstractFileGenerationComparator

    Test the ability of the skymap_info command line interface to correctly register and save a new event. Specify
    different cli_args, FLAGS, and FLAGS_PRESET to test different cases (though note that only one of FLAGS or
    FLAGS_PRESET should be overridden with non-None values).

    FLAGS = None
    FLAGS_PRESET = None
    NON_DETERMINISTIC_JSON_FIELDS = ['notice_time_iso']
    SKYMAP_FILENAME = None
    STARTING_MANIFEST = ()
    property cli_args
        Specify the command-line arguments (besides --flags-preset, which is handled by FLAGS_PRESET,
        and --flags, which is handled by FLAGS; both of these will be prepended to cli_args) to pass to
        run_on_args when testing the skymap_info CLI. By default, the output directory, skymap_filename
        (if it is specified as something other than None in SKYMAP_FILENAME), and graceid are the only things
        specified.
    property cli_flags
        Get the flag-specification command-line arguments for this test by constructing them from either self.
        FLAGS or self.FLAGS_PRESET. Exactly one of these must be specified or else an assertion will fail.
    compare()
        We are comparing two JSON files, but some of the contents are non-deterministic; ignore those fields and
        compare the outputs as JSON dictionaries. Also checks that the saved flag values are as expected.
    execute()
        Run the skymap_info CLI to generate the expected output file.
    property pipeline
        A Pipeline instance defining the output files generated by this test. These are the output files that will
        be compared by AbstractFileGenerationComparator.compare to make sure that all outputs are as
        expected (and, as a sanity check, that no inputs have been mutated).
    property spec_flags
        Get a dictionary of the flags specified by either FLAGS or FLAGS_PRESET with which to check the output
        flag values. Exactly one of these must be specified or else an assertion will fail.

class llama.test.test_bin.TestS190412mSkymapInfoCliPublic
    Bases: llama.test.test\_bin.AbstractTestS190412mSkymapInfoCli

    AbstractTestS190412mSkymapInfoCli with TRIGGERED_PUBLIC flag preset.

    FLAGS_PRESET = 'TRIGGERED_PUBLIC'
```

```
class llama.test.test_bin.TestS190412mSkymapInfoCliTest
Bases: llama.test.test\_bin.AbstractTestS190412mSkymapInfoCli
AbstractTestS190412mSkymapInfoCli with TRIGGERED_PUBLIC flag preset.

FLAGS_PRESET = 'TRIGGERED_TEST'

llama.test.test_bin.test_llama_pipeline_parser()
Test the llama.pipeline.Parsers.pipeline parsers with various command line inputs. Make sure they
parse the expected Pipeline instances.

llama.test.test_bin.test_llama_run_parser()
Test the llama.run.Parsers.eventfiltering CLI parser, which interprets which directories a run should
operate on, by making sure it parses inputs as expected.

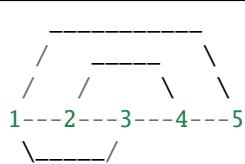
llama.test.test_bin.test_mjd_interval()
Make sure llama.files.i3.__main__.mjd_interval correctly parses input times making sure that a few
equivalent input argument combinations produce the same output.
```

39.5 `llama.test.test_filehandler` module

Test `FileHandler` and other related basic LLAMA classes found in `llama.filehandler`. Also provides useful methods for testing `FileHandler` subclasses in general.

```
class llama.test.test_filehandler.AbstractTestObsolescence
Bases: abc.ABC
```

Test our ability to mark files as obsolete in the expected ways (with the expected effects on file generation). These test are run with a mock Pipeline with topology (directed left-to-right):



This allows for tests of a few different scenarios in which file generation ordering can be tested to have the correct ordering. As a trivial example, changing 1 will obsolete all remaining `FileHandler` instances, but because they depend on one another in sequence, the regeneration order must be in increasing order of label value. This lets us test methods related to checking file obsolescence and reacting by updating output files.

```
EVENTID = 'obsolescence'
```

```
property event
```

The event for this test.

```
abstract main()
```

Run the main part of the test (after the temporary test directory has been set up and the files in the MockPipeline have been generated). This is where you should put obsolescence-related test logic.

```
property rundir
```

The rundir for this test (if it is set). Raises an `AttributeError` if none has yet been specified.

```
sorted_by_generation_time()
```

Get a list of the generated FileHandlers in temporal order of file generation.

```
test()
```

Run the `main` test after having set everything up in a temporary test directory.

```
class llama.test.test_filehandler.MockFh1(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler
```

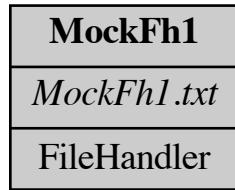


Fig. 1: `llama.test.test_filehandler.MockFh1` is created from external triggers. It therefore has no LLAMA-representable input dependencies, but instead acts as initial input for other `FileHandler` classes.

```
DEPENDENCIES = ()
FILENAME = 'MockFh1.txt'
MANIFEST_TYPES = (<class 'llama.test.test_filehandler.MockFh1'>,)
UR_DEPENDENCIES = ()
UR_DEPENDENCY_TREE = frozenset({})

class llama.test.test_filehandler.MockFh2(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler
```

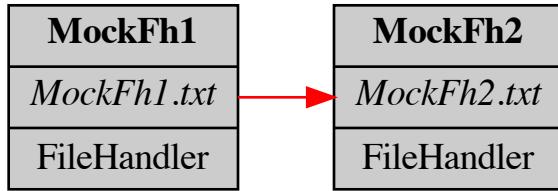


Fig. 2: Required input files for `llama.test.test_filehandler.MockFh2` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.test.test_filehandler.MockFh2` to be generated.

```
DEPENDENCIES = (<class 'llama.test.test_filehandler.MockFh1'>,)
FILENAME = 'MockFh2.txt'
MANIFEST_TYPES = (<class 'llama.test.test_filehandler.MockFh2'>,)
UR_DEPENDENCIES = (<class 'llama.test.test_filehandler.MockFh1'>,)
UR_DEPENDENCY_TREE = frozenset({<class 'llama.test.test_filehandler.MockFh1'>)})

class llama.test.test_filehandler.MockFh3(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler

DEPENDENCIES = (<class 'llama.test.test_filehandler.MockFh1'>, <class
'	llama.test.test_filehandler.MockFh2'>)

FILENAME = 'MockFh3.txt'
MANIFEST_TYPES = (<class 'llama.test.test_filehandler.MockFh3'>,)
UR_DEPENDENCIES = (<class 'llama.test.test_filehandler.MockFh1'>, <class
'	llama.test.test_filehandler.MockFh2'>)
```

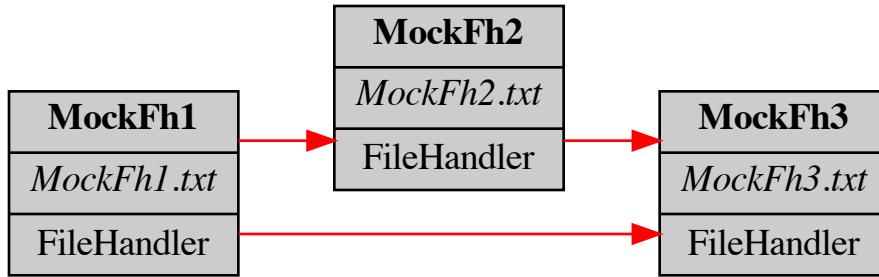


Fig. 3: Required input files for `llama.test.test_filehandler.MockFh3` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.test.test_filehandler.MockFh3` to be generated.

```
UR_DEPENDENCY_TREE = ImmutableDict({<class 'llama.test.test_filehandler.MockFh1':  
    ImmutableDict({}), <class 'llama.test.test_filehandler.MockFh2':  
    ImmutableDict({<class 'llama.test.test_filehandler.MockFh1':  ImmutableDict({})})})  
  
class llama.test.test_filehandler.MockFh4(eventid_or_fh, rundir=None)  
    Bases: llama.filehandler.FileHandler
```

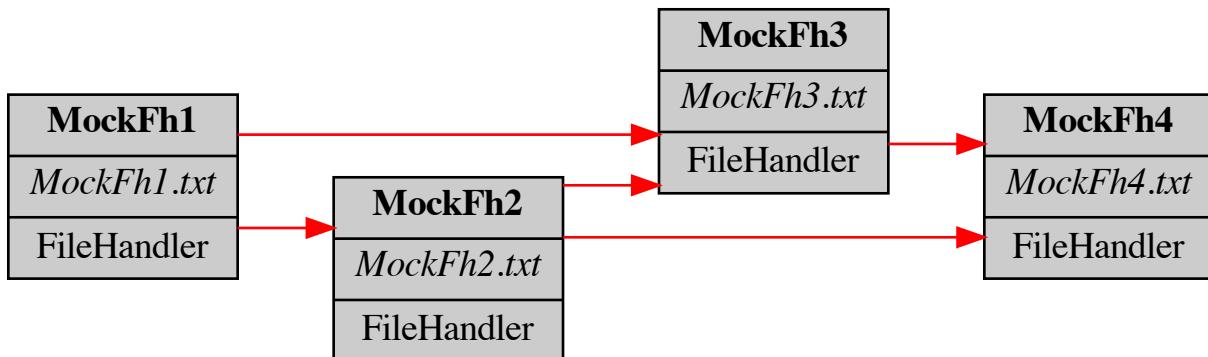


Fig. 4: Required input files for `llama.test.test_filehandler.MockFh4` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.test.test_filehandler.MockFh4` to be generated.

```
DEPENDENCIES = (<class 'llama.test.test_filehandler.MockFh2'>, <class  
'llama.test.test_filehandler.MockFh3'>)  
  
FILENAME = 'MockFh4.txt'  
  
MANIFEST_TYPES = (<class 'llama.test.test_filehandler.MockFh4'>,)  
  
UR_DEPENDENCIES = (<class 'llama.test.test_filehandler.MockFh1'>, <class  
'llama.test.test_filehandler.MockFh2'>, <class  
'llama.test.test_filehandler.MockFh3'>)  
  
UR_DEPENDENCY_TREE = ImmutableDict({<class 'llama.test.test_filehandler.MockFh3':  
    ImmutableDict({<class 'llama.test.test_filehandler.MockFh1':  ImmutableDict({}),  
        <class 'llama.test.test_filehandler.MockFh2':  ImmutableDict({<class  
            'llama.test.test_filehandler.MockFh1':  ImmutableDict({})})}), <class  
        'llama.test.test_filehandler.MockFh2':  ImmutableDict({<class  
            'llama.test.test_filehandler.MockFh1':  ImmutableDict({})})})
```

```
class llama.test.test_filehandler.MockFh5(eventid_or_fh, rundir=None)
Bases: llama.filehandler.FileHandler
```

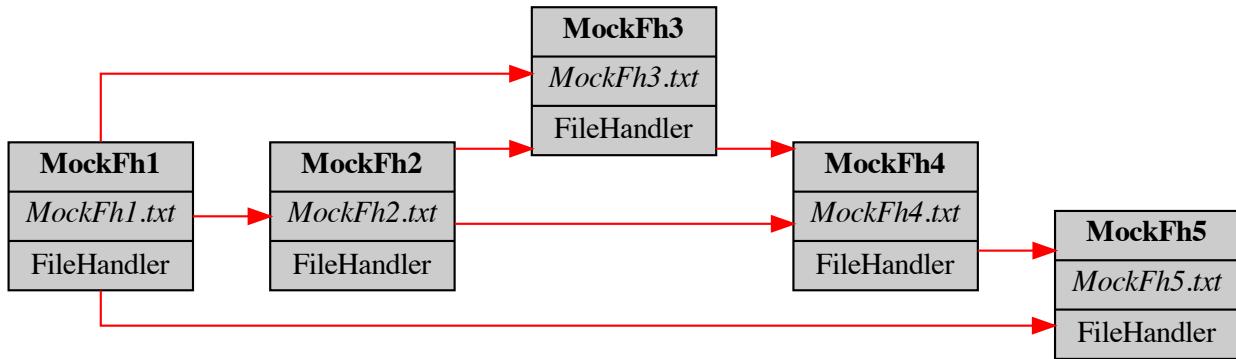


Fig. 5: Required input files for `llama.test.test_filehandler.MockFh5` (located on the far right of the graph). For a typical trigger, the leftmost files will be generated as triggers become available. They will be used as input for files to their right, eventually allowing `llama.test.test_filehandler.MockFh5` to be generated.

```
DEPENDENCIES = (<class 'llama.test.test_filehandler.MockFh1'>, <class 'llama.test.test_filehandler.MockFh4'>)

FILENAME = 'MockFh5.txt'

MANIFEST_TYPES = (<class 'llama.test.test_filehandler.MockFh5'>)

UR_DEPENDENCIES = (<class 'llama.test.test_filehandler.MockFh1'>, <class 'llama.test.test_filehandler.MockFh2'>, <class 'llama.test.test_filehandler.MockFh3'>, <class 'llama.test.test_filehandler.MockFh4'>)

UR_DEPENDENCY_TREE = ImmutableDict({<class 'llama.test.test_filehandler.MockFh1'>: ImmutableDict({}), <class 'llama.test.test_filehandler.MockFh4'>: ImmutableDict({<class 'llama.test.test_filehandler.MockFh3'>: ImmutableDict({<class 'llama.test.test_filehandler.MockFh1'>: ImmutableDict({}), <class 'llama.test.test_filehandler.MockFh2'>: ImmutableDict({<class 'llama.test.test_filehandler.MockFh1'>: ImmutableDict({})})}, <class 'llama.test.test_filehandler.MockFh2'>: ImmutableDict({<class 'llama.test.test_filehandler.MockFh1'>: ImmutableDict({})})}}))

class llama.test.test_filehandler.TestGenerationOrder
Bases: llama.test.test_filehandler.AbstractTestObsolescence

Make sure that, by obsoleting a file, we end up regenerating its descendants in the correct order.

main()
Make sure we regenerate files in the correct order.

class llama.test.test_filehandler.TestObsolescenceAndLocking
Bases: llama.test.test_filehandler.AbstractTestObsolescence

Test FileHandler's ability to mark files as obsolete, then lock those files, marking them as nonobsolete in perpetuity, then unlock them again, returning to the original state.

assertObsolete()
Check that everything else is obsolete after changing MockFh1.
```

main()

Run the main part of the test (after the temporary test directory has been set up and the files in the MockPipeline have been generated). This is where you should put obsolescence-related test logic.

`llama.test.test_filehandler.check_filehandler_definition_consistency(filehandler: llama.filehandler.FileHandler)`

Check whether a FileHandler has been consistently defined, raising an `AssertionError` if not.

`llama.test.test_filehandler.check_required_attributes(filehandler: llama.filehandler.FileHandler, err: bool = False)`

Return `False` if `filehandler` does not have all of its required class constants (in `filehandler.required_attributes()`) defined. If `err` is `True`, raise an assertion error instead of returning `False`. Return `True` if all required attributes are set (meaning that this FileHandler class is valid and ready to use).

`llama.test.test_filehandler.implemented_filehandler(item)`

Check whether an object is an implemented FileHandler (not abstract, all required attributes set).

`llama.test.test_filehandler.make_mock_filehandler(name, deps)`

Make a mock filehandler for simple mock pipeline tests. Give it the requested DEPENDENCIES as `deps`. Will simply write the current timestamp to file.

`llama.test.test_filehandler.test_filehandler_definition_consistency()`

Run `check_filehandler_definition_consistency` on all FileHandler classes that can be found defined in `llama` and its submodules that have their `required_attributes` set.

39.6 `llama.test.test_pipeline` module

Test for full `llama.pipeline` functionality by seeing whether past results pipeline output results are recomputed faithfully.

`llama.test.test_pipeline.test_default_pipeline_consistency()`

Check that all FileHandler classes in `DEFAULT_PIPELINE` are fully-implemented and self-consistent.

39.7 `llama.test.test_utils` module

Testing for utils.py

`llama.test.test_utils.memoize_func_helper(**memoize_args)`

Test `utils.memoize` for functions on both persistent and in-memory memoization (persistent memoization is available to functions only, not methods). Takes arguments to be passed to `utils.memoize`.

`llama.test.test_utils.memoize_helper(*returnvals)`

Creates a function that can only be called once to make sure that `memoize` doesn't unnecessarily get called many times.

Parameters `returnvals` (`iterable`) – The values that `called_once` should return (regardless of input args). If only one argument is provided, then that value will be returned. If multiple arguments are provided, they will be returned as a tuple. If no arguments are provided, then there will be no return value, i.e. `None` will be returned.

Returns `called_once` – A function that can only be called once (or else it raises an `AssertionError`).

Return type function

`llama.test.test_utils.test_get_grid_make_grid()`

Test `utils.get_grid`. Simply see if we can make a grid.

```
llama.test.test_utils.test_get_grid_north_pole()
    Test utils.get_grid. See if we can place a point at the north pole.

llama.test.test_utils.test_memoize()
    Test in-memory memoize on a function.

llama.test.test_utils.test_memoize_class()
    Test utils.memoize for classes.

llama.test.test_utils.test_memoize_helper()
    Test the memoize_helper. Returned function should raise an AssertionError when called more than once.

llama.test.test_utils.test_rotate_angs2vec()
    Test utils.rotate_angs2vec.

llama.test.test_utils.test_write_gzip()
    Test utils.write_gzip.
```


LLAMA.UTILS MODULE

Utility functions for LLAMA with *no dependencies* on other LLAMA tooling.

class `llama.utils.ColorFormatter(fmt=None, datefmt=None, style='%', validate=True)`

Bases: `logging.Formatter`

A formatter that colors log output for easier reading.

`COLORS = {'CRITICAL': '\x1b[91m', 'DEBUG': '\x1b[94m', 'ERROR': '\x1b[93m', 'INFO': '\x1b[92m', 'WARNING': '\x1b[95m'}`

format(record)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using `LogRecord.getMessage()`. If the formatting string uses the time (as determined by a call to `usesTime()`, `formatTime()` is called to format the event time. If there is exception information, it is formatted using `formatException()` and appended to the message.

exception `llama.utils.GenerationError`

Bases: `OSError`

An error meant to indicate that a given file could not be generated at the time its `generate()` method was called due to missing DEPENDENCIES (or some other benign failure mode). This error should be thrown when behavior is otherwise nominal as an indication that a particular file simply could not be generated at a particular moment in time.

class `llama.utils.RemoteFileCacher(url, localpath=None)`

Bases: `llama.utils.RemoteFileCacherTuple`

A simple loader for remotely-cached data accessible on public URLs. Use this to download cloud-based resources and locally cache them in `/Users/s/.cache/llama/objects`. Files will only be downloaded as-needed to save space and bandwidth.

Parameters

- **url** (`str`) – Specify the URL of the remote resource.
- **localpath** (`str, optional`) – The (optional) local path at which to cache this resource. By default, will just be `/Users/s/.cache/llama/objects/filename` where `filename` is actually taken from the remote URL filename.

get(query=None)

If the file is not available locally, download it and store it at `localpath` (do nothing if present). Return `localpath`. Optionally append a query string `query`.

class `llama.utils.RemoteFileCacherTuple(url, localpath)`

Bases: `tuple`

localpath

Alias for field number 1

url

Alias for field number 0

llama.utils.archive_figs(*plots*, *outfile*, *exts*=('pdf', 'png'), *fname_list*=None, *close_figs*=False)

Take a list of figures and save them with filenames that look like {*fname*}.{*suffix*} to a .tar.gz archive named *outfile*. The *list_index* will be the zero-padded index of the given plot in the *plots* list. Default filenames are generated if *fname_list* is not given. For instance, with no *fname_list* specified, the third plot in *plots* would have files named plt_003.pdf and plt_003.png saved to *outfile* (assuming default kwargs values).

Parameters

- **plots** (*list*) – A list of matplotlib figures to save to file.
- **outfile** (*str*) – Filename of the output archive. Should end in .tar.gz since this will be a zipped tarball of figures.
- **exts** (*list, optional*) – List of filename extensions to use. Each extension becomes the *suffix* in the above format string. Allows multiple image formats to be saved for each plot.
- **fname_list** (*list, optional*) – A list of file names to use for each figure (sans file extensions). *NOTE:* If not provided, some default filenames are generated instead. These follow the filename pattern plt_{list_index}.{suffix}.
- **close_figs** (*bool, optional*) – If True, clear the figures in *plots* after plotting is finished. This will help reduce memory usage if the plots are not going to be reused.

llama.utils.bytes2str(*bytes_or_str*)

Convert an object that could be a bytes object (in python3) to a str object. If it is already a str object, leave it alone. Used for python2/python3 compatibility.

llama.utils.color_logger(*loglevel*=None)

Return a function log that prints formatted strings to either stdout or with the logger using a certain log level (if *loglevel* is specified)

llama.utils.find_in_submodules(*modulename*: str, *predicate*: function)

Return a dict of fully-qualified variable names mapping to all objects in *modulename* and its submodules for which *predicate* is True.

llama.utils.getVOEventGPSNanoseconds(*voeventpath*)

Get the number of nanoseconds past the start of the current GPS second of the event described in this VOEvent file. For example, if the current GPS time is 123456789.033, this method will return 33000000.

llama.utils.getVOEventGPSSeconds(*voeventpath*)

Get the GPS time of the event described in this VOEvent file to the nearest second.

llama.utils.get_grid(*ra*, *dec*, *radius*, *pixels*=100, *degrees*=True)

Get a list of pixel positions arranged in a rectangular grid and filling a square with sizes of length *radius* centered at Right Ascension *ra* and Declination *dec* with *pixels* as the width and height in pixels of the square grid.

This technically only works properly if the radius is very small, so a ValueError will be raised for radii larger than MAX_GRID_SCALE_RAD. For larger sky areas, HEALPix pixelizations should be used to assure equal area.

Parameters

- **ra** (*float*) – The Right Ascension of the center of the grid.
- **dec** (*float*) – The Declination of the center of the grid.

- **radius** (*float*) – The radius of the circle. Must be larger than MAX_GRID_SCALE_RAD (in radians) to keep pixel areas nearly constant.
- **pixels** (*float, optional*) – The diameter, in pixels, of the circle along the cardinal directions of the grid. Higher values imply higher resolutions for the grid.
- **degrees** (*bool, optional*) – If true, interpret all input and returned angles as being measured in degrees. Otherwise, interpret them all as being measured in radians.

Returns

- **ras** (*array*) – Right Ascension values of the grid pixels in the specified units.
- **decs** (*array*) – Declination values of the grid pixels in the specified units.
- **area** (*float*) – the (approximate) solid-angle per-pixel in the specified units.

Raises ValueError – Raised when the `radius` is too large (larger than MAX_GRID_SCALE_RAD, see note above).

llama.utils.get_test_file(filename, eventid)

Get the path to an example file as would be generated by LLAMA. These are stored in a single directory (equivalent to an event directory) and can be used for unit, integration, and doc tests.

Parameters

- **filename** (*str*) – The name of the file as it would appear in an event directory with no path information before it.
- **eventid** (*str*) – The name of the event directory we will be using for this unit test. Since different test cases apply to different file handlers, you'll need to make sure to avoid specifying an `eventid` that this filehandler is not meant to be tested on (whatever the reason).

Returns `test_filepath` – The full path to the file to be used for testing.

Return type str

llama.utils.get_voevent_gps_time(voeventpath)

Get the GPS time of the event described in this VOEvent file. Maximum precision is in nanoseconds, though in practice it will not be this high.

llama.utils.get_voevent_notice_time(voeventpath)

Get a unicode string with the date and time of the notification rather than the event itself. Is in ISO format.

llama.utils.get_voevent_param(voeventpath, param_name)

Get a ‘What’ parameter from a VOEvent XML file and return it as a string (no clever type inference is performed). For example, get the FAR (False Alarm Rate) for event G298048 (first BNS detection), which is included as a standard test case in ‘llama/data/tests/voevents’:

Examples

```
>>> get_voevent_param(get_test_file('lvc_gcn.xml',
...                               'MS181101ab-2-Initial'), 'FAR')
'9.11069936486e-14'
```

Note that the type of the returned data will (probably) be a unicode string, so be sure to convert as necessary.

llama.utils.get_voevent_role(voeventpath)

Get the role of this VOEvent, test or observation.

llama.utils.get_voevent_time(voeventpath)

Get a unicode string with the ISO date and time of the event:

Examples

```
>>> get_voevent_time(get_test_file('lvc_gcn.xml',
...                                'MS181101ab-2-Initial'))
'2018-11-01T22:22:46.654437'
```

`llama.utils.gps2mjd(gps)`
Convert GPS seconds to MJD time

`llama.utils.gps2utc(gps)`
Convert GPS seconds to UTC string

`llama.utils.memoize(method=False)`
A decorator that takes a method and memoizes it by storing the value in a class or instance variable. Useful when the same arguments are likely to keep recurring and the computation time is long. Data is stored in the function (for regular functions and staticmethods), the class (for classmethods) or in the instance (for properties and methods), this approach can't work for staticmethods nor bare functions. For this to work, all arguments (except `self` or `cls`) must be hashable.

NOTE: all arguments must be hashable (with the exception of the class or instance when `method` is `True`; see below).

Parameters `method (bool, default=False)` – If true, treat this function like a method, so that the first argument is a class or instance, and store the cache with that class or instance instead of storing it with the function. This lets you use function memoization on properties, classmethods, and instance methods even if the class or instance is not hashable.

`llama.utils.mjd2gps(mjd)`
Convert MJD time to GPS seconds

`llama.utils.mjd2utc(mjd)`
Convert MJD time to UTC string

`llama.utils.parameter_factory(classname, description, **kwargs)`
Return a class that can be used as a namespace for a bunch of parameters related to a specific search. Very similar to defining a `namedtuple` except that each parameter is defined as a property with a docstring, making these objects suitable for interactive use without having to open the source code to see parameter descriptions. Parameter names are provided as keyword argument names (see below).

The main point of this factory is that it provides a compact way of defining a new hashable class with descriptive docstrings; these features are good for classes representing collections of parameters (hence the function name).

The reason to implement this as a factory function rather than an abstract superclass is that the returned class has static attributes with docstrings available in a way that ipython can parse, making for easier interactive use of the built-in documentation.

Parameters

- `classname (str)` – The name of the new class.
- `description (str)` – A description of what sort of analysis these parameters are to be used for. Becomes part of the docstring of the new class (along with the descriptions included in `parameters`).
- `kwargs (str or function)` – The names and descriptions (or, for derived values, formulas) of parameters to be stored. The names of the arguments become the parameter names; these will be attributes of the returned class. The values of the arguments can be strings serving as descriptions of the parameter (for INDEPENDENT parameters provided by the user, in which case the descriptive string becomes the docstring for the parameter in the returned class) or functions that take `self` as their only argument where `self` is an instance

of the new parameter class and some of the attributes of said instance are used to calculate the value of this (DEPENDENT) parameter; in these cases, the docstring of the function is reused for that parameter. These dependent parameters become properties of the new class and are calculated dynamically from the class's other values (possibly including other dependent parameters as long as there are no circular method calls).

Returns `new_class` – a class whose name is `classname`, whose properties are all the parameter names, whose docstrings are their descriptions, whose docstring is the class `description` combined with the names and descriptions in the `kwargs` parameters, and whose `__init__` signature requires that the parameters be passed as `kwargs` for the sake of explicitness in instantiation.

Return type `type`

Examples

```
>>> def buz(self):
...     "bar+baz"
...     return self.bar + self.baz
>>> def boz(self):
...     "double buz"
...     return 2*self.buz
>>> Foo = parameter_factory(
...     "Foo",
...     "A silly example set of parameters.",
...     bar='a fake param',
...     baz='another fake param',
...     buz=buz,
...     boz=boz,
... )
>>> Foo.bar.__doc__
'(independent)\na fake param'
>>> Foo.baz.__doc__
'(independent)\nanother fake param'
>>> Foo.buz.__doc__
'(dependent)\nbar+baz'
>>> Foo.boz.__doc__
'(dependent)\ndouble buz'
>>> quux = Foo(bar=1, baz=2)
>>> quux.bar
1
>>> quux.baz
2
>>> quux.buz
3
>>> quux.boz
6
```

llama.utils.plot_graphviz(`dot`, `outfile`)

Plot a .dot graph (graph here used in the mathematical sense) and save it to `outfile`. Calls `graphviz dot` executable, used for plotting .dot graphs, so `graphviz` must be installed on the system.

Parameters

- `dot` (`str`) – Graphviz .dot format graph (the same contents you would expect in a .dot file).

- **outfile** (*str*) – Output file to save graph to. File type is inferred from `outfile`'s extension. This file type must be one of the ones recognized in `GRAPH_EXTENSIONS`.

Raises `subprocess.CalledProcessError` – If the dot process fails to generate the output file. This exception object will have the nonzero return code in it.

llama.utils.rotate_angs2angs(*ra, dec, yrot, zrot, degrees=True*)

Take a list of angular sky positions (points on the sphere) and rotate them first through an angle `yrot` about the y-axis (right-handed) followed by an angle `zrot` about the z-axis. Return the new points as (ra, dec) coordinates of the rotated vectors.

Parameters

- **ra** (*array-like*) – A list or `numpy.ndarray` of Right Ascension values for each point on the sphere.
- **dec** (*array-like*) – A list or `numpy.ndarray` of Declination values for each point on the sphere. Must have same length as `ra`.
- **yrot** (*float*) – The angle through which to do a right-handed rotation about the positive y-axis. This rotation is applied first.
- **zrot** (*float*) – The angle through which to do a right-handed rotation about the positive z-axis. This rotation is applied second.
- **degrees** (*bool, optional*) – (DEFAULT: True) If true, interpret all input angles as being measured in degrees and return values in degrees. Otherwise, interpret them as being measured in radians and return values in radians.

Returns

- **outra** (*array*) – The Right Ascensions of the rotated input vectors in the specified units.
- **outdec** (*array*) – The Declinations of the rotated input vectors in the specified units.

Examples

```
>>> import numpy as np
>>> def eql(a, b, prec=1e-13, mod=360): # compares floating point arrays
...     a = np.array(a).flatten()
...     b = np.array(b).flatten()
...     return all(prec > (a%mod - b%mod))
>>> ra = range(360)
>>> dec = np.zeros(360)
>>> eql(rotate_angs2angs(ra, dec, 0, 0), [ra, dec])
True
```

llama.utils.rotate_angs2vec(*ra, dec, yrot, zrot, degrees=True*)

Take a list of angular sky positions (points on the sphere) and rotate them first through an angle `yrot` about the y-axis (right-handed) followed by an angle `zrot` about the z-axis. Return the new points as (x, y, z) coordinates of the rotated directions as vectors on the unit sphere.

Parameters

- **ra** (*array-like*) – A list or `numpy.ndarray` of Right Ascension values for each point on the sphere.
- **dec** (*array-like*) – A list or `numpy.ndarray` of Declination values for each point on the sphere. Must have same length as `ra`.

- **yrot** (*float*) – The angle through which to do a right-handed rotation about the positive y-axis. This rotation is applied first.
- **zrot** (*float*) – The angle through which to do a right-handed rotation about the positive z-axis. This rotation is applied second.
- **degrees** (*bool, optional*) – (DEFAULT: True) If true, interpret all input angles as being measured in degrees. Otherwise, interpret them as being measured in radians.

Returns

- **x** (*array*) – The x-coordinates of the rotated input vectors [cartesian].
- **y** (*array*) – The y-coordinates of the rotated input vectors [cartesian].
- **z** (*array*) – The z-coordinates of the rotated input vectors [cartesian].

Examples

```
>>> import numpy as np
>>> def eql(a, b, prec=1e-15): # compares floating point arrays
...     return all(prec > np.array(a).flatten() - np.array(b).flatten())
>>> eql(rotate_angs2vec(0, 0, 0, 0), [1, 0, 0])
True
>>> eql(rotate_angs2vec(270, 0, 0, 0), [0, -1, 0])
True
>>> eql(rotate_angs2vec(270, 0, 0, 90), [1, 0, 0])
True
>>> eql(rotate_angs2vec(180, 0, 90, 90), [0, 0, 1])
True
>>> eql(rotate_angs2vec(0, 90, 90, 90), [0, 1, 0])
True
>>> eql(rotate_angs2vec(np.pi/2, 0, 0, np.pi, degrees=False), [0, -1, 0])
True
>>> eql(rotate_angs2vec(np.pi, 0, np.pi/2, 0, degrees=False), [0, 0, 1])
True
```

`llama.utils.send_email(subject, recipients, body='', attachments=())`
Send an email using the default configured email address. No return value.

Parameters

- **recipients** (*list*) – A list of strings representing email addresses of the recipients.
- **body** (*str or file-like, optional*) – Send contents of this string or file-like object.
- **attachments** (*str or list, optional*) – A single filepath or a list of filepaths to files that should also be attached to this email.

`llama.utils.setup_logger(logfile, loglevel='DEBUG')`

Set up a logger for llama and all submodules that logs to both `logfile` and to `STDOUT`. You will still need to actually make a logger for whatever module you are calling this from.

Parameters

- **logfile** (*str*) – The logfile to write all output to. If the path equals or resolves to `/dev/null`, no logfile will be configured.

- **loglevel** (*int, str, or NoneType, optional*) – The loglevel to pass to `setLevel` for the `StreamHandler` writing to the terminal; the `logfile` handler always writes at maximum verbosity (DEBUG).

`llama.utils.sizeof_fmt(num, suffix='B')`

Format the size of a file in a human-readable way. `num` is the file size, presumed to be in bytes. Specify `suffix='b'` to indicate that the unit of `num` is bits.

`llama.utils.sort_files_by_modification_time(paths)`

Take the list of `path` strings sorted by ascending modification time. This is the last time the contents were modified, or, for a new file, its creation time.

`llama.utils.tbhighlight(tb)`

Syntax-highlight a traceback for a 256-color terminal.

Parameters `tb` (`str`) – The traceback string (as provided by `traceback.format_exc()`)

Returns `highlighted_tb` – The same traceback string with syntax highlighting applied.

Return type `str`

`llama.utils.utc2gps(utc)`

Convert UTC time string to GPS seconds

`llama.utils.utc2mjd(utc)`

Convert UTC time string to MJD time

`llama.utils.veccls(func)`

Like `vecstr` but for class arguments instead of string arguments.

`llama.utils.vecstr(func)`

Given a function taking a `FileHandler` instance `fh` and a `query` argument and returning a `bool`, return a function with the same signature that first checks whether `query` is a `str` before calling `func`. If `query` is a `str`, return `func(fh, query)`, and if it is not, return `any(func(fh, q) for q in query)`, i.e. check whether `func` is true for any of the values given in `query`. Note that this means `query` can either be a check against a `str` or against several possible values of `str` contained in an iterable. This function is an implementation detail used for downselection checks.

`llama.utils.vectypes(func, types)`

Implementation of `vecstr` and `veccls`.

`llama.utils.write_gzip(infile, outfilename)`

See whether a file is actually a valid gzip file. If not, zip it. Either way, write the result to `outfilename`.

Parameters

- **infile** (*file-like*) – File-like object to be compressed to a gzip file.
- **outfilename** (`str`) – Path to the output compressed file. Will be overwritten if it already exists.

Returns `success` – True if it was already a valid gzip file and `False` if we were forced to compress it.

Return type `bool`

Examples

Compress a string in a file-like BytesIO wrapper to some temporary .gz file:

```
>>> from io import BytesIO
>>> import tempfile
>>> with tempfile.NamedTemporaryFile(suffix='.gz', delete=False) as f:
...     outfilename = f.name
>>> infile = BytesIO('this is obviously not a valid gzip file'.encode())
>>> assert not write_gzip(infile, outfilename)
>>> with open(outfilename, 'rb') as actuallyzippedfile:
...     assert write_gzip(actuallyzippedfile, outfilename)
```

`llama.utils.write_to_zip(fname_list, write_functions, outfilename, suffix='')`

Create a new .tar.gz tarfile archive names `outfilename` and fill it with filenames given in `fname_list`. `write_functions` should be a list of functions that takes an input filename as its only argument and writes data to that filename. It should have the same length as `fname_list`, since each element of `write_functions` will be used to generate the corresponding file in `fname_list`. If the `write_functions` expect some sort of specific file extension, that can be provided by manually setting `suffix` (DEFAULT: '') to the appropriate extension. If each file requires its own specific filename extension, then `suffix` can be specified as a list of suffixes with length matching `fname_list` and `write_functions`.

Examples

```
>>> import tempfile, tarfile
>>> def foo(file):
...     with open(file, 'w') as f:
...         f.write('foo!')
>>> write_functions = [foo] * 3
>>> fname_list = ['bar.txt', 'baz.txt', 'quux.txt']
>>> outfile = tempfile.NamedTemporaryFile(delete=False)
>>> outfilename = outfile.name
>>> outfile.close()
>>> write_to_zip(fname_list, write_functions, outfilename, suffix='.txt')
>>> tar = tarfile.open(outfilename)
>>> [f.name for f in tar.getmembers()]
['bar.txt', 'baz.txt', 'quux.txt']
>>> os.remove(outfilename)
```

CHAPTER
FORTYONE

LLAMA.VERSION MODULE

CHAPTER
FORTYTWO

LLAMA.VERSIONING MODULE

Classes for versioning files using in a given directory. Currently implemented with git.

class llama.versioning.GitDirMixin

Bases: object

A mixin for EventTuple and FileHandlerTuple subclasses that allows you to manipulate their event directories through a `git` property returning a `GitHandler` pointing to that property.

property git

Get a `GitHandler` for manipulating the `eventdir` as a git repository. Used for versioning events.

class llama.versioning.GitHandler(eventdir)

Bases: `llama.versioning.GitHandlerTuple`

A class that performs `git` operations on an `eventdir`.

You can also call an instance as if it were a function to perform git commands conveniently; the interface is the same as `subprocess.Popen` with `cwd` set to the `GitHandler`'s `eventdir` (for convenience at the command line).

eventdir [str] The path to the directory that the new `GitHandler` instance will manipulate.

add(*files)

Run `git add` for all `files`. Raises a `GitRepoUninitialized` exception if not a git repository.

commit_changes(message)

`git add` all files in the `eventdir` and commit changes using `message` as the commit message. Raises a `GitRepoUninitialized` exception if not a git repository. This will FAIL with a `GenerationError` if there are no new changes.

copy_file(filename, outpath, commit_hash=None, serial_version=None)

Check out a copy of a file, optionally specifying a particular version of the file from this event's history, to the given `outpath`. If no version is specified with `commit_hash` or `serial_version`, the latest version will be copied.

Parameters

- **filename (str)** – The relative path to the file from `self.eventdir` (in most cases just the filename).
- **outpath (str)** – The path to the output file, or, if this path corresponds to an existing directory, the directory in which it should be saved (with the `os.path.basename` of ``filename``). **If the file exists, it will be overwritten without warning.**
- **commit_hash (str, optional)** – The commit hash, or partial commit hash containing the starting characters of the full hash (as long as enough characters are provided to disambiguate hashes), of the version of `filename` that is to be copied to `outpath`. **You can only specify one of commit_hash or serial_version.**

- **serial_version**(*int, optional*) – The serial_version (i.e. the numbered version) of filename to checkout. This is potentially more ambiguous than using commit_hash.
You can only specify one of commit_hash or serial_version.

Returns `outfile` – Path to the final output file.

Return type str

Raises

- **GitRepoUninitialized** – If the event directory is not a git directory.
- **ValueError** – If both commit_hash and serial_version are specified or if they do not correspond to available file versions.
- **IOError** – If the outpath cannot be written to.
- **FileNotFoundException** – If the file checkout fails.

property current_hash

Get the current git hash for this directory.

diff(*args)

Return the git diff for the given file paths (from their last commits) as a string. Raises a `GitRepoUninitialized` exception if not a git repository. This diff can be applied using `git apply`.

Parameters *`args`(str, optional) – File paths relative to the root of the git directory whose diffs should be taken. If no args are provided, the result will always be an empty string.

Returns `diff` – The exact text returned by `git diff ARG1 ARG2...` for the provided arguments.

An empty string is returned if none of the file contents of the given paths have changed since the last commit OR if no paths are specified (note that this differs from standard `git diff` behavior, where ALL diffs from the last commit are provided if no arguments are specified).

Return type str

property eventid

Parse an eventid from the eventdir by splitting off the basename.

filename_for_download(filename, last_hash=None)

Get a filename that includes the eventid, revision number, and version hash for `filename` (i.e. what version number this is in the version history; e.g. if three versions of *this* file exist in the version history, then this is version 3). If this `filename` does not appear in the git history, it will be marked ‘v0’ and the hash will be ‘UNVERSIONED’. The output format is eventid, version, first 7 digits of commit hash, and `filename`, split by hyphens, so that the third version of `skymap_info.json` for event S1234a with git hash dedb33f would be called S1234a-v3-dedb33f-skymap_info.json. Use this for file downloads or files sent to other services in order to facilitate data product tracking outside the highly-organized confines of a pipeline run directory.

hashes(*filenames, pretty='', last_hash=None)

Get a list of full commit hashes for all commits related to the provided filenames. Returns an empty list if no filenames are provided or if the filename is not being tracked by git.

Parameters

- **filenames**(list) – Relative paths from the eventdir whose commits should be retrieved. Returns an empty list if no filenames are specified. To match all paths in the commit history, specify ‘–’ as the only filename.
- **pretty**(str, optional) – The git format string specifying what to return for each commit. By default, only returns the git hash for each commit pertaining to the given `filenames`.

- **last_hash** (*str, optional*) – If specified, only return hashes up to and including this hash; does not return hashes appearing topologically later than this one. This can be a partial hash containing only the starting characters of the full hash (e.g. the first 7 characters, as is typically seen elsewhere) as long as enough characters are provided to disambiguate the available hashes.

Returns **hashes** – A list of git checksums for the commits related to the specified filenames (or some other per-commmit string whose contents are defined by `pretty`).

Return type list

Raises

- **GitRepoUninitialized** – If not a git repository.
- **ValueError** – If the command cannot be run with the given filenames in the given `eventdir`.
- **ValueError** – If the input `last_hash` is ambiguous (matches more than one hash) or if it matches no hashes.

init()

Initialize the `eventdir` as a git repository.

is_ancestor(*possible_ancestor_hash, commit_hash*)

Check whether `possible_ancestor_hash` is a topological ancestor of `commit_hash`. Returns True if the hashes refer to the same commit. Raises a `GitRepoUninitialized` exception if not a git repository. Useful for figuring out if one commit came after another (from a data flow perspective).

Returns **is_ancestor** – True if `possible_ancestor_hash` is an ancestor of `commit_hash`, False otherwise. NOTE that a value of False does not imply that `commit_hash` is an ancestor of `possible_ancestor_hash` (since they can be from different branches altogether).

Return type bool

is_clean()

Return whether there are any changes made to the `eventdir` since the last commit. Raises a `GitRepoUninitialized` exception if not a git repository.

is_repo()

Checks whether this event directory is a git repo by seeing if it contains a `.git` subdirectory. Raises a `GitRepoUninitialized` exception if not a git repository.

remove(*files)

Run `git rm` for all `files`. Raises a `GitRepoUninitialized` exception if not a git repository.

reset_hard(*ref*)

Hard reset the status of the branch to a given ref, losing all subsequent changes.

serial_version(*last_hash=None*)

The serial version of this file as stored in the version history. Note that this is merely a count of how many prior versions of the file exist in this history; it is not an unambiguous label (in the same way that the hash value is). Use this for human and interpretation. If the file does not exist, this function returns 0 (unversioned), so it effectively starts at 1.

show_log(*ref='HEAD'*)

Show the git commit message and notes for the given `ref`.

text_graph(*filenames, style='html')

Print a text graph of all files in the past history.

Parameters

- ***filenames** (*str, optional*) – An arbitrary list of filenames that will be spliced onto the end of the argument list for `git log`. Use this to narrow down the history shown. Use `--` to specify all files in the past history of the HEAD state.
- **style** (*str, optional*) – The format to put the output in. Options include ‘html’ (if this is going to go on a summary page).

exception `llama.versioning.GitRepoUninitialized`

Bases: `ValueError`

An exception indicating that a git repository has not been initialized.

CHAPTER
FORTYTHREE

LLAMA.VETOES MODULE

A mechanism for vetoing steps in an analysis.

`exception llama.vetoes.VetoException`

Bases: OSError

Indicates that a FileHandler's generation was vetoed and should proceed no further at this time. This is a benign exception that should be caught at file generation time.

`class llama.vetoes.VetoHandler(eventdir, manifest, vetoes)`

Bases: `llama.vetoes.VetoHandlerTuple, llama.flags.FlagsMixin`

A class that performs veto checks to see whether a FileHandler for a *specific* trigger should be vetoed. Instances can be called like functions to mark a FileHandler as vetoed.

`check()`

No return value. Raises a `VetoException` if this file has somehow been vetoed, which should be interpreted as meaning that file generation should be aborted.

First checks if `self.permanently_vetoed()` returns True (this is to be interpreted as meaning that the file should never be generated); if so, `VetoException` is raised.

Next, runs all veto functions in `self.vetoes` to see if these files have been vetoed in any way. If and when the first veto fails (returns True), its corresponding action (see the `vetoes` argument in `__init__`) is run, further vetoes are not checked, and `VetoException` is raised.

If these procedures are completed without a `VetoException` being raised, the method returns `None`.

`permanently_vetoed()`

Check whether the files in `self.manifest` should be generated based on whether their vetofiles exist (see `vetofilepaths` and `vetofilenames`); as long as this is the case, the files will be considered permanently vetoed and should not be generated.

`read_json()`

Read in the JSON dump of this vetofile. Returns `None` if the file is not vetoed.

`unveto()`

Remove *PERMANENT* veto status by deleting vetofiles for the file names specified in `self.manifest` and event directory specified by `eventdir`. This does not affect the checks contained in `self.vetoes` that are called by `self.check`, so file generation will still get vetoed if those automated checks fail. This method is more useful for removing permanent vetoes from files that were manually vetoed or vetoed under older pipeline versions.

`property vetofilenames`

A list of filenames whose existence indicates that these files should *never* be generated (at least for as long as these files exist). The contents of these files can optionally contain text descriptions of why the veto happened.

```
property vetofilepaths
    Absolute paths to vetofiles listed in self.vetofilenames (created by joining self.eventdir to those
    filenames).

class llama.vetoes.VetoHandlerTuple(eventdir, manifest, vetoes)
Bases: tuple

eventdir
    Alias for field number 0

manifest
    Alias for field number 1

veto
    Alias for field number 2

class llama.vetoes.VetoMixin
Bases: object

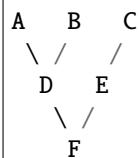
A feature for FileHandler-like objects (producing a manifest and an eventdir) that defines a list of vetoes in a
way that inherits additively from superclasses. Notably, provides a veto property returning a VetoHandler for
this instance. New vetoes can be added to classes by defining them in class_vetoes (vetoes from superclasses
class_vetoes attributes are dynamically included in the VetoHandler returned by veto. See VetoMixin.
veto docstring for details.

class_vetoes = ()

property veto
    Returns a VetoHandler instance for this FileHandler. See VetoHandler docstring for more info. Com-
    bines vetoes from all superclasses.

classmethod vetoes()
```

The list of veto checks and their corresponding actions called by `self.veto.checks()`. Returns the vetoes defined in `cls.class_vetoes`, where vetoes specific to this class are defined, along with the vetoes returned by `cls.class_vetoes` in all superclasses; for example, the order of vetoes pulled from `class_vetoes` (in a class and its superclasses) for a class F with inheritance tree



would be: F, D, A, B, E, C (with duplicates after the first instance of a veto check removed, where the ordering of superclasses is listed left-to-right in the above graph).

By default, no vetoes are applied to a base `FileHandler`. Vetoes must be added to `FileHandler` subclasses. If you really, truly need to get rid of vetoes from superclasses, consider not inheriting from those superclasses or factoring the `veto` functionality out of those superclasses as a mixin before overriding this property to only return a reduced `veto` list.

A description of the form of this data structure from the `VetoHandler.__init__` docstring:

An iterable of N length-2 iterables of the form:

`((veto1, action1), ..., (vetoN, actionN))`

matching `veto` functions to `action` functions.

`veto` functions are functions that take `eventdir` as arguments and return `True` if that particular veto criterion has been met (for example, a `veto` could return `True` if a trigger has somehow been marked as a test trigger).

`action` functions are callables that indicate some action that should be taken to handle the `veto`. `action` can also be `None` for the default behavior: to mark the set of filenames (specified by `manifest`) for this trigger as vetoed, which would prevent those files from being generated.

Alternatively, you can also use an actual function for `action` to do something more subtle than just vetoing the file (e.g. delaying file generation, advanced error logging, dummy testing, etc; this could even involve an alternative file generation method for test cases or edge cases). The `veto` functions will be run in order and the first to return `True` will have its `action` executed; the rest will be ignored.

NOTE that if you specify an action, the file will NOT be vetoed by default (since your veto-handling might involve something like delaying file generation or generating it through some other method, both of which are cases where you are not actually trying to prevent the file from being generated). If you want to fully veto file generation, you will have to manually call this `VetoHandler` instance at the end of your `action` to mark these files as PERMANENTLY vetoed and prevent file generation.

A `VetoException` WILL always be raised, however, since the called `action` is supposed to cancel any IMMEDIATE file generation by the default method (even if later attempts are still allowed).

`llama.vetoes.eventdir_path_contains_string_injection(eventdir)`

If the path to the trigger directory contains the substring ‘INJECTION’ (case-insensitive), this veto is triggered (returns `True`).

`llama.vetoes.eventdir_path_contains_string_manual(eventdir)`

If the path to the trigger directory contains the substring ‘MANUAL’ (case-insensitive), this veto is triggered (returns `True`).

`llama.vetoes.eventdir_path_contains_string_scratch(eventdir)`

If the path to the trigger directory contains the substring ‘SCRATCH’ (case-insensitive), this veto is triggered (returns `True`).

`llama.vetoes.eventdir_path_contains_string_test(eventdir)`

If the path to the trigger directory contains the substring ‘TEST’ (case-insensitive), this veto is triggered (returns `True`).

`llama.vetoes.utcnow()`

Return a new datetime representing UTC day and time.

PERFORMANCE PROFILING

This section shows a breakdown of the code executed in each test. Profiling results are organized by module and are auto-generated using [pytest-profiling](#)¹¹⁰ and [gprof2dot](#)¹¹¹.

44.1 Combined Results

Results of the full test suite.

44.2 Unit Tests

Unit tests apart from doctests.

44.3 Doctests

Results for doctests.

¹¹⁰ <https://pypi.org/project/pytest-profiling/>

¹¹¹ <https://github.com/jrfonseca/gprof2dot>

SOURCE CODE PLOTS

These plots show the lifetime of lines of code in the pipeline. They were generated using `git-of-theseus` and are here primarily for amusement. You can regenerate these plots in the source code directory by running `make theseus` and committing the updated plots.

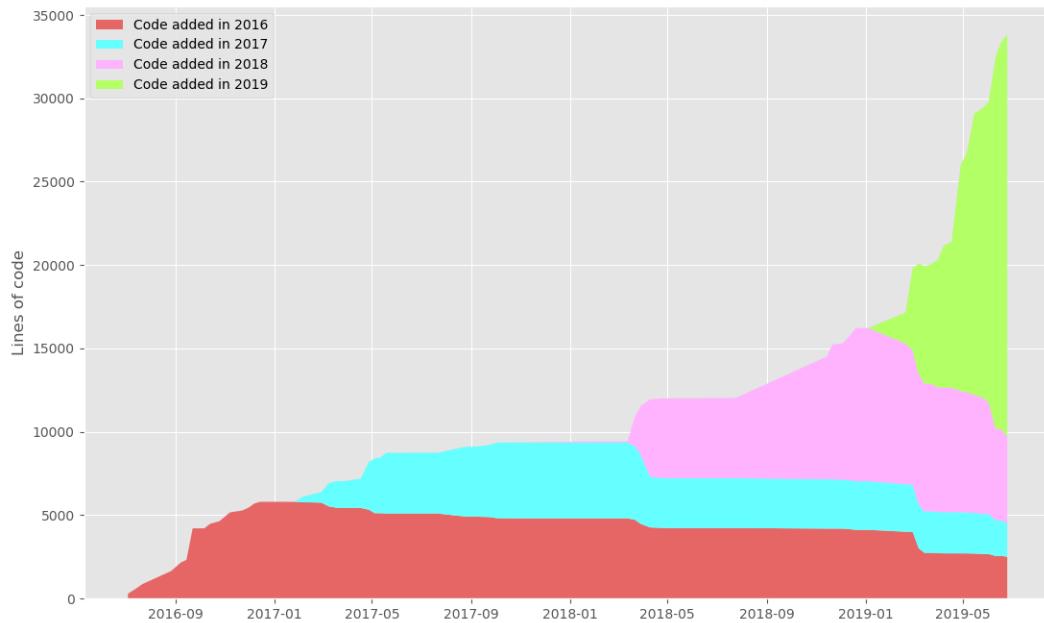


Fig. 1: History of python code in the repository over time, showing rates at which code has been added and replaced.

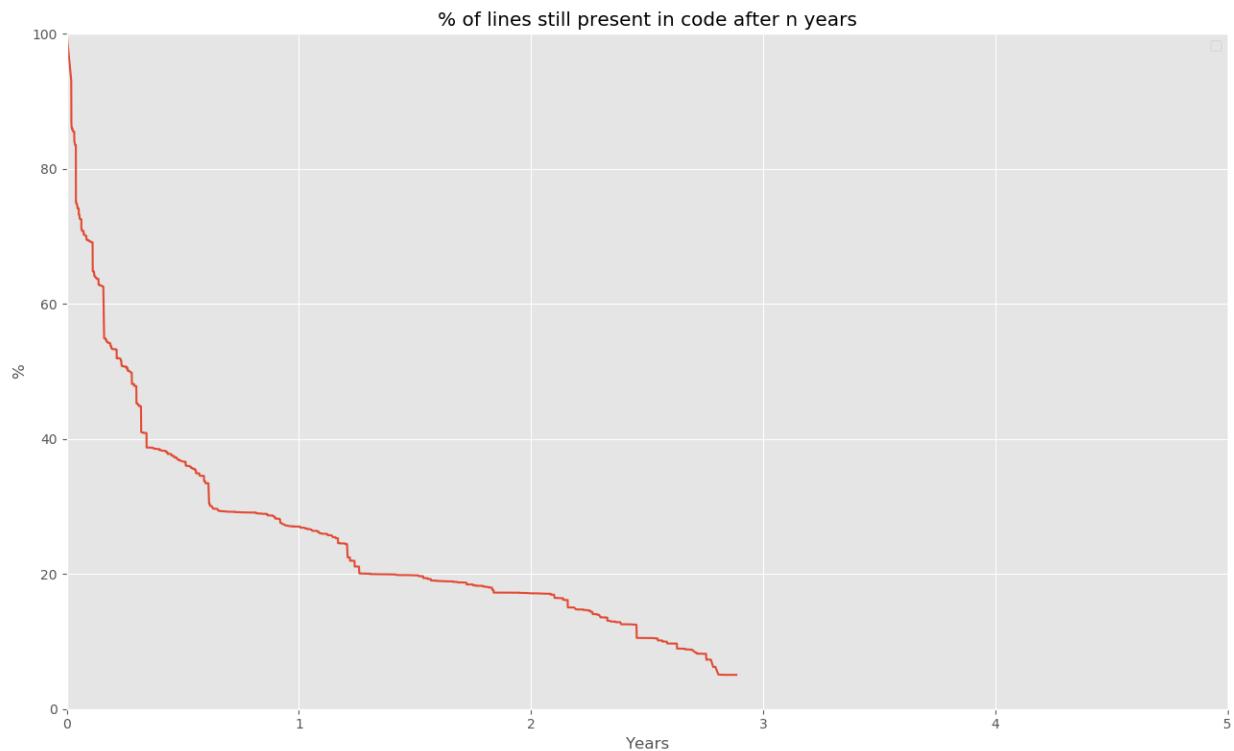


Fig. 2: Average lifetimes of lines of code in the repository.

LLAMA COMMAND LINE INTERFACE

This section covers utilities available at the command line through the `llama` command-line interface (CLI). All major commands are accessed recursively through `llama` and correspond to packages in the `llama` library with the same names: for example, `llama run` is the command you would use to start up the main LLAMA analysis daemon, and it corresponds to the functionality defined in the `llama.run` module. Read through the subcommands (including `llama run` as well as others exposing different functionality) to learn how to use LLAMA to its fullest from the command line.

A unified interface for accessing LLAMA functionality. Call the available sub-commands to see their capabilities; for example, to list the file-related commands available, run `llama files --help`; to see available event listeners, run `llama listen --help`, and to activate the GCN Notice listener, run `llama listen gcn`.

By default, all warnings about missing optional feature warnings (specified using `llama.classes.OptionalFeatureWarning`) are suppressed. If you'd like to see which features are missing, set `LLAMA_OPTIONAL_FEATURE_WARNINGS=true` when calling this script. This will inform you of which features are missing and which workarounds are active based on the limitations of your LLAMA installation, but for most everyday work, these warnings are excessive and useless.

Show warnings about optional LLAMA features that may not work on your current installation:

```
LLAMA_OPTIONAL_FEATURE_WARNINGS=true llama -h
```

```
usage: llama [-V] [-h]
              [{batch,com,dev,event,files,flags,install,listen,poll,run,serve}]
              [subargs [subargs ...]]
```

46.1 Named Arguments

-V, --version Print the version number and exit.

Default: False

-h, --help If a SUBCOMMAND is provided, run `--help` for that subcommand.
If no SUBCOMMAND is provided, print this documentation and exit.

Default: False

46.2 subcommands (call one with ```--help``` for details on each)

subcommand Possible choices: batch, com, dev, event, files, flags, install, listen, poll, run, serve

If a subcommand is provided, ALL remaining arguments will be passed to that command and it will be called.

subargs If SUBCOMMAND takes its own positional arguments, provide them here. This includes sub-sub commands.

subcommand summaries (pick one, use `--help` for more info):

batch Batch process large sets of organized data using LLAMA.

com Utilities for communicating with remote resources.

dev Developer tools.

event Display the current status of some event directory.

files Commands related to making LLAMA analysis files.

flags Set flags for an event directory from the command line.

install Run post-installation routines.

listen Listen for external triggers and react to them as they come in.

poll Scripts that continuously poll external resources for updates.

run The main interface for running the LLAMA pipeline.

serve Control LLAMA server processes (for example: interactive status website).

CHAPTER
FORTYSEVEN

LLAMA BATCH

Batch process large sets of organized data using LLAMA. Input and output files can be stored locally or on cloud data. Good for large-scale simulations of injected or randomized data. Use the command-line interface to specify where input data should be loaded from, where outputs should be stored, and how many events should be processed. These source/destination paths can be specified as python format strings, in which case their variables will be populated with variables sourced from a list of possible values (specified with --varlist) or from the environment (if no --varlist is specified for that variable name).

```
usage: llama batch [-h] [-V] [-D] [-f [FILEHANDLER [FILEHANDLER ...]]]
                   [+f [FILEHANDLER [FILEHANDLER ...]]]
                   [-p {DEFAULT_PIPELINE,SUBTHRESHOLD_PIPELINE,LLAMA2 REVIEW_PIPELINE}]
                   [--print-default-pipeline] [--dry-run-pipeline]
                   [--flags [FLAGNAME=value [FLAGNAME=value ...]])] [--flag-presets]
                   [--dry-run-flags] [-l LOGFILE]
                   [-v {debug,info,warning,error,critical,none}] [--get [GET [GET ...]]]
                   [--put [PUT [PUT ...]]] [--errdump [ERRDUMP [ERRDUMP ...]]]
                   [--erralert] [--format [FORMAT [FORMAT ...]])] [--eventdir EVENTDIR]
                   [--random N] [--public]
                   [params [params ...]]]
```

47.1 Named Arguments

-V, --version	Print the version number and exit. Default: False
-D, --dev-mode	If specified, allow the program to run even if the LLAMA version does not conform to semantic version naming. You should not do this during production except in an emergency. If the flag is not specified but local changes to the source code exist, <code>llama run</code> will complain and quit immediately (the default behavior). Default: False
--flags	A single flag preset (see: <code>llama.flags</code>) to use (print choices with <code>--flag-presets</code>) in <code>--outdir</code> OR individual flag settings in the format <code>FLAGNAME=value</code> . YOU SHOULD PROBABLY USE A PRESET rather than individual flag settings. Any omitted flags will take on the default values in <code>DEFAULT_FLAGS</code> . If you don't specify <code>--flags</code> ,

you'll be prompted to provide one; just provide `--flags` with no arguments to accept the default/existing flags. Flags are used to set overall behaviors for an event and set intentions, e.g. to mark an event as "ONLINE" and therefore allowed to communicate with partner web APIs and send out products and alerts. Flag name options and default values (for new events) are `FlagPreset({ 'BLINDED_NEUTRINOS': 'false', 'MANUAL': 'false', 'VETOED': 'false', 'ONLINE': 'true', 'ICECUBE_UPLOAD': 'false', 'UPLOAD': 'false', 'ROLE': 'test' })`; the full set of allowed values is ('true', 'false') for ONLINE, ('true', 'false') for BLINDED_NEUTRINOS, ('true', 'false') for MANUAL, ('true', 'false') for VETOED, ('true', 'false') for ICECUBE_UPLOAD, ('test', 'observation') for ROLE, and ('true', 'false') for UPLOAD.

<code>--flag-presets</code>	Print available flag presets.
<code>--dry-run-flags</code>	Print flags parsed by the CLI and exit. Default: False

47.2 choose pipeline (see ``llama.pipeline``)

<code>-f, --filehandlers</code>	Possible choices: LVAlertJSON, CoincSignificanceI3Lvc, CoincScatterI3LvcPdf, RctSlkLmaCoincSignificanceI3Lvc, LVCGraceDbEventData, ZtfTriggerList, IceCubeNeutrinoListCoincTxt, RctSlkLmaCoincScatterI3LvcPng, CoincSummaryI3LvcPdf, RctSlkLmaLvcGenXml, Advok, LvcSkymapHdf5, CoincScatterZtfLVCpng, IceCubeNeutrinoListTxt, RctSlkLmaCoincScatterZtfLVCpng, LvcSkymapFits, LvcGenXml, IceCubeNeutrinoList, SkymapInfo, CoincSummaryI3LvcTex, RctSlkLmaLVCGraceDbEventData, CoincScatterZtfI3LvcPdf, RctSlkLmaCoincScatterZtfLVCpdf, LvcDistancesJson, LVAlertAdvok, CoincScatterI3LvcPng, CoincScatterZtfLVCpdf, RctSlkLmaLVAlertJSON, RctSlkLmaLvcRetractionXml, LvcRetractionXml, RctSlkLmaCoincSummaryI3LvcPdf, RctSlkLmaCoincScatterZtfI3LvcPng, CoincSignificanceSubthresholdI3Lvc, RctSlkLmaCoincSignificanceSubthresholdI3Lvc, RctSlkI3CoincSummaryI3LvcPdf, RctSlkLmaLvcDistancesJson, IceCubeNeutrinoListTex, RctSlkLmaCoincScatterI3LvcPdf, RctSlkLmaCoincScatterZtfI3LvcPdf, PAstro, FermiGRBsJSON, RctSlkLmaSkymapInfo, CoincScatterZtfI3LvcPng
	A list of <code>FileHandler</code> class names which should be used. If provided, <code>FileHandler</code> classes whose names are in this list will be included in the pipeline. If the dependencies of a requested file are not available, no attempt will be made to generate them unless they too are listed explicitly. Available filehandlers are drawn from the <code>DEFAULT_PIPELINE</code> (print them with <code>--print-default-pipeline</code>).

<code>+f, ++filehandlers</code>	Possible choices: LVAlertJSON, CoincSignificanceI3Lvc, CoincScatterI3LvcPdf, RctSlkLmaCoincSignificanceI3Lvc, LVCGraceDbEventData, ZtfTriggerList, IceCubeNeutrinoListCoincTxt, RctSlkLmaCoincScatterI3LvcPng, CoincSummaryI3LvcPdf, RctSlkLmaLvcGenXml, Advok, LvcSkymapHdf5, CoincScatterZtfLVCpng, IceCubeNeutrinoListTxt, RctSlkLmaCoincScatterZtfLVCpng, LvcSkymapFits, LvcGenXml, IceCubeNeutrinoList, SkymapInfo, CoincSummaryI3LvcTex, RctSlkLmaLVCGraceDbEventData, CoincScatterZtfI3LvcPdf, RctSlkLmaCoincScatterZtfLVCpdf, LvcDistancesJson, LVAlertAdvok, CoincScatterI3LvcPng, CoincScatterZtfLVCpdf, RctSlkLmaLVAlertJSON, RctSlkLmaLvcRetractionXml, LvcRetractionXml,
---------------------------------	---

RctSlkLmaCoincSummaryI3LvcPdf, RctSlkLmaCoincScatterZtfI3LvcPng,
 CoincSignificanceSubthresholdI3Lvc, RctSlkLmaCoincSignificanceSubthresholdI3Lvc,
 RctSlkI3CoincSummaryI3LvcPdf, RctSlkLmaLvcDistancesJson,
 IceCubeNeutrinoListTex, RctSlkLmaCoincScatterI3LvcPdf, RctSlkLmaCoincScatterZtfI3LvcPng,
 PAstro, FermiGRBsJSON, RctSlkLmaSkymapInfo,
 CoincScatterZtfI3LvcPng

Exact same as --filehandlers, but all ancestors of the files listed with the + prefix will also be included. This means that, if you ask to generate a single file, an attempt will be made to **also generate everything it depends on** if necessary. If you want all those files made, this is a handy shortcut; if you want file generation to fail when ancestors are missing, use the - prefix instead.

-p, --pipeline Possible choices: DEFAULT_PIPELINE, SUBTHRESHOLD_PIPELINE, LLAMA2 REVIEW_PIPELINE

The name of the pipeline to use. Must be the name of a Pipeline instance from `llama.pipeline`. Available choices: ['DEFAULT_PIPELINE', 'SUBTHRESHOLD_PIPELINE', 'LLAMA2 REVIEW_PIPELINE']. If both this and `filehandlers` are specified, then the resulting pipeline will include all requested filehandlers from both options.

--print-default-pipeline Print the contents of the default pipeline and quit.

--dry-run-pipeline Print the pipeline selected by the user and quit without taking further action. Use this to make sure you've specified the correct pipeline.

Default: False

47.3 logging settings

-l, --logfile File where logs should be written. By default, all logging produced by `llama run` goes to both an archival logfile shared by all instances of the process as well as `STDERR`. The archival logfile can be overridden with this argument. If you specify `/dev/null` or a path that resolves to the same, logfile output will be suppressed automatically. Logs written to the logfile are always at maximum verbosity, i.e. `DEBUG`. (default: `/Users/s/.local/share/llama/logs/llama.batch.log`)

Default: “`/Users/s/.local/share/llama/logs/llama.batch.log`”

-v, --verbosity Possible choices: `debug`, `info`, `warning`, `error`, `critical`, `none`

Set the verbosity level at which to log to `STDOUT`; the `--logfile` will **ALWAYS** receive maximum verbosity logs (unless it is completely suppressed by writing to `/dev/null`). Available choices correspond to logging severity levels from the logging library, with the addition of `none` if you want to completely suppress logging to standard out. (default: `info`)

Default: “`info`”

47.4 simulation configuration

Remote file paths in the below arguments can be specified using regular URLs (by prefixing `http://` or `https://` before the path) or Amazon/DigitalOcean S3 paths by prefixing `s3://`{bucket}/ before the path (where {bucket} is replaced with the relevant bucket name``). Note that this will only work for unauthenticated HTTP/HTTPS endpoints, and the S3 downloads will only work if you have S3 configured (see: [llama.com.s3](#) documentation). Further note that *uploads* to HTTP/HTTPS endpoints are not supported.

params	Specify variable names and files that contain their possible values. Same syntax as bash variable assignments, but with the value set as the file containing possible values. The list of possible values can be a JSON or newline-delimited list. For example, if you have a list of values for the <code>graceid</code> located at <code>data/graceids.txt</code> and a list of values for <code>neutrino_filename</code> located in S3 bucket <code>llama</code> under the path <code>data/neutrino_filenames.json</code> , you would write <code>--params graceid=data/graceids.txt neutrino_filename=s3://llama/data/neutrino_filenames.json</code> . Note that the pipeline will run through the full Cartesian product of variable values from these lists; you can instead run through a pre-determined number of random vectors from this space (see <code>--random</code> below).
--get	Specify paths from which files can be loaded to be used as inputs for a simulation run. This looks a lot like the <code>--params</code> syntax, but with the variable names equal to the output filename and the option of including variables using python format string syntax. For example, to download <code>skymap_info.json</code> files from S3 directories with the GraceID as part of the path name, you might specify <code>--get skymap_info.json=s3://llama/sim/{graceid}/skymap_info.json</code> .
	Default: <code>OrderedDict()</code>
--put	Like <code>--get</code> , but instead you specify the <i>destination</i> to which an output file should be saved. This can be a local path or an S3 path.
	Default: <code>OrderedDict()</code>
--errdump	Specify directories in which to dump the contents of the active event directory in case of an unhandled exception. You can specify multiple places for this dump (e.g. upload to S3 and save to a local directory) in order to aid debugging. Git history is not copied; everything else is. This argument is ignored during normal operation.
	Default: <code>()</code>
--erralert	If specified, alert maintainers to every error via Slack.
	Default: <code>False</code>
--format	Specify filenames (as relative paths within the event directory) whose contents should be loaded and string formatted with values from <code>--params</code> and the environment. This operation happens after <code>--get</code> and replaces the contents of the file, allowing you to use boilerplate files with parametric values inserted.
--eventdir	Specify a local path in which each simulated event should be saved. Once again, you can use a format string to specify a path that includes the <code>--params</code> . For example, save events to directories based on their <code>graceid</code> and <code>neutrino_filename</code> with <code>--eventdir ~/sim/{graceid}-{neutrino_filename}/</code> . If this argument is omitted,

then only outputs specified using `--put` will be persisted, and the actual simulation directory will be discarded immediately after those files have been copied.

--random

If the `--random` flag is specified, then the pipeline will be run on random sequences of the variables specified in the `--params` files, and the simulation will run `N` times on randomized inputs from those files. Set `N` to a negative integer to run indefinitely. If `--random` is *not* specified, then the full Cartesian product of input variables from the `--params` will be used in sequence.

--public

If specified, files uploaded to S3 will be publicly available for download. Otherwise, those files will require S3 credentials for access.

Default: False

CHAPTER
FORTYEIGHT

LLAMA COM

Utilities for communicating with remote resources.

```
usage: llama com [-V] [-h] [{do,gracedb,slack}] [subargs [subargs ...]]
```

48.1 Named Arguments

-V, --version Print the version number and exit.

Default: False

-h, --help If a SUBCOMMAND is provided, run `--help` for that subcommand.
If no SUBCOMMAND is provided, print this documentation and exit.

Default: False

48.2 subcommands (call one with ```--help``` for details on each)

subcommand Possible choices: do, gracedb, slack

If a subcommand is provided, ALL remaining arguments will be passed to that command and it will be called.

subargs If SUBCOMMAND takes its own positional arguments, provide them here.
This includes sub-sub commands.

subcommand summaries (pick one, use `--help` for more info):

do Provision or destroy a bunch of digitalocean servers.

gracedb Download and install the GWHEN Kerberos keytab used for accessing GraceDB on production machines, or remove those credentials.

slack Send messages or upload files to Slack using LLAMA's built-in methods.

48.3 llama com do

Provision or destroy a bunch of digitalocean servers. User will be asked for confirmation before creation or destruction of droplets proceeds.

```
usage: llama com do [-h] [-c [DROPLET_NAME [DROPLET_NAME ...]]]
                     [-d [DROPLET_NAME [DROPLET_NAME ...]]] [-i IMAGE_NAME]
                     [-k [KEY_NAME [KEY_NAME ...]]] [-t [TAG_NAME [TAG_NAME ...]]] [-n]
                     [-C [COLUMN_NAME [COLUMN_NAME ...]]] [-S] [-K] [-T]
                     [-D [DROPLET_NAME [DROPLET_NAME ...]]]
```

48.3.1 Named Arguments

-c, --create	A list of names of snapshots to create. Will raise an error if any of these snapshots exists. Default: ()
-d, --destroy	A list of droplets to destroy. Can also provide a list of UNIX-style globs, like <i>llama-parallel</i> *, to match multiple droplets following a pattern. If no droplets are specified, instead provide a list of tags with the <i>-t</i> flag; any droplets with any of these tags will be destroyed. You will be prompted for confirmation before anything is deleted.
-i, --image	The name of the digitalocean image or user-created snapshot to use for newly-created droplets. Use <i>-list-snapshots</i> to see snapshot options. DEFAULT: ‘gwhen.com-post-er13’ Default: “gwhen.com-post-er13”
-k, --ssh-keys	A list of SSH keys to include in newly-created droplets. Use <i>--list-ssh-keys</i> to see choices. Specify the SSH keys by ID or by name. (default: all available keys)
-t, --tags	Tags to use. If destroying droplets and no droplets are specified, destroy ANY droplets with at least one of the provided tags. If creating droplets, apply these tags after droplet creation <i>if</i> tags with those names already exist (non-existing tags will be ignored). Default: ()
-n, --no-header-rows	If this flag is provided, tables will only print their data rows. Table titles and headers will be omitted. Useful for piping lists of droplet attributes to another program. Default: False
-C, --columns	Possible choices: name, ip, status, tags Which columns of Droplet metadata to include when printing droplets. Can be useful if you just want to print out a bunch of IP addresses, for example. DEFAULT: name, ip, status, tags; CHOICES: name, ip, status, tags Default: (‘name’, ‘ip’, ‘status’, ‘tags’)

-S, --list-snapshots	If provided, print available user-created snapshots and quit. Default: False
-K, --list-ssh-keys	If provided, print available SSH keys that can be used on new droplets and quit. Default: False
-T, --list-tags	List available droplet tags and quit. Default: False
-D, --list-droplets	If provided, print current user droplets (with their names, IP addresses, statuses, and tags) and quit. Optionally, provide a list of droplet names after this flag to only have information about those droplets printed. Those droplet names can also take shell-style wildcards, e.g. <i>llama-parallel-*</i> , to search for multiple droplets whose names fit a pattern (matching done with <i>fnmatch</i>).

48.4 `llama com gracedb`

Download and install the GWHEN Kerberos keytab used for accessing GraceDB on production machines, or remove those credentials. To fetch or install a LIGO robot keytab, you will need access to LLAMA S3 (see `llama.com.s3` for details on credentials) and will need to set `LLAMA_GRACEDB_AUTH` to the S3 key for a valid LIGO GraceDb robot keytab stored in S3. **If you are running on your personal computer, just use `kinit` `your.username@LIGO.ORG` followed by `ligo-proxy-init -k` to get access to GraceDB instead of using this script.**

```
usage: llama com gracedb [-h] [-l LOGFILE] [-v {debug,info,warning,error,critical,none}] {fetch,install,rm}
```

48.4.1 Positional Arguments

subcmd	Possible choices: fetch, install, rm If <code>fetch</code> is specified, download the robot keytab from LLAMA S3 if it is not currently installed (this requires you to have LLAMA S3 credentials, which you don't need unless you are a developer or are putting this machine into production use; if you're just using LLAMA, use <code>kinit</code> instead of this keytab). If <code>install</code> is specified, fetch the keytab if missing and generate Kerberos credentials with it, installing them to <code>/Users/s/.local/share/llama/cilogon_cert</code> . If <code>rm</code> is specified, run <code>kdestroy</code> to deactivate those credentials and delete <code>/Users/s/.local/share/llama/cilogon_cert</code> . Also remove the keytab file from local storage. (default: <code>install</code>) Default: "install"
---------------	---

48.4.2 logging settings

-l, --logfile	File where logs should be written. By default, all logging produced by <code>llama run</code> goes to both an archival logfile shared by all instances of the process as well as STDERR. The archival logfile can be overridden with this argument. If you specify <code>/dev/null</code> or a path that resolves to the same, logfile output will be suppressed automatically. Logs written to the logfile are always at maximum verbosity, i.e. DEBUG. (default: <code>/dev/null</code>)
	Default: <code>"/dev/null"</code>
-v, --verbosity	Possible choices: debug, info, warning, error, critical, none
	Set the verbosity level at which to log to STDOUT; the <code>--logfile</code> will ALWAYS receive maximum verbosity logs (unless it is completely suppressed by writing to <code>/dev/null</code>). Available choices correspond to logging severity levels from the logging library, with the addition of <code>none</code> if you want to completely suppress logging to standard out. (default: <code>info</code>)
	Default: <code>"info"</code>

48.5 llama com slack

Send messages or upload files to Slack using LLAMA's built-in methods.

```
usage: llama com slack [-h] [-l LOGFILE] [-v {debug,info,warning,error,critical,none}]
                        [-o {LLAMA,IceCube}] [-m MESSAGE] [-u USERS [USERS ...]]
                        [-f FILE] [--print-users] [--print-channels]
                        [channel [channel ...]]
```

48.5.1 Positional Arguments

channel	Which Slack channel to upload to.
----------------	-----------------------------------

48.5.2 Named Arguments

-o, --organization	Possible choices: LLAMA, IceCube Which organization to send a message to. (default: LLAMA) Default: "LLAMA"
-m, --message	Text content to send. Can be formatted using Slack's markdown-esque syntax. If not specified, read message from a pipe or send with no message if no pipe is being used.
-u, --users	Names of users to tag. You can use real names, display names, or Slack user IDs. They will be alerted to your message if they have access to the channel you are posting to. Default: []

-f, --file	Path to a file to upload as part of the message.
--print-users	Print a list of taggable users for the specified organization and then exit. Default: False
--print-channels	Print a list of channels that can be posted to for the specified organization and then exit. Default: False

48.5.3 logging settings

-l, --logfile	File where logs should be written. By default, all logging produced by <code>llama run</code> goes to both an archival logfile shared by all instances of the process as well as STDERR. The archival logfile can be overridden with this argument. If you specify <code>/dev/null</code> or a path that resolves to the same, logfile output will be suppressed automatically. Logs written to the logfile are always at maximum verbosity, i.e. DEBUG. (default: <code>/dev/null</code>) Default: “ <code>/dev/null</code> ”
-v, --verbosity	Possible choices: debug, info, warning, error, critical, none Set the verbosity level at which to log to STDOUT; the <code>--logfile</code> will ALWAYS receive maximum verbosity logs (unless it is completely suppressed by writing to <code>/dev/null</code>). Available choices correspond to logging severity levels from the logging library, with the addition of <code>none</code> if you want to completely suppress logging to standard out. (default: <code>info</code>) Default: “ <code>info</code> ”

LLAMA DEV

Developer tools.

```
usage: llama dev [-V] [-h]
                  [{background,clean,data,docs,dv,log,upload}] [subargs [subargs ...]]
```

49.1 Named Arguments

-V, --version Print the version number and exit.

Default: False

-h, --help If a SUBCOMMAND is provided, run `--help` for that subcommand. If no SUBCOMMAND is provided, print this documentation and exit.

Default: False

49.2 subcommands (call one with ```--help``` for details on each)

subcommand Possible choices: background, clean, data, docs, dv, log, upload

If a subcommand is provided, ALL remaining arguments will be passed to that command and it will be called.

subargs If SUBCOMMAND takes its own positional arguments, provide them here. This includes sub-sub commands.

subcommand summaries (pick one, use `--help` for more info):

background Tools for doing background sensitivity studies.

clean Clean up the LLAMA output directory, rotating out-of-date auxilliary files out of the way and archiving them on remote storage and setting event flag values for matching events to the values specified in `--flags`.

data `llama` data tools are used to securely fetch, move, and store privileged data.

docs Tools for automating documentation workflows.

dv Execute actions across a bunch of DigitalOcean servers based on their names.

log Tools for parsing log files to extract logged information.

upload Tools for uploading manifests of data files to a DigitalOcean Spaces/Amazon S3 object storage solution (for later user installation using `llama install`).

49.3 `llama dev background`

Tools for doing background sensitivity studies.

```
usage: llama dev background [-V] [-h]
                             [{pvalue,table,table_singles}] [subargs [subargs ...]]
```

49.3.1 Named Arguments

-V, --version	Print the version number and exit. Default: False
-h, --help	If a SUBCOMMAND is provided, run <code>--help</code> for that subcommand. If no SUBCOMMAND is provided, print this documentation and exit. Default: False

49.3.2 subcommands (call one with ```--help``` for details on each)

subcommand	Possible choices: pvalue, table, table_singles If a subcommand is provided, ALL remaining arguments will be passed to that command and it will be called.
subargs	If SUBCOMMAND takes its own positional arguments, provide them here. This includes sub-sub commands.

subcommand summaries (pick one, use `--help` for more info):

pvalue Take the output files made by `llama dev background table` for each population you are interested in and collate them into a single HDF5 file for p-value calculations.

table Make a table with the coincident significance values as rows (one EVENT per row).

table_singles Make a table with the coincident significance values as rows (one NEUTRINO per row).

49.3.3 `llama dev background pvalue`

Take the output files made by `llama dev background table` for each population you are interested in and collate them into a single HDF5 file for p-value calculations. All files are expected to be in the current directory, and the output will be in this directory as well.

Both input and output files correspond to background significance values for various source populations. The HDF5 file simply contains a dictionary of ascending-order arrays whose values are the test statistics of background simulations and whose keys are the name of the background population in a specific hierarchical order. These keys are automatically selected when `populations-hdf5-collater` is run in this directory by splitting the source `.txt` table filenames on hyphens

(-). For example, if the populations folder containg *bns-design.txt* *bns-o2.txt*, the output HDF5 populations file will have the following dictionary in it:

```
{
  "bns": {
    "design": array([...]),
    "o2": array([...])
  }
}
```

The output HDF5 file is saved to the current directory by default, but if you want to use it for LLAMA p-value calculations, make sure to upload both the tarballed populations directory and the compressed HDF5 file to cloud storage using `llama dev upload` and then store the new URLs in the `RemoteFileCacher` instances used for `llama.files.coinc_significance.NEUTRINO_POPULATIONS` and `llama.files.coinc_significance.NEUTRINO_POPULATIONS_TARBALL`.

To summarize: you want to download the existing population tarball, decompress it with `tar -xzf populations.tar.gz` (assuming it's named `populations.tar.gz`, of course), move your `llama dev` background table output text files to that directory, name the new tables in the expected format of `{SEARCH_TYPE}-{POPULATION}-{LVCRUN}-{GWDETECTORS}.txt` (e.g. `opa-bbh-o3-H1V1.txt`), run this script in that directory, put the text tables in a compressed tarball called e.g. `populations.tar.gz` by running `tar -czf populations.tar.gz populations` in the containing directory, upload both that and the HDF5 file to the cloud with `llama dev upload`, and then use the URLs in the manifest printed by `llama dev upload` to update the values of `NEUTRINO_POPULATIONS` and `NEUTRINO_POPULATIONS_TARBALL` in the `llama.files.coinc_significance` source code.

```
usage: llama dev background pvalue [-h] [-c]
```

49.3.3.1 Named Arguments

-c, --clobber	Overwrite “./populations.hdf5” if it already exists. Will not overwrite by default.
----------------------	---

Default: False

49.3.4 `llama dev background table`

Make a table with the coincident significance values as rows (one EVENT per row). Works on doublet neutrinos too (as long as the directory structure is as expected).

```
usage: llama dev background table [-h] [-i INPUT_PATTERN] [-s] [outfile]
```

49.3.4.1 Positional Arguments

outfile	Name of the output significance table. DEFAULT: inc_significance_outputs.txt	co-
	Default: “coinc_significance_outputs.txt”	

49.3.4.2 Named Arguments

-i, --input-pattern	UNIX-style glob pointing to the significance output files. //*/significance_lvc-i3.json	DEFAULT: //*/significance_lvc-i3.json
	Default: “//*/significance_lvc-i3.json”	
-s, --skymap-info	If specified, get information about the skymap used from skymap_info. This is supported for old versions of the analy- sis runs. If not specified, use the relative filepath to the significance file (default for newer runs).	
	Default: False	

49.4 llama dev clean

Clean up the LLAMA output directory, rotating out-of-date auxilliary files out of the way and archiving them on remote storage and setting event flag values for matching events to the values specified in **--flags**. You will need to specify a pattern for matching events that need to be cleaned using the **run** positional argument and the **--downselect** flag (see **llama run -h** help documentation for details); for example, you can match events created more than 86400 seconds ago with **--downselect sec_since_v0_gt=86400** to clean up test events older than a day.

```
usage: llama dev clean [-h] [--dry-run-dirs] [--downselect DOWNSELECT]
                       [--print-downselections]
                       [--flags [FLAGNAME=value [FLAGNAME=value ...]])] [--flag-presets]
                       [--dry-run-flags] [-l LOGFILE]
                       [-v {debug,info,warning,error,critical,none}] [--repeat REPEAT]
                       [--dry-run] [--testevents] [--test-archive-dir TEST_ARCHIVE_DIR]
                       [run]
```

49.4.1 Named Arguments

--flags	A single flag preset (see: llama.flags) to use (print choices with --flag-presets) in --outdir OR individual flag settings in the format FLAGNAME=value. YOU SHOULD PROBABLY USE A PRE- SET rather than individual flag settings. Any omitted flags will take on the default values in DEFAULT_FLAGS. If you don't specify --flags, you'll be prompted to provide one; just provide --flags with no ar- guments to accept the default/existing flags. Flags are used to set over- all behaviors for an event and set intentions, e.g. to mark an event as
----------------	--

“ONLINE” and therefore allowed to communicate with partner web APIs and send out products and alerts. Flag name options and default values (for new events) are FlagPreset({ ‘BLINDED_NEUTRINOS’: ‘false’, ‘MANUAL’: ‘false’, ‘VETOED’: ‘false’, ‘ONLINE’: ‘true’, ‘ICECUBE_UPLOAD’: ‘false’, ‘UPLOAD’: ‘false’, ‘ROLE’: ‘test’ }); the full set of allowed values is (‘true’, ‘false’) for ONLINE, (‘true’, ‘false’) for BLINDED_NEUTRINOS, (‘true’, ‘false’) for MANUAL, (‘true’, ‘false’) for VETOED, (‘true’, ‘false’) for ICECUBE_UPLOAD, (‘test’, ‘observation’) for ROLE, and (‘true’, ‘false’) for UPLOAD.

--flag-presets	Print available flag presets.
--dry-run-flags	Print flags parsed by the CLI and exit. Default: False
--repeat	If provided, run again every REPEAT seconds.
--dry-run	If provided, don’t actually do anything; just log what would be done. Default: False
--testevents	Clean up matching event files created by GCN listeners, LVAAlert listeners, and the like and move them to --test-archive-dir. Default: False
--test-archive-dir	The destination directory for archived events if --testevents is specified. (default: /Users/s/.local/share/llama/test_event_archive) Default: “/Users/s/.local/share/llama/test_event_archive”

49.4.2 filter runs and events (see: ``llama.run``)

run	A pattern specifying a list of directories to update of the form /run/directory/event*glob. See end of llama run -h documentation for details. (default: /Users/s/.local/share/llama/current_run/*)
--dry-run-dirs	Print the runs and event directories that would be affected and exit without taking further action. Default: False
--downselect	Arguments to pass to the downselect method of runs selected by the run argument (note that eventid_filter is already implicitly set by the glob pattern specified in run). Arguments will only be parsed as booleans (if they equal “True” or “False”), ints (if they can be parsed as such), floats (if they can be parsed as such), or strings and should be separated by commas, e.g. --downselect manual=False, fnameexists=PAstro. Omit a list of downselections or provide an empty string to specify no further downselections beyond the one implied by the run argument. (default: None)
--print-downselections	Print available downselections.

49.4.3 logging settings

-l, --logfile	File where logs should be written. By default, all logging produced by <code>llama run</code> goes to both an archival logfile shared by all instances of the process as well as STDERR. The archival logfile can be overridden with this argument. If you specify <code>/dev/null</code> or a path that resolves to the same, logfile output will be suppressed automatically. Logs written to the logfile are always at maximum verbosity, i.e. DEBUG. (default: <code>/Users/s/.local/share/llama/logs/llama.dev.clean.log</code>)
	Default: <code>"/Users/s/.local/share/llama/logs/llama.dev.clean.log"</code>
-v, --verbosity	Possible choices: debug, info, warning, error, critical, none Set the verbosity level at which to log to STDOUT; the <code>--logfile</code> will ALWAYS receive maximum verbosity logs (unless it is completely suppressed by writing to <code>/dev/null</code>). Available choices correspond to logging severity levels from the logging library, with the addition of <code>none</code> if you want to completely suppress logging to standard out. (default: info) Default: "info"

49.5 `llama dev data`

`llama` data tools are used to securely fetch, move, and store privileged data. Use these tools to cache private detector data in a safe way on LLAMA servers. You will not be able to use these scripts unless you have your authentication credentials for relevant services set properly.

```
usage: llama dev data [-V] [-h] [{i3}] [subargs [subargs ...]]
```

49.5.1 Named Arguments

-V, --version	Print the version number and exit. Default: False
-h, --help	If a SUBCOMMAND is provided, run <code>--help</code> for that subcommand. If no SUBCOMMAND is provided, print this documentation and exit. Default: False

49.5.2 subcommands (call one with ```--help``` for details on each)

subcommand	Possible choices: i3 If a subcommand is provided, ALL remaining arguments will be passed to that command and it will be called.
subargs	If SUBCOMMAND takes its own positional arguments, provide them here. This includes sub-sub commands.

subcommand summaries (pick one, use --help for more info):

- i3 Use `llama dev data i3` to fetch archival neutrino data and store it securely on LLAMA's AWS S3/DigitalOcean Spaces storage.

49.5.3 `llama dev data i3`

Use `llama dev data i3` to fetch archival neutrino data and store it securely on LLAMA's AWS S3/DigitalOcean Spaces storage. You can use this script to see which archival neutrino releases are available from IceCube servers and to transfer that data to private S3 storage. The printed code can be added to the LLAMA codebase under `llama.files.s3.json.ARCHIVAL_NEUTRINOS`; it will allow you to access the archival neutrino data stored on LLAMA servers *if* you have proper LLAMA S3 authentication credentials on your computer (allowing for proper reproducibility and faster data access). That data is only downloaded from LLAMA S3 when it is needed and is persisted in LLAMA's cache directory (see `llama.utils.CACHEDIR`).

```
usage: llama dev data i3 [-h] [-l LOGFILE] [-v {debug,info,warning,error,critical,none}]
                           [-a] [-s ARCHIVE_VERSION] [-r ARCHIVE_ROOT]
```

49.5.3.1 Named Arguments

-a, --available	Fetch available IceCube neutrino archive versions from the IceCube servers, print them one-per-line, and exit. Default: False
-s, --store-s3	Fetch IceCube neutrino archives specified by the <code>ARCHIVE_VERSION</code> , upload them to LLAMA S3 storage, and print a block of code that can be inserted into the <code>llama.files.s3.json.ARCHIVAL_NEUTRINOS</code> declaration to make these newly-stored archival neutrinos available to pipeline users with LLAMA S3 credentials (see <code>llama.com.s3</code>).
-r, --archive-root	The root directory on IceCube's cobalt servers in which to search for archival neutrinos (using <code>--store-s3</code> or <code>--available</code>). Default: “/data/ana/analyses/gfu”

49.5.3.2 logging settings

-l, --logfile	File where logs should be written. By default, all logging produced by <code>llama run</code> goes to both an archival logfile shared by all instances of the process as well as <code>STDERR</code> . The archival logfile can be overridden with this argument. If you specify <code>/dev/null</code> or a path that resolves to the same, logfile output will be suppressed automatically. Logs written to the logfile are always at maximum verbosity, i.e. <code>DEBUG</code> . (default: <code>/dev/null</code>) Default: “/dev/null”
-v, --verbosity	Possible choices: <code>debug, info, warning, error, critical, none</code> Set the verbosity level at which to log to <code>STDOUT</code> ; the <code>--logfile</code> will ALWAYS receive maximum verbosity logs (unless it is completely suppressed by writing to <code>/dev/null</code>). Available choices correspond to logging severity levels from the logging library, with the addition

of `none` if you want to completely suppress logging to standard out.
(default: `info`)

Default: “info”

49.6 `llama dev docs`

Tools for automating documentation workflows.

```
usage: llama dev docs [-V] [-h] [{cli}] [subargs [subargs ...]]
```

49.6.1 Named Arguments

-V, --version Print the version number and exit.

Default: False

-h, --help If a SUBCOMMAND is provided, run `--help` for that subcommand.
If no SUBCOMMAND is provided, print this documentation and exit.

Default: False

49.6.2 subcommands (call one with ```--help``` for details on each)

subcommand Possible choices: `cli`

If a subcommand is provided, ALL remaining arguments will be passed to that command and it will be called.

subargs If SUBCOMMAND takes its own positional arguments, provide them here.
This includes sub-sub commands.

subcommand summaries (pick one, use `--help` for more info):

cli (Semi-)automatically collect command-line interface tools with `--help` documentation and dump them into an `rst` file.

49.6.3 `llama dev docs cli`

(Semi-)automatically collect command-line interface tools with `--help` documentation and dump them into an `rst` file.

```
usage: llama dev docs cli [-h] module outdir
```

49.6.3.1 Positional Arguments

module	Fully-qualified name of the module to parse.
outdir	Where to save the resulting .rst files, which will be named <code>cli.module.rst</code> (where “module” is replaced by the module name in which each CLI script is defined).

49.7 llama dev dv

Execute actions across a bunch of DigitalOcean servers based on their names. Useful for background and sensitivity runs. Executes a shell command on each server using SSH. You can either specify shell scripts to run (in non-interactive mode, as if you were calling “`ssh foo@bar ‘my shell command’`”) or can provide your own shell command as an argument to the script.

```
usage: llama dev dv [-h] [-d] [-D] [-p PATTERN] [-l] [-f FORMAT] [-P PRELAUNCH]
                     [-n CONNECTIONS] [-t TIMEOUT] [-r {sum,linecount}]
                     [{destroy,done,eval,forget,init,killall,launch,batch,ls-procs,pull,
                     push,tail}]
                     [args [args ...]]
```

49.7.1 Positional Arguments

cmd	Possible choices: destroy, done, eval, forget, init, killall, launch, batch, ls-procs, pull, push, tail
	The command to run remotely.
args	Optional additional arguments to feed to the script specified in <code>cmd</code> . These arguments will be set using <code>set -- arg1 arg2 ... argN</code> before the script specified by <code>cmd</code> is evaluated, effectively making it as if these arguments were passed as <code>\$1</code> through <code>\$N</code> to that shell script.

49.7.2 Named Arguments

-d, --descriptions	Print names and descriptions of available commands (see “ <code>cmd</code> ” above).
	Default: False
-D, --defaults	Print default command line flags for each available command.
	Default: False
-p, --pattern	An fnmatch-style pattern that droplet names must match. Droplets whose names do not match this pattern will not be included. (default: <code>llama-*</code>)
	Default: “ <code>llama-*</code> ”
-l, --local	Run the given command locally instead of on the remote droplet. This is useful for e.g. rsync commands (and anything that is meant to interact with each droplet but which must run on the local machine). These

scripts will have the DROPNAME and DROPIP variables set to each droplet's name and IP address respectively.

Default: False

-f, --format

The format string to use to describe what happened for each command. Should be a typical python format string with “name” referring to the droplet name, “ip” referring to its IP address, “stdout” referring to the standard output of the command, “stderr” the standard error, and “retval” referring to the return code of the SSH call.

Default: “On droplet {name} at {ip} – Retval={retval} Stdout: {stdout}”

-P, --prelaunch

A python format string to print for each droplet BEFORE launch (for long-running processes). Specify the droplet name and IP as in --format (stdout, stderr, and retval are obviously not available before the thread starts).

Default: “”

-n, --connections

The max number of simultaneous connections (or processes) to make. For actions involving data transfer, set this to something low (probably 1). DEFAULT: 50

Default: 50

-t, --timeout

Max execution time on each droplet in s. No timeout by default.

Default: inf

-r, --reduce

Possible choices: sum, linecount

Specify a function that reduces the STDOOUT values, e.g. by summing them.

New commands can be added as bash scripts to `llama.dev.dv.SCRIPTS` (keep in mind that these run in a non-interactive SSH session, so you will need to initialize your environment accordingly).

These scripts can optionally specify default options (like “–format” below) by putting those arguments in a variable on a new line called “DROPVEC”, e.g. `DROPVEC=”–timeout 5 –connections 2”`.

A short description of the command can be specified in a variable on a new line called `HELP`, e.g. `HELP=”check running servers”`.

Functions to run before and after the vectorized commands (“SETUP” and “CLEANUP” commands, respectively) can be defined using those variables in the command scripts, similarly to `DROPVEC`. For example, to make a directory called “foo” before any of the vectorized commands run, put `SETUP=”mkdir foo”` somewhere in the script. To remove that directory afterwards, put `CLEANUP=”rm -r foo”` somewhere in the script. You can specify multiple SETUP and CLEANUP commands this way; they will be run sequentially as encountered. Environmental variable expansion will be performed on “SETUP” and “CLEANUP” commands in the local environment; specify environmental variables using python format strings. The number of droplets being run on is available in SETUP and CLEANUP scripts as `DROPNUM`.

49.8 llama dev log

Tools for parsing log files to extract logged information.

```
usage: llama dev log [-V] [-h] [{lvalert}] [subargs [subargs ...]]
```

49.8.1 Named Arguments

-V, --version	Print the version number and exit. Default: False
-h, --help	If a SUBCOMMAND is provided, run --help for that subcommand. If no SUBCOMMAND is provided, print this documentation and exit. Default: False

49.8.2 subcommands (call one with ``--help`` for details on each)

subcommand	Possible choices: lvalert If a subcommand is provided, ALL remaining arguments will be passed to that command and it will be called.
subargs	If SUBCOMMAND takes its own positional arguments, provide them here. This includes sub-sub commands.

subcommand summaries (pick one, use --help for more info):

lvalert Extract LVAlert message JSON from lvalert_listen log files.

49.8.3 llama dev log lvalert

Extract LVAlert message JSON from lvalert_listen log files. Reads the log file contents from STDIN and writes the JSON packets to STDOUT as a pretty-printed JSON list of message objects. Errors are logged to STDERR. Keeps reading from STDIN, so in principle you can do something like tail -f lvalert_listen.log to keep track of new messages in real-time (while also logging them).

```
usage: llama dev log lvalert [-h] [-s SCRIPT]
```

49.8.3.1 Named Arguments

-s, --script	A script that should be used as an LVAAlert handler for all well-defined LVAAlerts found in the log read in from STDIN. The LVAAlert Node will be passed as the first command line argument to this script; the FAR (False Alarm Rate) will be passed as the second argument. The LVAAlert JSON payload will be passed to this script through STDIN. If this argument is provided, then nothing will be printed to STDOUT, and instead, the script defined in this argument will be invoked once in the described manner for each parsed LVAAlert. Only one instance of the script runs at a time. STDOUT and STDERR from the invoked scripts will be forwarded to STDOUT and STDERR for this calling script.
---------------------	---

49.9 llama dev upload

Tools for uploading manifests of data files to a DigitalOcean Spaces/Amazon S3 object storage solution (for later user installation using `llama install`).

Prints out a manifest as a JSON dictionary mapping local filenames to tuples of corresponding remote URLs and file hashes.

```
usage: llama dev upload [-h] [-l LOGFILE] [-v {debug,info,warning,error,critical,none}]
                         [-r ROOT] [-g GLOB] [-p PREFIX] [-b BUCKET] [-P] [-R]
                         [--dry-run]
```

49.9.1 Named Arguments

-r, --root	The root directory to upload to remote storage. (default: current directory). Paths in the printed manifest will be relative to this. Default: “.”
-g, --glob	A glob matching files in --root that should be uploaded and thrown in the manifest. (default: all files in all subdirectories). Default: “**/*”
-p, --prefix	A prefix to the key used on the remote storage instance. This can be something pathlike with / indicating a directory, but note that it is just a prefix prepended to the front of the relative paths found in --root matching --glob, so you’ll have to add a trailing slash if you want this to be a containing directory. (default: llama/objects/) Default: “llama/objects/”
-b, --bucket	The storage bucket to upload to. For DigitalOcean Spaces, this is just the name of the directory in that space’s top-level directory. (default: llama) Default: “llama”

-P, --private	Upload this file privately, leaving no public link. You probably don't want to do this unless you're just dumping data onto a storage space to deal with later.
	Default: False
-R, --relpath	If specified, then the relative path of files from <code>--root</code> will become part of their key on the remote object store, sandwiched between the <code>--prefix</code> and the file's sha256sum. Use this if you are manually uploading something to the object store that you want to keep track of outside of source-code, e.g. one-off build artifacts and the like. You should <i>AVOID</i> using this flag if you're just interested in throwing files in an object store and organizing them with the returned manifest in a version-controlled way, since objects are by default only named after their hash. This lets you use the manifest to specify where files go on the filesystem without duplicating files on the remote object store.
	Default: False
--dry-run	If specified, just print the manifest that would result from an upload and quit. Use this to see where your files will be uploaded before actually doing it.
	Default: False

49.9.2 logging settings

-l, --logfile	File where logs should be written. By default, all logging produced by <code>llama run</code> goes to both an archival logfile shared by all instances of the process as well as <code>STDERR</code> . The archival logfile can be overridden with this argument. If you specify <code>/dev/null</code> or a path that resolves to the same, logfile output will be suppressed automatically. Logs written to the logfile are always at maximum verbosity, i.e. <code>DEBUG</code> . (default: <code>/dev/null</code>)
	Default: “ <code>/dev/null</code> ”
-v, --verbosity	Possible choices: <code>debug</code> , <code>info</code> , <code>warning</code> , <code>error</code> , <code>critical</code> , <code>none</code>
	Set the verbosity level at which to log to <code>STDOUT</code> ; the <code>--logfile</code> will ALWAYS receive maximum verbosity logs (unless it is completely suppressed by writing to <code>/dev/null</code>). Available choices correspond to logging severity levels from the <code>logging</code> library, with the addition of <code>none</code> if you want to completely suppress logging to standard out. (default: <code>info</code>)

Default: “`info`”

LLAMA EVENT

Display the current status of some event directory. Defaults to the current directory if no argument provided. Specify the pipeline (collection of files) whose status you would like to see using pipeline flags or just show the status of the entire `llama.pipeline.DEFAULT_PIPELINE`.

```
usage: llama event [-h] [-f [FILEHANDLER [FILEHANDLER ...]]]
                     [+f [FILEHANDLER [FILEHANDLER ...]]]
                     [-p {DEFAULT_PIPELINE,SUBTHRESHOLD_PIPELINE,LLAMA2 REVIEW_PIPELINE}]
                     [--print-default-pipeline] [--dry-run-pipeline]
                     [eventdirs [eventdirs ...]]
```

50.1 Positional Arguments

eventdirs	Specify the directories whose statuses you want to show. Run on the current directory by default. Default: ['/Users/s/dev/multimessenger-pipeline-o3/docs']
------------------	--

50.2 choose pipeline (see ``llama.pipeline``)

-f, --filehandlers	Possible choices: LVAlertJSON, CoincSignificanceI3Lvc, CoincScatterI3LvcPdf, RctSlkLmaCoincSignificanceI3Lvc, LVCGraceDbEventData, ZtfTriggerList, IceCubeNeutrinoListCoincTxt, RctSlkLmaCoincScatterI3LvcPng, CoincSummaryI3LvcPdf, RctSlkLmaLvcGcnXml, Advok, LvcSkymapHdf5, CoincScatterZtfLVCpng, IceCubeNeutrinoListTxt, RctSlkLmaCoincScatterZtfLVCpng, LvcSkymapFits, LvcGcnXml, IceCubeNeutrinoList, SkymapInfo, CoincSummaryI3LvcTex, RctSlkLmaLVCGraceDbEventData, CoincScatterZtfI3LvcPdf, RctSlkLmaCoincScatterZtfLVCpdf, LvcDistancesJson, LVAlertAdvok, CoincScatterI3LvcPng, CoincScatterZtfLVCpdf, RctSlkLmaLVAlertJSON, RctSlkLmaLvcRetractionXml, LvcRetractionXml, RctSlkLmaCoincSummaryI3LvcPdf, RctSlkLmaCoincScatterZtfI3LvcPng, CoincSignificanceSubthresholdI3Lvc, RctSlkLmaCoincSignificanceSubthresholdI3Lvc, RctSlkI3CoincSummaryI3LvcPdf, RctSlkLmaLvcDistancesJson, IceCubeNeutrinoListTex, RctSlkLmaCoincScatterI3LvcPdf, RctSlkLmaCoincScatterZtfI3LvcPdf, PAstro, FermiGRBsJSON, RctSlkLmaSkymapInfo, CoincScatterZtfI3LvcPng
---------------------------	---

A list of `FileHandler` class names which should be used. If provided, `FileHandler` classes whose names are in this list will be included in the pipeline. If the dependencies of a requested file are not available, no attempt will be made to generate them unless they too are listed explicitly. Available filehandlers are drawn from the `DEFAULT_PIPELINE` (print them with `--print-default-pipeline`).

+f, ++filehandlers	Possible choices: <code>LVAlertJSON</code> , <code>CoincSignificanceI3Lvc</code> , <code>CoincScatterI3LvcPdf</code> , <code>RctSlkLmaCoincSignificanceI3Lvc</code> , <code>LVCGraceDbEventData</code> , <code>ZtfTriggerList</code> , <code>IceCubeNeutrinoListCoincTxt</code> , <code>RctSlkLmaCoincScatterI3LvcPng</code> , <code>CoincSummaryI3LvcPdf</code> , <code>RctSlkLmaLvcGcnXml</code> , <code>Advok</code> , <code>LvcSkymapHdf5</code> , <code>CoincScatterZtfLVCPng</code> , <code>IceCubeNeutrinoListTxt</code> , <code>RctSlkLmaCoincScatterZtfLVCPng</code> , <code>LvcSkymapFits</code> , <code>LvcGcnXml</code> , <code>IceCubeNeutrinoList</code> , <code>SkymapInfo</code> , <code>CoincSummaryI3LvcTex</code> , <code>RctSlkLmaLVCGraceDbEventData</code> , <code>CoincScatterZtfI3LvcPdf</code> , <code>RctSlkLmaCoincScatterZtfLVCPdf</code> , <code>LvcDistancesJson</code> , <code>LVAlertAdvok</code> , <code>CoincScatterI3LvcPng</code> , <code>CoincScatterZtfLVCPdf</code> , <code>RctSlkLmaLVAlertJSON</code> , <code>RctSlkLmaLvcRetractionXml</code> , <code>LvcRetractionXml</code> , <code>RctSlkLmaCoincSummaryI3LvcPdf</code> , <code>RctSlkLmaCoincScatterZtfI3LvcPng</code> , <code>CoincSignificanceSubthresholdI3Lvc</code> , <code>RctSlkLmaCoincSignificanceSubthresholdI3Lvc</code> , <code>RctSlkI3CoincSummaryI3LvcPdf</code> , <code>RctSlkLmaLvcDistancesJson</code> , <code>IceCubeNeutrinoListTex</code> , <code>RctSlkLmaCoincScatterI3LvcPdf</code> , <code>RctSlkLmaCoincScatterZtfI3LvcPdf</code> , <code>PAstro</code> , <code>FermiGRBsJSON</code> , <code>RctSlkLmaSkymapInfo</code> , <code>CoincScatterZtfI3LvcPng</code>
	Exact same as <code>--filehandlers</code> , but all ancestors of the files listed with the <code>+</code> prefix will also be included. This means that, if you ask to generate a single file, an attempt will be made to also generate everything it depends on if necessary. If you want all those files made, this is a handy shortcut; if you want file generation to fail when ancestors are missing, use the <code>-</code> prefix instead.
-p, --pipeline	Possible choices: <code>DEFAULT_PIPELINE</code> , <code>SUBTHRESHOLD_PIPELINE</code> , <code>LLAMA2 REVIEW_PIPELINE</code>
	The name of the pipeline to use. Must be the name of a Pipeline instance from <code>llama.pipeline</code> . Available choices: <code>['DEFAULT_PIPELINE', 'SUBTHRESHOLD_PIPELINE', 'LLAMA2 REVIEW_PIPELINE']</code> . If both this and <code>filehandlers</code> are specified, then the resulting pipeline will include all requested filehandlers from both options.
--print-default-pipeline	Print the contents of the default pipeline and quit.
--dry-run-pipeline	Print the pipeline selected by the user and quit without taking further action. Use this to make sure you've specified the correct pipeline.
	Default: False

CHAPTER
FIFTYONE

LLAMA FILES

Commands related to making LLAMA analysis files.

```
usage: llama files [-V] [-h] [{i3,skymap_info}] [subargs [subargs ...]]
```

51.1 Named Arguments

-V, --version Print the version number and exit.

Default: False

-h, --help If a SUBCOMMAND is provided, run --help for that subcommand.
If no SUBCOMMAND is provided, print this documentation and exit.

Default: False

51.2 subcommands (call one with ``--help`` for details on each)

subcommand Possible choices: i3, skymap_info

If a subcommand is provided, ALL remaining arguments will be passed to that command and it will be called.

subargs If SUBCOMMAND takes its own positional arguments, provide them here.
This includes sub-sub commands.

subcommand summaries (pick one, use --help for more info):

i3 Command Line Interface for manually creating IceCube GFU neutrino lists (by pulling them from either archives or the GFU API) and returning the neutrino lists in the desired formats.

skymap_info Manually create a ``skymap_info``.

51.3 llama files i3

Command Line Interface for manually creating IceCube GFU neutrino lists (by pulling them from either archives or the GFU API) and returning the neutrino lists in the desired formats. You must specify either a time window with `--interval` or a central trigger time with `--time` (the way you would do for a triggered event).

```
usage: llama files i3 [-h] [-t TIME] [-i START STOP] [-u] [--mjd] [-p [PREFIX]]  
                      [--json [JSON]] [--txt [TXT]]
```

51.3.1 Named Arguments

-t, --time

If specified, this is the central time of the search (for the common use case where we are triggering on an external event happening at this time and searching for neutrinos in the surrounding time window). If this can be parsed as a float, it will be interpreted as a GPS timestamp (unless `--mjd` is specified, in which case it will be interpreted as an MJD date). Otherwise, an attempt will be made to parse it as a UTC datetime string, e.g. “2019-05-01T12:34:56”.

-i, --interval

Provide an interval surrounding `--time` in which to search. If `--time` is provided, find neutrinos in the interval (TIME+START, TIME+STOP) and interpret START and STOP as seconds (or, if `--mjd` is provided, as days). If `--time` is omitted, find neutrinos in the interval (START, STOP), where both variables are interpreted as dates in the same way as `--time`; note that in this case you **must** specify (START, STOP) since it cannot be inferred. Defaults to the same values as used by `IceCubeNeutrinoList`: (-500, 500)

-u, --unblinded

Whether to unblind the neutrinos. **If not specified, neutrinos will be blinded.**

Default: False

--mjd

If specified, interpret numerical arguments to `--time` or `--interval` as MJD dates rather than GPS times.

Default: False

-p, --prefix

If specified, prepend the neutrino search interval to the default filenames in a human-readable format. Optionally provide an explicit prefix for these default filenames to override this default. If specific filenames are given, this option is ignored.

Default: False

51.3.2 output formats (specify at least 1)

--json Save this as a JSON file (LLAMA's internal storage format used by the pipeline). Optionally specify a file path (default: icecube_neutrino_list.json) in the current directory.

Default: False

--txt Save this as a human-readable ASCII table. Optionally specify a file path (default: icecube_neutrino_list.txt) in the current directory.

Default: False

You must specify at least 1 output format from above. Default names are the names as used in the pipeline (augmentable with the **--prefix** argument). Existing files will not be overwritten.

CHAPTER
FIFTYTWO

LLAMA FLAGS

Set flags for an event directory from the command line. Only flags specified with `--flags` will be changed. To print current flags without changing values, provide `--flags` with no arguments.

```
usage: llama flags [-h] [--dry-run-dirs] [--downselect DOWNSELECT]
                   [--print-downselections]
                   [--flags [FLAGNAME=value [FLAGNAME=value ...]]] [--flag-presets]
                   [--dry-run-flags]
                   [run]
```

52.1 Named Arguments

--flags	A single flag preset (see: <code>llama.flags</code>) to use (print choices with <code>--flag-presets</code>) in <code>--outdir</code> OR individual flag settings in the format <code>FLAGNAME=value</code> . YOU SHOULD PROBABLY USE A PRESET rather than individual flag settings. Any omitted flags will take on the default values in <code>DEFAULT_FLAGS</code> . If you don't specify <code>--flags</code> , you'll be prompted to provide one; just provide <code>--flags</code> with no arguments to accept the default/existing flags. Flags are used to set overall behaviors for an event and set intentions, e.g. to mark an event as "ONLINE" and therefore allowed to communicate with partner web APIs and send out products and alerts. Flag name options and default values (for new events) are <code>FlagPreset({ 'BLINDED_NEUTRINOS': 'false', 'MANUAL': 'false', 'VETOED': 'false', 'ONLINE': 'true', 'ICECUBE_UPLOAD': 'false', 'UPLOAD': 'false', 'ROLE': 'test' })</code> ; the full set of allowed values is ('true', 'false') for ONLINE, ('true', 'false') for BLINDED_NEUTRINOS, ('true', 'false') for MANUAL, ('true', 'false') for VETOED, ('true', 'false') for ICECUBE_UPLOAD, ('test', 'observation') for ROLE, and ('true', 'false') for UPLOAD.
--flag-presets	Print available flag presets.
--dry-run-flags	Print flags parsed by the CLI and exit. Default: False

52.2 filter runs and events (see: ``llama.run``)

run	A pattern specifying a list of directories to update of the form /run/directory/event*glob. See end of llama run -h documentation for details. (default: /Users/s/.local/share/llama/current_run/*)
--dry-run-dirs	Print the runs and event directories that would be affected and exit without taking further action.
	Default: False
--downselect	Arguments to pass to the downselect method of runs selected by the run argument (note that eventid_filter is already implicitly set by the glob pattern specified in run). Arguments will only be parsed as booleans (if they equal “True” or “False”), ints (if they can be parsed as such), floats (if they can be parsed as such), or strings and should be separated by commas, e.g. --downselect manual=False,fhnameexists=PAstro. Omit a list of downselections or provide an empty string to specify no further downselections beyond the one implied by the run argument. (default: None)
--print-downselections	Print available downselections.

LLAMA INSTALL

Run post-installation routines. Downloads and installs large library data files from remote directories.

```
usage: llama install [-h] [-l LOGFILE] [-v {debug,info,warning,error,critical,none}]  
                      [-d {data}]
```

53.1 Named Arguments

-d, --download Possible choices: data

Download and install data from specified destinations. These files are data files that are necessary for the pipeline to function.

53.2 logging settings

-l, --logfile File where logs should be written. By default, all logging produced by `llama run` goes to both an archival logfile shared by all instances of the process as well as STDERR. The archival logfile can be overridden with this argument. If you specify `/dev/null` or a path that resolves to the same, logfile output will be suppressed automatically. Logs written to the logfile are always at maximum verbosity, i.e. DEBUG. (default: `/dev/null`)

Default: “`/dev/null`”

-v, --verbosity Possible choices: debug, info, warning, error, critical, none

Set the verbosity level at which to log to STDOUT; the `--logfile` will ALWAYS receive maximum verbosity logs (unless it is completely suppressed by writing to `/dev/null`). Available choices correspond to logging severity levels from the `logging` library, with the addition of `none` if you want to completely suppress logging to standard out. (default: `info`)

Default: “`info`”

CHAPTER
FIFTYFOUR

LLAMA LISTEN

Listen for external triggers and react to them as they come in.

```
usage: llama listen [-V] [-h] [{gcn,lvalert}] [subargs [subargs ...]]
```

54.1 Named Arguments

-V, --version Print the version number and exit.

Default: False

-h, --help If a SUBCOMMAND is provided, run --help for that subcommand.
If no SUBCOMMAND is provided, print this documentation and exit.

Default: False

54.2 subcommands (call one with ``--help`` for details on each)

subcommand Possible choices: gcn, lvalert

If a subcommand is provided, ALL remaining arguments will be passed to that command and it will be called.

subargs If SUBCOMMAND takes its own positional arguments, provide them here.
This includes sub-sub commands.

subcommand summaries (pick one, use --help for more info):

gcn Launch the GCN Notice listener.

lvalert Listen for new LVAAlerts from GraceDB containing GW triggers.

54.3 llama listen gcn

Launch the GCN Notice listener.

```
usage: llama listen gcn [-h] [-l LOGFILE] [-v {debug,info,warning,error,critical,none}]
                         [-r RUNDIR] [-L] [-g GCN_ARCHIVE] [-i HOST] [-p PORT]
```

54.3.1 Named Arguments

-r, --rundir	Change the run directory (i.e. where new LLAMA triggers are saved by the GCN handler) to this directory. Creates the directory if it does not already exist. (default: /Users/s/.local/share/llama/current_run/) Default: "/Users/s/.local/share/llama/current_run/"
-L, --lvc-private	If provided, listen for private LIGO GCN circulars on port 8099. This argument will OVERRIDE the -p--port argument. Default: False
-g, --gcn-archive	Change the GCN archive directory (i.e. where all GCN Notices are archived too for recordkeeping). Creates the directory if it does not already exist. Default: "/Users/s/.local/share/llama/gcn/archive"
-i, --host	The host to listen to. Defaults to the main GCN server: 68.169.57.253 Default: "68.169.57.253"
-p, --port	The port on the GCN server to connect to. Default: 8099 Default: 8099

54.3.2 logging settings

-l, --logfile	File where logs should be written. By default, all logging produced by llama run goes to both an archival logfile shared by all instances of the process as well as STDERR. The archival logfile can be overridden with this argument. If you specify /dev/null or a path that resolves to the same, logfile output will be suppressed automatically. Logs written to the logfile are always at maximum verbosity, i.e. DEBUG. (default: /Users/s/.local/share/llama/logs/gcn.log) Default: "/Users/s/.local/share/llama/logs/gcn.log"
-v, --verbosity	Possible choices: debug, info, warning, error, critical, none Set the verbosity level at which to log to STDOUT; the --logfile will ALWAYS receive maximum verbosity logs (unless it is completely suppressed by writing to /dev/null). Available choices correspond to logging severity levels from the logging library, with the addition of none if you want to completely suppress logging to standard out. (default: debug) Default: "debug"

54.4 llama listen lvalert

Listen for new LVAAlerts from GraceDB containing GW triggers. This is LIGO/Virgo's internal infrastructure for distributing notifications about updates to all gravitational wave (GW) triggers.

```
usage: llama listen lvalert [-h] [-l LOGFILE]
                            [-v {debug,info,warning,error,critical,none}] [-r RUNDIR]
                            [-n [NODES [NODES ...]]] [-s SERVER]
                            [--list-nodes [{lvalert.cgca.uwm.edu,lvalert-playground.cgca.
                            ↵uwm.edu}]]
```

54.4.1 Named Arguments

-r, --rundir	Change the run directory (i.e. where new LLAMA triggers are saved by the GCN handler) to this directory. Creates the directory if it does not already exist. (default: /Users/s/.local/share/llama/current_run/)
	Default: "/Users/s/.local/share/llama/current_run/"
-n, --nodes	The LVAAlert nodes to subscribe to. Make sure to subscribe to all the searches you are interested in. (default: {'cbc_gstlal', 'cbc_mbtaonline', 'cbc_lowmass', 'cbc_pycbc', 'test_superevent', 'superevent', 'burst_cwb', 'cbc_spiir', 'stc-testnode'})
	Default: ('cbc_gstlal', 'cbc_mbtaonline', 'cbc_lowmass', 'cbc_pycbc', 'test_superevent', 'superevent', 'burst_cwb', 'cbc_spiir', 'stc-testnode')
-s, --server	The server to subscribe to. (default: lvalert.cgca.uwm.edu)
	Default: "lvalert.cgca.uwm.edu"
--list-nodes	Possible choices: lvalert.cgca.uwm.edu, lvalert-playground.cgca.uwm.edu
	Print available LVAAlert nodes for the given server and exit. (default: lvalert.cgca.uwm.edu)

54.4.2 logging settings

-l, --logfile	File where logs should be written. By default, all logging produced by <code>llama</code> run goes to both an archival logfile shared by all instances of the process as well as STDERR. The archival logfile can be overridden with this argument. If you specify /dev/null or a path that resolves to the same, logfile output will be suppressed automatically. Logs written to the logfile are always at maximum verbosity, i.e. DEBUG. (default: /Users/s/.local/share/llama/logs/lvalert.log)
	Default: "/Users/s/.local/share/llama/logs/lvalert.log"
-v, --verbosity	Possible choices: debug, info, warning, error, critical, none
	Set the verbosity level at which to log to STDOUT; the --logfile will ALWAYS receive maximum verbosity logs (unless it is completely suppressed by writing to /dev/null). Available choices correspond to logging severity levels from the logging library, with the addition

of `None` if you want to completely suppress logging to standard out.
(default: `debug`)

Default: “debug”

LLAMA POLL

Scripts that continuously poll external resources for updates. Used in cases when trigger listeners (see: `llama.listen`), a more elegant approach to querying for updates, are not available.

```
usage: llama poll [-V] [-h] [{gracedb}] [subargs [subargs ...]]
```

55.1 Named Arguments

-V, --version Print the version number and exit.

Default: False

-h, --help If a SUBCOMMAND is provided, run `--help` for that subcommand. If no SUBCOMMAND is provided, print this documentation and exit.

Default: False

55.2 subcommands (call one with ```--help``` for details on each)

subcommand Possible choices: gracedb

If a subcommand is provided, ALL remaining arguments will be passed to that command and it will be called.

subargs If SUBCOMMAND takes its own positional arguments, provide them here. This includes sub-sub commands.

subcommand summaries (pick one, use `--help` for more info):

gracedb Launch the LLAMA GCN poller.

55.3 `llama poll gracedb`

Launch the LLAMA GCN poller.

Poll GraceDB for new superevents that have been marked EM_READY and have had a GCN Notice VOEvent generated. Download and handle those GCN Notices. Runs as a script checking for new VOEvents when called from the command line.

```
usage: llama poll gracedb [-h]
```

CHAPTER
FIFTYSIX

LLAMA RUN

The main interface for running the LLAMA pipeline. Turns on the automated pipeline, either in the foreground (for immediate work) or in the background (launching a long-running daemon process).

The pipeline checks for new LLAMA events and event data and (re)feteches/(re)generates data files for each event as new data becomes available. Provides a very flexible and powerful command line interface for running LLAMA.

```
usage: llama run [-h] [-V] [-D] [-f [FILEHANDLER [FILEHANDLER ...]]]
                  [+f [FILEHANDLER [FILEHANDLER ...]]]
                  [-p {DEFAULT_PIPELINE,SUBTHRESHOLD_PIPELINE,LLAMA2 REVIEW_PIPELINE}]
                  [--print-default-pipeline] [--dry-run-pipeline] [--dry-run-dirs]
                  [--downselect DOWNSELECT] [--print-downselections] [-l LOGFILE]
                  [-v {debug,info,warning,error,critical,none}] [-o] [-n NUM_UPDATED]
                  [-F] [-R] [-k] [-K] [-r]
                  [run]
```

56.1 Named Arguments

-V, --version	Print the version number and exit. Default: False
-D, --dev-mode	If specified, allow the program to run even if the LLAMA version does not conform to semantic version naming. You should not do this during production except in an emergency. If the flag is not specified but local changes to the source code exist, <code>llama run</code> will complain and quit immediately (the default behavior). Default: False
-o, --runonce	If specified, each event directory will only be updated once, and then the script will exit. Default: False
-n, --num-updated	Number of event directories to keep updated. Older event directories will be skipped and not updated if they do not fall within the top <code>num_updated</code> . Default: 40
-F, --foreground	Run <code>llama run</code> in the foreground instead of immediately sending it to the background as a daemon process. If running in the foreground, logs

are printed to STDERR (in addition to the global archival logfile, see `--logfile`); if running in the background (i.e. omitting this flag), logs are appended to a logfile specific to this set of event directories (again, in addition to the archival logfile). This CLI-argument-specific logfile goes in the same log directory as the default `--logfile` option but has a filename specific to this `llama run` invocation; the name is the same as the `lockdir` name (see: `lockdir`, but with `.log` appended).

Default: False

-R, --running-daemons Print the currently-running llama daemons as recorded in the `llama run` lockfile directory, `/Users/s/.cache/llama/llamad.run.d`. Note that this will not catch `llama run` processes whose lock files have somehow been deleted from this directory. Run `ps -ax | grep 'llama run'` if you're paranoid about missing anything.

-k, --kill Kill any llama processes covering potentially competing `run` and `eventidfilter` values (as decided by conservative comparison algorithms) by finding their PIDs in their semaphore lock directories and killing the associated processes (if they are still running). You can run this if a `llama run` instance fails to launch due to competing instances in order to kill them safely.

Default: False

-K, --killfirst Like `--kill`, but continues execution after killing. In other words, start a new process but kill competition first.

Default: False

-r, --relaunch If `relaunch` is specified, then wrap `llama run` as a subprocess and keep relaunching it if it fails; if it exits gracefully (returncode 0), it will not be relaunched. All command line arguments (besides this one) will be passed to subprocess `llama run` invocations.

Default: False

56.2 choose pipeline (see ```llama.pipeline```)

-f, --filehandlers Possible choices: LVAlertJSON, CoincSignificanceI3Lvc, CoincScatterI3LvcPdf, RctSlkLmaCoincSignificanceI3Lvc, LVCGraceDbEventData, ZtfTriggerList, IceCubeNeutrinoListCoincTxt, RctSlkLmaCoincScatterI3LvcPng, CoincSummaryI3LvcPdf, RctSlkLmaLvcGenXml, Advok, LvcSkymapHdf5, CoincScatterZtfLVCPng, IceCubeNeutrinoListTxt, RctSlkLmaCoincScatterZtfLVCPng, LvcSkymapFits, LvcGcnXml, IceCubeNeutrinoList, SkymapInfo, CoincSummaryI3LvcTex, RctSlkLmaLVCGraceDbEventData, CoincScatterZtfI3LvcPdf, RctSlkLmaCoincScatterZtfLVCPdf, LvcDistancesJson, LVAlertAdvok, CoincScatterI3LvcPng, CoincScatterZtfLVCPdf, RctSlkLmaLVAlertJSON, RctSlkLmaLvcRetractionXml, LvcRetractionXml, RctSlkLmaCoincSummaryI3LvcPdf, RctSlkLmaCoincScatterZtfI3LvcPng, CoincSignificanceSubthresholdI3Lvc, RctSlkLmaCoincSignificanceSubthresholdI3Lvc, RctSlkI3CoincSummaryI3LvcPdf, RctSlkLmaLvcDistancesJson, IceCubeNeutrinoListTex, RctSlkLmaCoincScatterI3LvcPdf, RctSlkLmaCoincScatterZtfI3LvcPdf, PAstro, FermiGRBsJSON, RctSlkLmaSkymapInfo, CoincScatterZtfI3LvcPng

A list of `FileHandler` class names which should be used. If provided, `FileHandler` classes whose names are in this list will be included in the pipeline. If the dependencies of a requested file are not available, no attempt will be made to generate them unless they too are listed explicitly. Available filehandlers are drawn from the `DEFAULT_PIPELINE` (print them with `--print-default-pipeline`).

+f, ++filehandlers	Possible choices: <code>LVAlertJSON</code> , <code>CoincSignificanceI3Lvc</code> , <code>CoincScatterI3LvcPdf</code> , <code>RctSlkLmaCoincSignificanceI3Lvc</code> , <code>LVCGraceDbEventData</code> , <code>ZtfTriggerList</code> , <code>IceCubeNeutrinoListCoincTxt</code> , <code>RctSlkLmaCoincScatterI3LvcPng</code> , <code>CoincSummaryI3LvcPdf</code> , <code>RctSlkLmaLvcGcnXml</code> , <code>Advok</code> , <code>LvcSkymapHdf5</code> , <code>CoincScatterZtfLVCPng</code> , <code>IceCubeNeutrinoListTxt</code> , <code>RctSlkLmaCoincScatterZtfLVCPng</code> , <code>LvcSkymapFits</code> , <code>LvcGcnXml</code> , <code>IceCubeNeutrinoList</code> , <code>SkymapInfo</code> , <code>CoincSummaryI3LvcTex</code> , <code>RctSlkLmaLVCGraceDbEventData</code> , <code>CoincScatterZtfI3LvcPdf</code> , <code>RctSlkLmaCoincScatterZtfLVCPdf</code> , <code>LvcDistancesJson</code> , <code>LVAlertAdvok</code> , <code>CoincScatterI3LvcPng</code> , <code>CoincScatterZtfLVCPdf</code> , <code>RctSlkLmaLVAlertJSON</code> , <code>RctSlkLmaLvcRetractionXml</code> , <code>LvcRetractionXml</code> , <code>RctSlkLmaCoincSummaryI3LvcPdf</code> , <code>RctSlkLmaCoincScatterZtfI3LvcPng</code> , <code>CoincSignificanceSubthresholdI3Lvc</code> , <code>RctSlkLmaCoincSignificanceSubthresholdI3Lvc</code> , <code>RctSlkI3CoincSummaryI3LvcPdf</code> , <code>RctSlkLmaLvcDistancesJson</code> , <code>IceCubeNeutrinoListTex</code> , <code>RctSlkLmaCoincScatterI3LvcPdf</code> , <code>RctSlkLmaCoincScatterZtfI3LvcPdf</code> , <code>PAstro</code> , <code>FermiGRBsJSON</code> , <code>RctSlkLmaSkymapInfo</code> , <code>CoincScatterZtfI3LvcPng</code>
	Exact same as <code>--filehandlers</code> , but all ancestors of the files listed with the <code>+</code> prefix will also be included. This means that, if you ask to generate a single file, an attempt will be made to also generate everything it depends on if necessary. If you want all those files made, this is a handy shortcut; if you want file generation to fail when ancestors are missing, use the <code>-</code> prefix instead.
-p, --pipeline	Possible choices: <code>DEFAULT_PIPELINE</code> , <code>SUBTHRESHOLD_PIPELINE</code> , <code>LLAMA2 REVIEW_PIPELINE</code>
	The name of the pipeline to use. Must be the name of a Pipeline instance from <code>llama.pipeline</code> . Available choices: <code>['DEFAULT_PIPELINE', 'SUBTHRESHOLD_PIPELINE', 'LLAMA2 REVIEW_PIPELINE']</code> . If both this and <code>filehandlers</code> are specified, then the resulting pipeline will include all requested filehandlers from both options.
--print-default-pipeline	Print the contents of the default pipeline and quit.
--dry-run-pipeline	Print the pipeline selected by the user and quit without taking further action. Use this to make sure you've specified the correct pipeline.
	Default: False

56.3 filter runs and events (see: ``llama.run``)

run	A pattern specifying a list of directories to update of the form /run/directory/event*glob. See end of <code>llama run -h</code> documentation for details. (default: <code>/Users/s/.local/share/llama/current_run/*</code>)
--dry-run-dirs	Print the runs and event directories that would be affected and exit without taking further action.
	Default: False
--downselect	Arguments to pass to the <code>downselect</code> method of runs selected by the <code>run</code> argument (note that <code>eventid_filter</code> is already implicitly set by the glob pattern specified in <code>run</code>). Arguments will only be parsed as booleans (if they equal “True” or “False”), ints (if they can be parsed as such), floats (if they can be parsed as such), or strings and should be separated by commas, e.g. <code>--downselect manual=False, fnameexists=PAstro</code> . Omit a list of downselections or provide an empty string to specify no further downselections beyond the one implied by the <code>run</code> argument. (default: <code>manual=False</code>)
	Default: “ <code>manual=False</code> ”
--print-downselections	Print available downselections.

56.4 logging settings

-l, --logfile	File where logs should be written. By default, all logging produced by <code>llama run</code> goes to both an archival logfile shared by all instances of the process as well as <code>STDERR</code> . The archival logfile can be overridden with this argument. If you specify <code>/dev/null</code> or a path that resolves to the same, logfile output will be suppressed automatically. Logs written to the logfile are always at maximum verbosity, i.e. <code>DEBUG</code> . (default: <code>/Users/s/.local/share/llama/logs/llamad.log</code>)
	Default: “ <code>/Users/s/.local/share/llama/logs/llamad.log</code> ”
-v, --verbosity	Possible choices: <code>debug</code> , <code>info</code> , <code>warning</code> , <code>error</code> , <code>critical</code> , <code>none</code>
	Set the verbosity level at which to log to <code>STDOUT</code> ; the <code>--logfile</code> will ALWAYS receive maximum verbosity logs (unless it is completely suppressed by writing to <code>/dev/null</code>). Available choices correspond to logging severity levels from the <code>logging</code> library, with the addition of <code>none</code> if you want to completely suppress logging to standard out. (default: <code>info</code>)
	Default: “ <code>info</code> ”

Each LLAMA trigger gets its own directory. The name of this directory is called the `eventid` and the trigger itself is a **LLAMA Event** (see: `llama.event`). For a given LLAMA run, all event directories should go in a command directory called a “run directory”; the collection of events is called a Run (see: `llama.run`). Most things the pipeline does work on a single Run and are meant to affect one or more matching Event instances. When you specify directories, you are implicitly specifying the Run (i.e. collection of triggers) as well as a UNIX-style glob (like the asterisk matching all

files, `*`) which describes the `eventid` pattern you want to match. For example, matching all event IDs that start with “S” (corresponding to O3 LIGO/Virgo superevents) would require using `S*` as your event glob.

If you want to explicitly print which currently-existing Event directories will be impacted by the arguments you provide, you can use `--dry-run-dirs` to print the impacted directories and exit without taking further action. This is good practice while getting used to this interface.

The syntax for specifying the Run and Event glob is the path of the run directory followed by a slash followed by the event glob with **no slash at the end** (be sure to escape the `*` so the shell doesn’t expand it):

```
'/run/directory/event*glob'
```

Specify **only** the event glob by leaving the run directory out but keeping the leading `/` (if for some insane reason your root directory is your run directory, a double-leading `/` will communicate your perverse desire). In this case the default Run directory `/Users/s/.local/share/llama/current_run/` is implied, so the following are equivalent:

```
'/event*glob'  
/Users/s/.local/share/llama/current_run/'event*glob'
```

Specify **only** the Run directory by leaving a trailing slash and omitting the event glob; in this case, the default event glob `*` will be used, so the following are equivalent:

```
/run/directory/  
/run/directory/*
```

You can use relative paths for the Run directory, the final part of the path will **not** be expanded and will be treated as the base directory. The only exception to this is if you are using relative paths and don’t put *any* `/` in the specified path, in which case the relative path will be expanded. This allows the common and intuitive behavior of running specific events in the current directory when you pass their name alone, or alternatively to treat the current directory as the only event directory by passing a single `.` as the run argument. Something like `./..`, however, will be interpreted as meaning you want the *current* directory to be the run directory only matching Event ids of `..`.

The following examples assume you are currently in the event directory `/some/directory/`. Let’s say this is the event directory, and you want to update **only** the contents of this directory. You can specify the run as `/some/` and the event glob as `directory` with either of the following commands paths:

```
*  
/some/directory
```

Alternatively, if `/some/directory/` is a run directory, and you want to affect the event directories it contains that match the default event glob `*`, you can run use any of the following (note again that the event glob is in quotes to prevent your shell from expanding it into multiple arguments):

```
./  
./'*'  
/some/directory/  
/some/directory/*
```

If you want to use the name of the current directory as your event glob (so that only `eventids` that have the *same* basename as your current directory are used) while **keeping** the default run directory `/Users/s/.local/share/llama/current_run/`, you would have to place a leading slash followed by the actual name of the run directory; as noted above, `./` not work because the dot will be treated literally as the `eventid` you want to use. (Note that you usually wouldn’t want to do this; why would you be in this directory if you want to operate on an event stored in a different run directory?):

```
/directory  
/Users/s/.local/share/llama/current_run/directory
```

You can further specify which types of events should be processed by specifying `--downselect` followed by a string to be passed as the arguments to `Run.downselect` (run `--print-downselections` to see possible options).

See `llama.run` and `llama.event` for more information on Run and Event objects.

Note that by default, `llama run` will start a background process and exit from the foreground, allowing you to keep working or disconnect your SSH session while `llama run` progresses. Override this behaviour with `-F`.

Continually monitor and update the default run directory

```
llama run
```

Continually monitor and update a temporary test directory but NOT the default run directory

```
llama run /tmp/llamatest
```

Keep monitoring only the current directory

```
llama run .
```

Make a single attempt to generate `IceCubeNeutrinoList` in the current directory (if its input files exist) and then quit:

```
llama run . -o -f IceCubeNeutrinoList
```

Same as above, but generate any missing `IceCubeNeutrinoList` inputs if possible:

```
llama run . -o +f IceCubeNeutrinoList
```

Same as above, but instead of spinning up a background process, keep control of the terminal and print all logging output until the job is done:

```
llama run . -o +f IceCubeNeutrinoList -F
```

CHAPTER
FIFTYSEVEN

LLAMA SERVE

Control LLAMA server processes (for example: interactive status website).

```
usage: llama serve [-V] [-h] [{gui,jupyter}] [subargs [subargs ...]]
```

57.1 Named Arguments

-V, --version Print the version number and exit.

Default: False

-h, --help If a SUBCOMMAND is provided, run --help for that subcommand.
If no SUBCOMMAND is provided, print this documentation and exit.

Default: False

57.2 subcommands (call one with ``--help`` for details on each)

subcommand Possible choices: gui, jupyter

If a subcommand is provided, ALL remaining arguments will be passed to that command and it will be called.

subargs If SUBCOMMAND takes its own positional arguments, provide them here.
This includes sub-sub commands.

subcommand summaries (pick one, use --help for more info):

gui Serve pages for events in the current run.

jupyter Launch a Jupyter Notebook server.

57.3 llama serve gui

Serve pages for events in the current run. Runs a Flask server in debug mode. Specify the domain using environment variable LLAMA_DOMAIN (default: localhost) and the port as LLAMA_GUI_PORT (default: 80).

```
usage: llama serve gui [-h] [-l LOGFILE] [-v {debug,info,warning,error,critical,none}]  
                      [--use-auth]
```

57.3.1 Named Arguments

--use-auth Force the user to log in using basic HTTP auth. Requires that environmental variables LLAMA_GUI_USERNAME and LLAMA_GUI_PASSWORD are both set.
Default: False

57.3.2 logging settings

-l, --logfile File where logs should be written. By default, all logging produced by `llama run` goes to both an archival logfile shared by all instances of the process as well as STDERR. The archival logfile can be overridden with this argument. If you specify /dev/null or a path that resolves to the same, logfile output will be suppressed automatically. Logs written to the logfile are always at maximum verbosity, i.e. DEBUG. (default: /Users/s/.local/share/llama/logs/gui.log)

Default: “/Users/s/.local/share/llama/logs/gui.log”

-v, --verbosity Possible choices: debug, info, warning, error, critical, none

Set the verbosity level at which to log to STDOUT; the --logfile will ALWAYS receive maximum verbosity logs (unless it is completely suppressed by writing to /dev/null). Available choices correspond to logging severity levels from the `logging` library, with the addition of none if you want to completely suppress logging to standard out. (default: info)

Default: “info”

57.4 llama serve jupyter

Launch a Jupyter Notebook server. Specify the domain using environment variable LLAMA_DOMAIN (default: localhost) and the port as LLAMA_JUP_PORT (default: 8080).

```
usage: llama serve jupyter [-h] [-l LOGFILE]  
                           [-v {debug,info,warning,error,critical,none}]  
                           [--notebook-dir NOTEBOOK_DIR] [--ip IP] [-a] [-w] [-k]  
                           [--show-hidden]
```

57.4.1 Named Arguments

--notebook-dir	Where to store jupyter notebook files. By default, they will be saved in the LLAMA data directory in /Users/s/.local/share/llama
	Default: “/Users/s/.local/share/llama”
--ip	The IP address the server will listen on. (default: 0.0.0.0)
	Default: “0.0.0.0”
-a, --alert-maintainers	If provided, use <code>llama.com.slack.alert_maintainers</code> to message LLAMA maintainers with a list of active Jupyter notebooks and their login tokens. This allows Slack users to access the notebook at the provided URL. These tokens can be used to log in to this Jupyter notebook (and any others running on this server/container). BE CAREFUL WITH THESE TOKENS! They provide full access to the Jupyter notebook; you should probably only use this in production, and make sure not to share those tokens. You should also make sure to regenerate those tokens regularly. Note also that the script will fail if you try to alert maintainers without providing valid Slack credentials (see <code>llama.com.slack</code>).
	Default: False
-w, --writeable-docs	If provided, set any documentation notebooks (like README.ipynb) to writeable. Use this for development mode and then commit the notebook with <code>llama dev upload -g README.ipynb</code> (from the notebook directory) and update the remote URL for <code>llama.serve.jupyter</code> .README with the printed URL.
	Default: False
-k, --keep-docs	If provided, don't bother downloading new README if it's stored locally; in other words, keep existing documentation if present so as not to accidentally delete development work. As with <code>--writeable-docs</code> , this should probably only be used in development mode.
	Default: False
--show-hidden	If provided, show hidden files in the file browser. This is useful for modifying metadata rider files used by the pipeline.
	Default: False

57.4.2 logging settings

-l, --logfile	File where logs should be written. By default, all logging produced by <code>llama run</code> goes to both an archival logfile shared by all instances of the process as well as STDERR. The archival logfile can be overridden with this argument. If you specify <code>/dev/null</code> or a path that resolves to the same, logfile output will be suppressed automatically. Logs written to the logfile are always at maximum verbosity, i.e. DEBUG. (default: /Users/s/.local/share/llama/logs/jupyter.log)
	Default: “/Users/s/.local/share/llama/logs/jupyter.log”
-v, --verbosity	Possible choices: debug, info, warning, error, critical, none

Set the verbosity level at which to log to STDOUT; the --logfile will ALWAYS receive maximum verbosity logs (unless it is completely suppressed by writing to /dev/null). Available choices correspond to logging severity levels from the logging library, with the addition of none if you want to completely suppress logging to standard out.
(default: info)

Default: “info”

CHAPTER
FIFTYEIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

|

llama, 85
llama.batch, 93
llama.classes, 95
llama.cli, 99
llama.com, 105
llama.com.dl, 105
llama.com.do, 105
llama.com.email, 106
llama.com.gracedb, 106
llama.com.s3, 112
llama.com.slack, 113
llama.com.utils, 114
llama.detectors, 117
llama.dev, 119
llama.dev.background, 119
llama.dev.background.pvalue, 119
llama.dev.background.table, 120
llama.dev.background.table_singles, 120
llama.dev.clean, 120
llama.dev.data, 120
llama.dev.data.i3, 120
llama.dev.docs, 121
llama.dev.docs.cli, 121
llama.dev.dv, 122
llama.dev.log, 123
llama.dev.log.lvalert, 123
llama.dev.upload, 123
llama.event, 127
llama.filehandler, 131
llama.filehandler.mixins, 140
llama.files, 143
llama.files.advok, 241
llama.files.coinc_analyses, 242
llama.files.coinc_o2, 248
llama.files.coinc_plots, 254
llama.files.coinc_significance, 179
llama.files.coinc_significance.opa, 180
llama.files.coinc_significance.subthreshold,
 185
llama.files.coinc_significance.utils, 192
llama.files.fermi_grb, 269
llama.files.gcn_draft_o2, 271
llama.files.gracedb, 275
llama.files.gwastro, 279
llama.files.healpix, 194
llama.files.healpix.plotters, 198
llama.files.healpix.psf, 199
llama.files.healpix.skymap, 201
llama.files.healpix.utils, 206
llama.files.i3, 211
llama.files.i3.json, 217
llama.files.i3.realtime_tools_stubs, 224
llama.files.i3.tex, 226
llama.files.i3.txt, 227
llama.files.i3.utils, 230
llama.files.lvalert_advok, 282
llama.files.lvalert_json, 282
llama.files.lvc_gcn_xml, 283
llama.files.lvc_skymap, 232
llama.files.lvc_skymap.utils, 234
llama.files.lvc_skymap_mat, 286
llama.files.lvc_skymap_txt, 287
llama.files.matlab, 287
llama.files.skymap_info, 235
llama.files.skymap_info.cli, 237
llama.files.skymap_info.utils, 238
llama.files.slack, 238
llama.files.sms_receipts, 288
llama.files.team_receipts, 290
llama.files.timing_checks, 290
llama.files.uw_summary, 291
llama.files.ztf_trigger_list, 291
llama.flags, 293
llama.flags.cli, 294
llama.install, 297
llama.install.manifest, 297
llama.intent, 299
llama.listen, 301
llama.listen.gcn, 301
llama.listen.lvalert, 303
llama.lock, 305
llama.meta, 307
llama.pipeline, 309

llama.poll, 315
llama.poll.gracedb, 315
llama.run, 317
llama.serve, 323
llama.serve.gui, 323
llama.serve.gui.domain, 324
llama.serve.gui.wsgi, 323
llama.serve.jupyter, 324
llama.serve.jupyter.logs, 324
llama.serve.jupyter.utils, 324
llama.test, 325
llama.test.classes, 327
llama.test.test_bin, 329
llama.test.test_filehandler, 330
llama.test.test_files, 325
llama.test.test_listeners, 325
llama.test.test_listeners.test_gcn, 325
llama.test.test_pipeline, 334
llama.test.test_utils, 334
llama.utils, 337
llama.version, 347
llama.versioning, 349
llama.vetoes, 353

INDEX

A

abbrev (*llama.detectors.DetectorTuple* attribute), 118
AbstractFileGenerationChecker (class in *llama.test.classes*), 327
AbstractFileGenerationComparator (class in *llama.test.classes*), 327
AbstractGcndTester (class in *llama.test.test_listeners.test_gcn*), 325
AbstractTestLlamaHandlers (class in *llama.test.test_listeners.test_gcn*), 325
AbstractTestObsolescence (class in *llama.test.test_filehandler*), 330
AbstractTestS190412mSkymapInfoCli (class in *llama.test.test_bin*), 329
AbstractTestSkymapInfoCli (class in *llama.test.test_bin*), 329
add () (*llama.versioning.GitHandler* method), 349
add_scatterplot () (in module *llama.files.healpix.plotters*), 198
Advok (class in *llama.files*), 143
Advok (class in *llama.files.advok*), 241
advok (*llama.listen.lvalert.Alert* property), 303
advreq (*llama.listen.lvalert.Alert* property), 303
Aeff () (in module *llama.files.coinc_significance.utils*), 192
aeff_lookup_table () (in module *llama.files.coinc_significance.utils*), 192
Aeff_skymap () (in module *llama.files.coinc_significance.utils*), 192
Alert (class in *llama.listen.lvalert*), 303
alert_file_handler (*llama.files.Advok* property), 143
alert_file_handler (*llama.files.advok.Advok* property), 242
alert_file_handler (*llama.files.skymap_info.SkymapInfo* property), 236
alert_file_handler (*llama.files.SkymapInfo* property), 177
alert_maintainers () (in module *llama.com.slack*), 113
alert_type (*llama.files.Advok* property), 143
alert_type (*llama.files.advok.Advok* property), 242
alert_type (*llama.listen.lvalert_json.LVAlertJSON* property), 282
alert_type (*llama.files.LVAlertJSON* property), 160
alert_type (*llama.files.lvc_gcn_xml.LvcGcnXml* property), 284
alert_type (*llama.files.LvcGcnXml* property), 162
alert_type (*llama.files.skymap_info.SkymapInfo* property), 236
alert_type (*llama.files.SkymapInfo* property), 177
ALLOWED_VALUES (*llama.flags.FlagDict* attribute), 293
always_veto () (in module *llama.files.ztf_trigger_list*), 292
analysis_kernel () (*llama.files.healpix.HEALPixSkyMapAnalysis* static method), 194
analyze_skymaps () (*llama.files.healpix.HEALPixSkyMapAnalysis* method), 194
ang2pix () (*llama.files.healpix.skymap.HEALPixSkyMap* method), 202
angle_list () (in module *llama.files.coinc_significance.utils*), 192
apply_default_cli_args () (in module *llama.dev.dv*), 122
archive_figs () (in module *llama.utils*), 338
are_dependencies_met () (*llama.filehandler.FileHandler* method), 135
are_ur_dependencies_met () (*llama.filehandler.FileHandler* method), 135
area_per_pixel (*llama.files.healpix.skymap.SkyMap* property), 204

argparse_to_dict () (in module *llama.dev.dv*), 123
argset_string () (in module *llama.dev.dv*), 123
argsort_end_of_cr () (in module *llama.files.lvc_skymap_mat*), 286
as_pdf (*llama.files.healpix.skymap.SkyMap* property), 204
assert_obsolete ()
 (*llama.test.test_filehandler.TestObsolescenceAndLocking* method), 333
authors (*llama.files.gcn_draft_o2.I3LvcGcnDraft* property), 273
auxiliary_paths (*llama.Event* property), 85
auxiliary_paths (*llama.event.Event* property), 127
auxiliary_paths (*llama.filehandler.FileHandler* property), 135
available_skymaps () (in module *llama.files.lvc_skymap*), 233
available_skymaps () (in module *llama.files.lvc_skymap.utils*), 235
available_versions () (in module *llama.dev.data.i3*), 120
azimuth (*llama.files.i3.Neutrino* property), 216

B

BACKGROUND_SKYMAP (*llama.files.coinc_plots.CoincScatterI3Lvc* attribute), 254
BACKGROUND_SKYMAP (*llama.files.coinc_plots.CoincScatterZtfI3Lvc* attribute), 256
BACKGROUND_SKYMAP (*llama.files.coinc_plots.CoincScatterZtfLVC* attribute), 258
BACKGROUND_SKYMAP (*llama.files.healpix.plotters.JointSkymapScatter* attribute), 198
base (*llama.intent.CoolDownParams* attribute), 299
base_filename (*llama.files.lvalert_json.LVAlertJSON* property), 283
base_filename (*llama.files.LVAlertJSON* property), 160
basename (*llama.files.lvc_skymap.utils.SkymapFilename* property), 234
bashlines () (in module *llama.com.gracedb*), 111
batch_error_alert_maintainers () (in module *llama.batch*), 93
batch_error_dump () (in module *llama.batch*), 93
best_neutrino
 (*llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson* property), 249
blinder () (in module *llama.files.i3.json*), 220
BLUE (*llama.classes.Colors* attribute), 95
blue (*llama.classes.Colors* attribute), 95
BOLD (*llama.classes.Colors* attribute), 95
bold (*llama.classes.Colors* attribute), 95
bucket (*llama.com.s3.PrivateFileCacherTuple* attribute), 112
bytes2str () (in module *llama.utils*), 338

C

cache_obsolescence () (in module *llama.lock*), 306
calc_cr_pull () (in module *llama.files.i3.realtime_tools_stubs*), 224
calculate () (*llama.files.healpix.Psf.Psf* method), 199
calculate () (*llama.files.healpix.Psf.PsfGaussian* method), 200
canonicalize () (*llama.files.lvc_skymap.utils.SkymapFilename* method), 234
CanonicalPathAction (class in *llama.cli*), 99

check() (*llama.vetoes.VetoHandler* method), 353
check_consistency() (*llama.Pipeline* method), 88
check_consistency() (*llama.pipeline.Pipeline* method), 312
check_filehandler_definition_consistency() (in module *llama.test.test_filehandler*), 334
check_heartbeat() (*llama.listen.lvalert.Client* method), 304
check_required_attributes() (in module *llama.test.test_filehandler*), 334
check_valid_nside() (in module *llama.files.healpix.utils*), 206
check_valid_nuniq() (in module *llama.files.healpix.utils*), 206
checksum() (*llama.filehandler.FileHandler* method), 135
checksum() (*llama.filehandler.JSONFile* method), 139
checksums (*llama.meta.MetaData* property), 307
chirp_mass (*llama.files.gracedb.LVCGraceDbEventData* property), 277
chirp_mass (*llama.files.LVCGraceDbEventData* property), 161
citations (*llama.detectors.DetectorTuple* attribute), 118
class_vetoes (*llama.com.utils.UploadReceipt* attribute), 115
class_vetoes (*llama.filehandler.mixins.ObservingVetoMixin* attribute), 140
class_vetoes (*llama.filehandler.mixins.OnlineVetoMixin* attribute), 141
class_vetoes
 (*llama.files.coinc_plots.RctSlkI3CoincSummaryI3LvcPdf*
 attribute), 264
class_vetoes
 (*llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPdf*
 attribute), 265
class_vetoes
 (*llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPng*
 attribute), 265
class_vetoes
 (*llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng*
 attribute), 267
class_vetoes
 (*llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng*
 attribute), 268
class_vetoes
 (*llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPdf*
 attribute), 268
class_vetoes
 (*llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPng*
 attribute), 268
class_vetoes
 (*llama.files.coinc_plots.RctSlkLmaCoincSummaryI3LvcPdf*
 attribute), 269
class_vetoes
 (*llama.files.coinc_significance.opa.RctSlkLmaCoincSignificance*
 attribute), 182
class_vetoes
 (*llama.files.coinc_significance.subthreshold.RctSlkLmaCoinc*
 attribute), 188
class_vetoes (*llama.files.gracedb.RctSlkLmaLVCGraceDbEventData* attribute), 278
class_vetoes (*llama.files.i3.json.RctSlkLmaIceCubeNeutrinoList* attribute), 220
class_vetoes (*llama.files.RctSlkI3CoincSummaryI3LvcPdf* attribute), 166
class_vetoes (*llama.files.RctSlkLmaCoincScatterI3LvcPdf* attribute), 167
class_vetoes (*llama.files.RctSlkLmaCoincScatterI3LvcPng* attribute), 167
class_vetoes (*llama.files.RctSlkLmaCoincScatterZtfI3LvcPdf* attribute), 168
class_vetoes (*llama.files.RctSlkLmaCoincScatterZtfI3LvcPng* attribute), 169
class_vetoes (*llama.files.RctSlkLmaCoincScatterZtfLVCPdf* attribute), 170
class_vetoes (*llama.files.RctSlkLmaCoincScatterZtfLVCPng* attribute), 170
class_vetoes (*llama.files.RctSlkLmaCoincScatterZtfLVCPng* attribute), 170
class_vetoes (*llama.files.RctSlkLmaCoincSignificanceI3Lvc* attribute), 171
class_vetoes (*llama.files.RctSlkLmaCoincSummaryI3LvcPdf* attribute), 172
class_vetoes (*llama.files.RctSlkLmaLVAlertJSON* attribute), 172
class_vetoes (*llama.files.RctSlkLmaLvcDistancesJson* attribute), 174
class_vetoes (*llama.files.RctSlkLmaLvcGcnXml* attribute), 175
class_vetoes (*llama.files.RctSlkLmaLVCGraceDbEventData* attribute), 174
class_vetoes (*llama.files.RctSlkLmaLvcRetractionXml* attribute), 176
class_vetoes (*llama.files.RctSlkLmaSkymapInfo* attribute), 176
class_vetoes (*llama.files.skymap_info.FarThresholdMixin* attribute), 235
class_vetoes (*llama.files.slack.RctSlkLmaLVAlertJSON* attribute), 239
class_vetoes (*llama.files.slack.RctSlkLmaLvcGcnXml* attribute), 239
class_vetoes (*llama.files.slack.RctSlkLmaLvcRetractionXml* attribute), 240
class_vetoes (*llama.files.slack.RctSlkLmaSkymapInfo* attribute), 241
class_vetoes (*llama.files.ztf_trigger_list.ZtfTriggerList* attribute), 292
class_vetoes (*llama.files.ZtfTriggerList* attribute), 179
class_vetoes (*llama.vetoes.VetoMixin* attribute), 354
CLASSNAME_FMT (*llama.com.utils.UploadReceipt* attribute), 114
CLASSNAME_FMT (*llama.files.gracedb.GraceDBReceipt* attribute), 275
CLASSNAME_FMT (*llama.files.gwastro.GWAstroReceipt* attribute), 279
CLASSNAME_FMT (*llama.files.slack.SlackReceiptIcecube* attribute), 241
CLASSNAME_FMT (*llama.files.slack.SlackReceiptLlama* attribute), 241
CLASSNAME_FMT (*llama.files.team_receipts.TeamEmailReceipt* attribute), 290
cleanup() (*llama.dev.dv.Command* method), 122
cleanup_testevents() (in module *llama.dev.clean*), 120
CLEAR (*llama.classes.Colors* attribute), 95
cli_args (*llama.test.test_bin.AbstractTestSkymapInfoCli* property), 329
cli_flags (*llama.test.test_bin.AbstractTestSkymapInfoCli* property), 329
Client (class in *llama.listen.lvalert*), 303
client() (in module *llama.com.slack*), 113
CliParser (class in *llama.cli*), 99
clobber (*llama.cli.Parsers* attribute), 100
clone() (*llama.Event* method), 85
clone() (*llama.event.Event* method), 127
close_stdin_stdout() (in module *llama.cli*), 102
clsname (*llama.classes.FileHandlerTuple* attribute), 96
cmdLine() (*llama.dev.dv.Command* method), 122
CoincAnalysis (class in *llama.files.coinc_o2*), 248
CoincAnalysisInitialIcecubeJson (class in *llama.files.coinc_o2*), 248
CoincLikelihoodPltsFrmI3 (class in *llama.files.coinc_analyses*), 242
CoincLikelihoodPltsFrmI3Lvc (class in *llama.files.coinc_analyses*), 243
CoincLikelihoodPltsFrmLvc (class in *llama.files.coinc_analyses*), 243
CoincLikelihoodPltsI3Lvc (class in *llama.files.coinc_analyses*), 244
CoincScatterI3Lvc (class in *llama.files.coinc_plots*), 254
CoincScatterI3LvcPdf (class in *llama.files*), 143
CoincScatterI3LvcPdf (class in *llama.files.coinc_plots*), 254
CoincScatterI3LvcPng (class in *llama.files*), 144
CoincScatterI3LvcPng (class in *llama.files.coinc_plots*), 255
CoincScatterZtfI3Lvc (class in *llama.files.coinc_plots*), 256
CoincScatterZtfI3LvcPdf (class in *llama.files*), 145
CoincScatterZtfI3LvcPdf (class in *llama.files.coinc_plots*), 256
CoincScatterZtfI3LvcPng (class in *llama.files*), 146

CoincScatterZtfI3LvcPng (class in `llama.files.coinc_plots`), 257
 CoincScatterZtfLVC (class in `llama.files.coinc_plots`), 258
 CoincScatterZtfLVCPdf (class in `llama.files`), 147
 CoincScatterZtfLVCPdf (class in `llama.files.coinc_plots`), 258
 CoincScatterZtfLVCPng (class in `llama.files`), 147
 CoincScatterZtfLVCPng (class in `llama.files.coinc_plots`), 258
 CoincSignificanceI3Lvc (class in `llama.files`), 148
 CoincSignificanceI3Lvc (class in `llama.files.coinc_significance.opa`), 180
 CoincSignificanceSubthresholdI3Lvc (class in `llama.files.coinc_significance.subthreshold`), 185
 CoincSkymapFrmI3HDF5 (class in `llama.files.coinc_analyses`), 245
 CoincSkymapFrmI3LvcHDF5 (class in `llama.files.coinc_analyses`), 245
 CoincSkymapFrmLvcHDF5 (class in `llama.files.coinc_analyses`), 246
 CoincSkymapI3LvcHDF5 (class in `llama.files.coinc_analyses`), 247
 CoincSkymapInitialIcecube (class in `llama.files.coinc_o2`), 249
 CoincSkymapO2Large (class in `llama.files.coinc_o2`), 250
 CoincSummaryI3LvcPdf (class in `llama.files`), 149
 CoincSummaryI3LvcPdf (class in `llama.files.coinc_plots`), 259
 CoincSummaryI3LvcTex (class in `llama.files`), 151
 CoincSummaryI3LvcTex (class in `llama.files.coinc_plots`), 261
 color (`llama.filehandler.Status` attribute), 139
 color_logger() (in module `llama.utils`), 338
 ColorFormatter (class in `llama.classes`), 95
 COLORS (`llama.utils.ColorFormatter` attribute), 337
 combined_p_value
 (`llama.files.coinc_significance.opa.CoincSignificanceI3Lvc` property), 181
 combined_p_value
 (`llama.files.coinc_significance.subthreshold.CoincSignificanceI3Lvc` property), 186
 combined_p_value (`llama.files.CoincSignificanceI3Lvc` property), 149
 Command (class in `llama.dev.dev`), 122
 comment (`llama.files.slack.SlackReceipt` property), 241
 commit_changes() (`llama.versioning.GitHandler` method), 349
 compare() (`llama.test.classes.AbstractFileGenerationChecker` method), 327
 compare() (`llama.test.classes.AbstractFileGenerationComparator` method), 327
 compare() (`llama.test.test_bin.AbstractTestSkymapInfoCli` method), 329
 compare_contents() (`llama.Event` method), 86
 compare_contents() (`llama.event.Event` method), 128
 compare_contents() (`llama.filehandler.FileHandler` method), 135
 conclusion (`llama.files.gcn_draft_o2.I3LvcGcnDraft` property), 273
 confidence_region_size() (`llama.files.healpix.skymap.SkyMap` method), 204
 conform_skymaps() (`llama.files.healpix.skymap.HEALPixSkyMap` method), 202
 convert_json_neutrinos_to_latex() (in module `llama.files.i3.tex`), 227
 convert_json_neutrinos_to_txt() (in module `llama.files.i3.txt`), 230
 CoolDown (class in `llama.intent`), 299
 COOLDOWN_PARAMS (`llama.filehandler.FileHandler` attribute), 135
 COOLDOWN_PARAMS (`llama.files.uw_summary.UwSummary` attribute), 291
 CoolDownParams (class in `llama.intent`), 299
 copy_file() (`llama.versioning.GitHandler` method), 349
 create_download_link() (in module `llama.serve.jupyter.utils`), 324
 create_droplets() (in module `llama.com.do`), 105
 creation_time() (`llama.Event` method), 86
 creation_time() (`llama.event.Event` method), 128
 cruff_files (`llama.Event` property), 86
 cruff_files (`llama.event.Event` property), 128
 current_hash (`llama.versioning.GitHandler` property), 350

D

dangle (`llama.files.healpix.psf.Psf` property), 199
 dangle() (`llama.files.healpix.skymap.HEALPixSkyMap` method), 202
 dangle() (`llama.files.healpix.skymap.SkyMap` method), 204
 dangle_rad() (in module `llama.files.healpix.utils`), 206
 dec (`llama.files.healpix.psf.Psf` property), 199
 dec (`llama.files.healpix.skymap.HEALPixSkyMap` property), 203
 dec (`llama.files.healpix.skymap.SkyMap` property), 204
 dec (`llama.files.i3.Neutrino` property), 216
 decorator_dict() (`llama.com.utils.UploadReceipt` class method), 115
 decorator_dict() (`llama.files.gracedb.GraceDBReceipt` class method), 275
 decorator_dict() (`llama.files.team_receipts.TeamEmailReceipt` class method), 290
 default_cmap() (in module `llama.files.healpix.plotters`), 198
 DEFAULT_FLAGS (`llama.flags.FlagDict` attribute), 293
 default_str (`llama.dev.dev.Command` property), 122
 defaults (`llama.dev.dev.Command` property), 122
 delete() (`llama.classes.RiderFile` method), 97
 delete() (`llama.filehandler.FileHandler` method), 135
 DEP_CHECKSUM_KWARGS (`llama.filehandler.FileHandler` attribute), 135
 dep_checksums (`llama.meta.MetaData` property), 307
 dep_checksums() (`llama.filehandler.FileHandler` method), 135
 dep_subset_checksums (`llama.meta.MetaData` property), 307
 DEPENDENCIES (`llama.filehandler.EventTriggeredFileHandler` attribute), 131
 DEPENDENCIES (`llama.filehandler.FileHandler` attribute), 135
 DEPENDENCIES (`llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3` attribute), 243
 DEPENDENCIES (`llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3Lvc` attribute), 243
 DEPENDENCIES (`llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3Lvc` attribute), 243
 DEPENDENCIES
 (`llama.files.coinc_analyses.CoincLikelihoodPltsFrmLvc` attribute), 243
 DEPENDENCIES (`llama.files.coinc_analyses.CoincLikelihoodPltsI3Lvc` attribute), 244
 DEPENDENCIES (`llama.files.coinc_analyses.CoincSkymapFrmI3HDF5` attribute), 245
 DEPENDENCIES
 (`llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDF5` attribute), 245
 DEPENDENCIES
 (`llama.files.coinc_analyses.CoincSkymapFrmLvcHDF5` attribute), 246
 DEPENDENCIES (`llama.files.coinc_analyses.CoincSkymapI3LvcHDF5` attribute), 248
 DEPENDENCIES (`llama.files.coinc_o2.CoincAnalysis` attribute), 248
 DEPENDENCIES (`llama.files.coinc_o2.CoincSkymapInitialIcecube` attribute), 249
 DEPENDENCIES (`llama.files.coinc_o2.RctGdbCoincSkymapO2Large` attribute), 251
 DEPENDENCIES
 (`llama.files.coinc_o2.RctGwaCoincAnalysisInitialIcecubeJson` attribute), 252
 DEPENDENCIES (`llama.files.coinc_o2.RctGwaCoincSkymapO2Large` attribute), 253
 DEPENDENCIES
 (`llama.files.coinc_o2.RctTeamCoincAnalysisInitialIcecubeJson` attribute), 253
 DEPENDENCIES (`llama.files.coinc_plots.CoincScatterI3LvcPdf` attribute), 254
 DEPENDENCIES (`llama.files.coinc_plots.CoincScatterI3LvcPng` attribute), 255
 DEPENDENCIES (`llama.files.coinc_plots.CoincScatterZtfI3LvcPdf` attribute), 256

DEPENDENCIES (*llama.files.coinc_plots.CoincScatterZtfI3LvcPng attribute*), 257
 DEPENDENCIES (*llama.files.coinc_plots.CoincScatterZtfLVCPdf attribute*), 258
 DEPENDENCIES (*llama.files.coinc_plots.CoincScatterZtfLVCPng attribute*), 259
 DEPENDENCIES (*llama.files.coinc_plots.CoincSummaryI3LvcPdf attribute*), 259
 DEPENDENCIES (*llama.files.coinc_plots.CoincSummaryI3LvcTex attribute*), 261
 DEPENDENCIES
 (*llama.files.coinc_plots.RctSlkI3CoincSummaryI3LvcPdf attribute*), 263
 DEPENDENCIES
 (*llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPfd attribute*), 264
 DEPENDENCIES
 (*llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPng attribute*), 265
 DEPENDENCIES
 (*llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPfd attribute*), 265
 DEPENDENCIES
 (*llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng attribute*), 266
 DEPENDENCIES
 (*llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPdf attribute*), 267
 DEPENDENCIES
 (*llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPng attribute*), 268
 DEPENDENCIES
 (*llama.files.coinc_plots.RctSlkLmaCoincSummaryI3LvcPfd attribute*), 268
 DEPENDENCIES
 (*llama.files.coinc_significance.opa.CoincSignificanceI3Lvc attribute*), 180
 DEPENDENCIES
 (*llama.files.coinc_significance.opa.RctSlkLmaCoincSignificanceI3Lvc attribute*), 182
 DEPENDENCIES
 (*llama.files.coinc_significance.subthreshold.CoincSignificanceI3Lvc attribute*), 185
 DEPENDENCIES
 (*llama.files.coinc_significance.subthreshold.RctSlkLmaCoincSignificanceI3Lvc attribute*), 187
 DEPENDENCIES (*llama.files.CoincScatterI3LvcPfd attribute*), 144
 DEPENDENCIES (*llama.files.CoincScatterI3LvcPng attribute*), 144
 DEPENDENCIES (*llama.files.CoincScatterZtfI3LvcPfd attribute*), 145
 DEPENDENCIES (*llama.files.CoincScatterZtfI3LvcPng attribute*), 146
 DEPENDENCIES (*llama.files.CoincScatterZtfLVCPdf attribute*), 147
 DEPENDENCIES (*llama.files.CoincScatterZtfLVCPng attribute*), 147
 DEPENDENCIES (*llama.files.CoincSignificanceI3Lvc attribute*), 148
 DEPENDENCIES (*llama.files.CoincSummaryI3LvcPfd attribute*), 149
 DEPENDENCIES (*llama.files.CoincSummaryI3LvcTex attribute*), 151
 DEPENDENCIES (*llama.files.fermi_grb.FermiGRBsJSON attribute*), 270
 DEPENDENCIES (*llama.files.FermiGRBsJSON attribute*), 154
 DEPENDENCIES (*llama.files.gcn_draft_o2.I3LvcGcnDraft attribute*), 272
 DEPENDENCIES (*llama.files.gcn_draft_o2.RctTeamI3LvcGcnDraft attribute*), 274
 DEPENDENCIES (*llama.files.gracedbGraceDBPollingFileHandler attribute*), 275
 DEPENDENCIES (*llama.files.gracedbLVCGraceDbEventData attribute*), 276
 DEPENDENCIES (*llama.files.gracedb.RctSlkLmaLVCGraceDbEventData attribute*), 278
 DEPENDENCIES (*llama.files.gwastro.RctGwaLVAlertJSON attribute*), 279
 DEPENDENCIES (*llama.files.gwastro.RctGwaLvcGcnXml attribute*), 280
 DEPENDENCIES (*llama.files.gwastro.RctGwaLvcRetractionXml attribute*), 280
 DEPENDENCIES (*llama.files.gwastro.RctGwaSkymapInfo attribute*), 281
 DEPENDENCIES (*llama.files.i3.IceCubeNeutrinoList attribute*), 211
 DEPENDENCIES (*llama.files.i3.IceCubeNeutrinoListCoincTxt attribute*), 212
 DEPENDENCIES (*llama.files.i3.IceCubeNeutrinoListTex attribute*), 214
 DEPENDENCIES (*llama.files.i3.IceCubeNeutrinoListTxt attribute*), 215
 DEPENDENCIES (*llama.files.i3.json.IceCubeNeutrinoList attribute*), 217
 DEPENDENCIES (*llama.files.i3.json.RctGdbIceCubeNeutrinoList attribute*), 218
 DEPENDENCIES (*llama.files.i3.json.RctSlkLmaIceCubeNeutrinoList attribute*), 219
 DEPENDENCIES (*llama.files.i3.tex.IceCubeNeutrinoListTex attribute*), 226
 DEPENDENCIES (*llama.files.i3.txt.IceCubeNeutrinoListCoincTxt attribute*), 227
 DEPENDENCIES (*llama.files.i3.txt.IceCubeNeutrinoListTxt attribute*), 228
 DEPENDENCIES (*llama.files.i3.txt.RctGwaIceCubeNeutrinoListTxt attribute*), 229
 DEPENDENCIES (*llama.files.IceCubeNeutrinoList attribute*), 155
 DEPENDENCIES (*llama.files.IceCubeNeutrinoListCoincTxt attribute*), 156
 DEPENDENCIES (*llama.files.IceCubeNeutrinoListTex attribute*), 157
 DEPENDENCIES (*llama.files.IceCubeNeutrinoListTxt attribute*), 158
 DEPENDENCIES (*llama.files.lvc_skymap.LvcDistancesJson attribute*), 232
 DEPENDENCIES (*llama.files.lvc_skymap.LvcSkymapFits attribute*), 233
 DEPENDENCIES (*llama.files.lvc_skymap.LvcSkymapHdf5 attribute*), 233
 DEPENDENCIES (*llama.files.lvc_skymap_mat.LvcSkymapMat attribute*), 286
 DEPENDENCIES (*llama.files.lvc_skymap_txt.LvcSkymapTxt attribute*), 287
 DEPENDENCIES (*llama.files.LvcDistancesJson attribute*), 161
 DEPENDENCIES (*llama.files.LVCGraceDbEventData attribute*), 160
 DEPENDENCIES (*llama.files.LvcSkymapFits attribute*), 163
 DEPENDENCIES (*llama.files.LvcSkymapHdf5 attribute*), 164
 DEPENDENCIES (*llama.files.RctSlkI3CoincSummaryI3LvcPfd attribute*), 165
 DEPENDENCIES (*llama.files.RctSlkLmaCoincScatterI3LvcPfd attribute*), 166
 DEPENDENCIES (*llama.files.RctSlkLmaCoincScatterI3LvcPng attribute*), 167
 DEPENDENCIES (*llama.files.RctSlkLmaCoincScatterZtfI3LvcPfd attribute*), 167
 DEPENDENCIES (*llama.files.RctSlkLmaCoincScatterZtfI3LvcPng attribute*), 168
 DEPENDENCIES (*llama.files.RctSlkLmaCoincScatterZtfLVCPdf attribute*), 169
 DEPENDENCIES (*llama.files.RctSlkLmaCoincScatterZtfLVCPng attribute*), 170
 DEPENDENCIES (*llama.files.RctSlkLmaCoincSignificanceI3Lvc attribute*), 170
 DEPENDENCIES (*llama.files.RctSlkLmaCoincSummaryI3LvcPfd attribute*), 171
 DEPENDENCIES (*llama.files.RctSlkLmaLVAlertJSON attribute*), 172
 DEPENDENCIES (*llama.files.RctSlkLmaLvcDistancesJson attribute*), 174
 DEPENDENCIES (*llama.files.RctSlkLmaLvcGcnXml attribute*), 174
 DEPENDENCIES (*llama.files.RctSlkLmaLVCGraceDbEventData attribute*), 172
 DEPENDENCIES (*llama.files.RctSlkLmaLvcRetractionXml attribute*), 175
 DEPENDENCIES (*llama.files.RctSlkLmaSkymapInfo attribute*), 176
 DEPENDENCIES (*llama.files.slack.RctSlkLmaLVAlertJSON attribute*), 238

DEPENDENCIES (*llama.files.slack.RctSlkLmaLvcGcnXml attribute*), 239
 DEPENDENCIES (*llama.files.slack.RctSlkLmaLvcRetractionXml attribute*), 239
 DEPENDENCIES (*llama.files.slack.RctSlkLmaSkymapInfo attribute*), 240
 DEPENDENCIES (*llama.files.sms_receipts.RctSMSAdvokJSON attribute*), 289
 DEPENDENCIES (*llama.files.timing_checks.TimingChecks attribute*), 290
 DEPENDENCIES (*llama.files.uw_summary.UwSummary attribute*), 291
 DEPENDENCIES (*llama.files.ztf_trigger_list.ZtfTriggerList attribute*), 291
 DEPENDENCIES (*llama.files.ZtfTriggerList attribute*), 178
 DEPENDENCIES (*llama.test.test_filehandler.MockFh1 attribute*), 331
 DEPENDENCIES (*llama.test.test_filehandler.MockFh2 attribute*), 331
 DEPENDENCIES (*llama.test.test_filehandler.MockFh3 attribute*), 331
 DEPENDENCIES (*llama.test.test_filehandler.MockFh4 attribute*), 332
 DEPENDENCIES (*llama.test.test_filehandler.MockFh5 attribute*), 333
 dependency_graph() (*llama.filehandler.FileGraph method*), 131
 dependency_graph() (*llama.Pipeline method*), 88
 dependency_graph() (*llama.pipeline.Pipeline method*), 312
 description (*llama.cli.RecursiveCli attribute*), 101
 description (*llama.detectors.DetectorTuple attribute*), 118
 description (*llama.files.gcn_draft_o2.I3LvcGcnDraft property*), 273
 destroy_droplets() (*in module llama.com.do*), 105
 Detector (*class in llama.detectors*), 117
 DETECTORS (*llama.filehandler.TriggerList attribute*), 140
 DETECTORS (*llama.files.coinc_analyses.CoincSkymapFrmI3HDF5 attribute*), 245
 DETECTORS (*llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDF5 attribute*), 246
 DETECTORS (*llama.files.coinc_analyses.CoincSkymapFrmLvcHDF5 attribute*), 247
 DETECTORS (*llama.files.coinc_analyses.CoincSkymapI3LvcHDF5 attribute*), 248
 DETECTORS (*llama.files.coinc_significance.opa.CoincSignificanceI3Lvc attribute*), 180
 DETECTORS
 (*llama.files.coinc_significance.subthreshold.CoincSignificanceSubthreshold attribute*), 185
 DETECTORS (*llama.files.CoincSignificanceI3Lvc attribute*), 149
 DETECTORS (*llama.files.fermi_grb.FermiGRBsJSON attribute*), 270
 DETECTORS (*llama.files.FermiGRBsJSON attribute*), 154
 DETECTORS (*llama.files.i3.IceCubeNeutrinoList attribute*), 211
 DETECTORS (*llama.files.i3.json.IceCubeNeutrinoList attribute*), 217
 DETECTORS (*llama.files.IceCubeNeutrinoList attribute*), 155
 DETECTORS (*llama.files.lvc_skymap.LvcSkymapFits attribute*), 233
 DETECTORS (*llama.files.lvc_skymap.LvcSkymapHdf5 attribute*), 233
 DETECTORS (*llama.files.LvcSkymapFits attribute*), 163
 DETECTORS (*llama.files.LvcSkymapHdf5 attribute*), 164
 DETECTORS (*llama.files.ztf_trigger_list.ZtfTriggerList attribute*), 291
 DETECTORS (*llama.files.ZtfTriggerList attribute*), 178
 DetectorTuple (*class in llama.detectors*), 118
 dev_mode (*llama.cli.Parsers attribute*), 100
 DictAction (*class in llama.batch*), 93
 diff() (*llama.versioning.GitHandler method*), 350
 diff_contents() (*llama.filehandler.FileHandler method*), 136
 distmean (*llama.files.lvc_skymap.LvcDistancesJson property*), 232
 distmean (*llama.files.LvcDistancesJson property*), 162
 diststd (*llama.files.lvc_skymap.LvcDistancesJson property*), 232
 diststd (*llama.files.LvcDistancesJson property*), 162
 doc_table_header() (*in module llama.detectors*), 118
 doctable_row() (*llama.detectors.TableRowRepresentable method*), 118
 download() (*in module llama.com.dl*), 105
 downres_mask_nest() (*in module llama.files.healpix.utils*), 206
 downselect() (*llama.filehandler.FileGraph method*), 132
 downselect() (*llama.files.i3.IceCubeNeutrinoList method*), 212
 downselect() (*llama.files.i3.json.IceCubeNeutrinoList method*), 218
 downselect() (*llama.files.IceCubeNeutrinoList method*), 155
 downselect() (*llama.Pipeline method*), 88
 downselect() (*llama.pipeline.Pipeline method*), 312
 downselect() (*llama.Run method*), 90
 downselect() (*llama.run.Run method*), 319
 downselect_events() (*in module llama.run*), 321
 downselect_pipeline() (*llama.Run method*), 91
 downselect_pipeline() (*llama.run.Run method*), 320
 downselection (*llama.run.RunTuple attribute*), 321
 droplet_header() (*in module llama.com.do*), 105
 droplet_row_fmt() (*in module llama.com.do*), 105

E

EmailReceipt (*class in llama.com.email*), 106
 emptyskymap() (*in module llama.files.coinc_significance.subthreshold*), 188
 end_of_neutrino_window() (*llama.files.i3.IceCubeNeutrinoList method*), 212
 end_of_neutrino_window()
 (*llama.files.i3.json.IceCubeNeutrinoList method*), 218
 end_of_neutrino_window() (*llama.files.IceCubeNeutrinoList method*), 155
 energy (*llama.files.i3.Neutrino property*), 216
 error() (*llama.cli.CliParser method*), 99
 etree (*llama.files.lvc_gcn_xml.LvcGcnXml property*), 284
 etree (*llama.files.LvcGcnXml property*), 162
 Event (*class in llama*), 85
 Event (*class in llama.event*), 127
 event (*llama.test.classes.AbstractFileGenerationComparator property*), 328
 event (*llama.test.test_filehandler.AbstractTestObsolescence property*), 330
 event() (*llama.com.gracedb.GraceDb method*), 107
 event_far_greater_than_10_per_day() (*in module llama.files.skymap_info*), 237
 event_time_gps (*llama.files.lvc_gcn_xml.LvcGcnXml property*), 284
 event_time_gps(*llama.files.LvcGcnXml property*), 162
 event_time_gps (*llama.files.skymap_info.SkymapInfo property*), 236
 event_time_gps (*llama.files.SkymapInfo property*), 177
 event_time_gps_nanoseconds (*llama.files.lvc_gcn_xml.LvcGcnXml property*), 284
 event_time_gps_nanoseconds (*llama.files.LvcGcnXml property*), 162
 event_time_gps_nanoseconds (*llama.files.skymap_info.SkymapInfo property*), 236
 event_time_gps_nanoseconds (*llama.files.SkymapInfo property*), 177
 event_time_gps_seconds (*llama.files.lvc_gcn_xml.LvcGcnXml property*), 284
 event_time_gps_seconds (*llama.files.LvcGcnXml property*), 162
 event_time_gps_seconds (*llama.files.skymap_info.SkymapInfo property*), 236
 event_time_gps_seconds (*llama.files.SkymapInfo property*), 177
 event_time_gps_seconds (*llama.files.SkymapInfo property*), 177
 event_time_mjd (*llama.files.lvc_gcn_xml.LvcGcnXml property*), 284
 event_time_mjd (*llama.files.LvcGcnXml property*), 162
 event_time_mjd (*llama.files.skymap_info.SkymapInfo property*), 236
 event_time_mjd (*llama.files.SkymapInfo property*), 177
 event_time_str (*llama.files.lvc_gcn_xml.LvcGcnXml property*), 284
 event_time_str (*llama.files.LvcGcnXml property*), 162
 event_time_str (*llama.files.skymap_info.SkymapInfo property*), 236
 event_time_str (*llama.files.SkymapInfo property*), 177
 eventdir (*llama.classes.ManifestTuple attribute*), 96
 eventdir (*llama.Event property*), 86
 eventdir (*llama.event.Event property*), 128
 eventdir (*llama.filehandler.FileHandler property*), 136
 eventdir (*llama.vetoes.VetoHandlerTuple attribute*), 354

eventdir_path_contains_string_injection() (in module `llama.vetoes`), 355
eventdir_path_contains_string_manual() (in module `llama.vetoes`), 355
eventdir_path_contains_string_scratch() (in module `llama.vetoes`), 355
eventdir_path_contains_string_test() (in module `llama.vetoes`), 355
eventfiltering (`llama.run.Parsers` property), 318
eventid (`llama.classes.FileHandlerTuple` attribute), 96
eventid (`llama.event.EventTuple` attribute), 129
eventid (`llama.filehandler.FileGraphTuple` attribute), 133
eventid (`llama.listen.lvalert.Alert` property), 303
EVENTID (`llama.test.classes.AbstractFileGenerationComparator` property), 327
EVENTID (`llama.test.classes.MS181101abMixin` attribute), 328
EVENTID (`llama.test.classes.S190412mMixin` attribute), 328
EVENTID (`llama.test.classes.S190425zMixin` attribute), 328
EVENTID (`llama.test.classes.S190521gMixin` attribute), 328
EVENTID (`llama.test.test_filehandler.AbstractTestObsolescence` attribute), 330
eventid (`llama.versioning.GitHandler` property), 350
eventid_is_not_real_event_graceid() (in module `llama.filehandler.mixins`), 141
eventid_is_not_valid_graceid() (in module `llama.filehandler.mixins`), 141
events (`llama.Run` property), 91
events (`llama.run.Run` property), 320
events() (`llama.com.gracedbGraceDb` method), 108
events_dtype() (in module `llama.files.i3.realtime_tools_stubs`), 224
EventTriggeredFileHandler (class in `llama.filehandler`), 131
EventTuple (class in `llama.event`), 129
execute() (`llama.test.classes.AbstractFileGenerationComparator` method), 328
execute() (`llama.test.test_bin.AbstractTestSkymapInfoCli` method), 329
execute() (`llama.test.test_listeners.test_gcn.AbstractGcndTester` method), 325
execute()
 (`llama.test.test_listeners.test_gcn.AbstractTestLlamaHandlers` method), 325
exists() (`llama.classes.RiderFile` method), 97
exists() (`llama.Event` method), 86
exists() (`llama.event.Event` method), 129
exists() (`llama.filehandler.FileHandler` method), 136
expnu() (in module `llama.files.coinc_significance.opa`), 182
expnu() (in module `llama.files.coinc_significance.subthreshold`), 188
extension (`llama.files.lvc_skymap.utils.SkymapFilename` property), 234

F

fA() (in module `llama.files.coinc_significance.opa`), 183
fA() (in module `llama.files.coinc_significance.subthreshold`), 188
far (`llama.files.lvalert_json.LVAlertJSON` property), 283
far (`llama.files.LVAlertJSON` property), 160
far (`llama.files.lvc_gcn_xml.LvcGcnXml` property), 284
far (`llama.files.LvcGcnXml` property), 162
far (`llama.files.skymap_info.SkymapInfo` property), 236
far (`llama.files.SkymapInfo` property), 177
FarThresholdMixin (class in `llama.files.skymap_info`), 235
FermiGRBsJSON (class in `llama.files.fermi_grb`), 153
FermiGRBsJSON (class in `llama.files.fermi_grb`), 269
fetch_archival_neutrinos() (in module `llama.dev.data.i3`), 121
fetch_recent() (in module `llama.poll.gracedb`), 315
file_handler_instances() (`llama.Pipeline` method), 89
file_handler_instances() (`llama.pipeline.Pipeline` method), 313
FILEEXT (`llama.filehandler.mixins.FileExtensionMixin` attribute), 140

FILEEXT (`llama.files.coinc_plots.CoincScatterI3LvcPdf` attribute), 254
FILEEXT (`llama.files.coinc_plots.CoincScatterI3LvcPng` attribute), 255
FILEEXT (`llama.files.coinc_plots.CoincScatterZtfI3LvcPdf` attribute), 256
FILEEXT (`llama.files.coinc_plots.CoincScatterZtfI3LvcPng` attribute), 257
FILEEXT (`llama.files.coinc_plots.CoincScatterZtfLVCPdf` attribute), 258
FILEEXT (`llama.files.coinc_plots.CoincScatterZtfLVCPng` attribute), 259
FILEEXT (`llama.files.CoincScatterI3LvcPdf` attribute), 144
FILEEXT (`llama.files.CoincScatterI3LvcPng` attribute), 145
FILEEXT (`llama.files.CoincScatterZtfI3LvcPdf` attribute), 145
FILEEXT (`llama.files.CoincScatterZtfI3LvcPng` attribute), 146
FILEEXT (`llama.files.CoincScatterZtfLVCPdf` attribute), 147
FILEEXT (`llama.files.CoincScatterZtfLVCPng` attribute), 147
FileExtensionMixin (class in `llama.filehandler.mixins`), 140
FileGraph (class in `llama.filehandler`), 131
FileGraphTuple (class in `llama.filehandler`), 133
FileHandler (class in `llama.filehandler`), 134
FileHandlerSubgraphAction (class in `llama.pipeline`), 309
FileHandlerTuple (class in `llama.classes`), 96
FILENAME (`llama.filehandler.FileHandler` attribute), 135
FILENAME (`llama.files.Advok` attribute), 143
FILENAME (`llama.files.advok.Advok` attribute), 242
FILENAME (`llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3` attribute), 243
FILENAME (`llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3Lvc` attribute), 243
FILENAME (`llama.files.coinc_analyses.CoincLikelihoodPltsFrmLvc` attribute), 244
FILENAME (`llama.files.coinc_analyses.CoincLikelihoodPltsI3Lvc` attribute), 244
FILENAME (`llama.files.coinc_analyses.CoincSkymapFrmI3HDF5` attribute), 245
FILENAME (`llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDF5` attribute), 246
FILENAME (`llama.files.coinc_analyses.CoincSkymapFrmLvcHDF5` attribute), 247
FILENAME (`llama.files.coinc_analyses.CoincSkymapI3LvcHDF5` attribute), 248
FILENAME (`llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson` attribute), 249
FILENAME (`llama.files.coinc_o2.CoincSkymapO2Large` attribute), 250
FILENAME (`llama.files.coinc_o2.HENlistONSOURCEIceCubeMat` attribute), 250
FILENAME (`llama.files.coinc_o2.IceCubeNeutrinoListMat` attribute), 251
FILENAME (`llama.files.coinc_o2.RctGdbCoincSkymapO2Large` attribute), 251
FILENAME
 (`llama.files.coinc_o2.RctGwaCoincAnalysisInitialIcecubeJson` attribute), 252
FILENAME (`llama.files.coinc_o2.RctGwaCoincSkymapO2Large` attribute), 253
FILENAME
 (`llama.files.coinc_o2.RctTeamCoincAnalysisInitialIcecubeJson` attribute), 253
FILENAME (`llama.files.coinc_plots.CoincScatterI3LvcPdf` attribute), 254
FILENAME (`llama.files.coinc_plots.CoincScatterI3LvcPng` attribute), 255
FILENAME (`llama.files.coinc_plots.CoincScatterZtfI3LvcPdf` attribute), 256
FILENAME (`llama.files.coinc_plots.CoincScatterZtfI3LvcPng` attribute), 257
FILENAME (`llama.files.coinc_plots.CoincScatterZtfLVCPdf` attribute), 258

FILENAME (*llama.files.coinc_plots.CoincScatterZtfLVCpng attribute*), 259
FILENAME (*llama.files.coinc_plots.CoincSummaryI3LvcPdf attribute*), 259
FILENAME (*llama.files.coinc_plots.CoincSummaryI3LvcTex attribute*), 262
FILENAME (*llama.files.coinc_plots.RctSlkI3CoincSummaryI3LvcPdf attribute*), 263
FILENAME (*llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPdf attribute*), 264
FILENAME (*llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPng attribute*), 265
FILENAME (*llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPdf attribute*), 266
FILENAME (*llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng attribute*), 266
FILENAME (*llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng attribute*), 267
FILENAME (*llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPng attribute*), 268
FILENAME (*llama.files.coinc_plots.RctSlkLmaCoincSummaryI3LvcPdf attribute*), 268
FILENAME (*llama.files.coinc_significance.opa.CoincSignificanceI3Lvc attribute*), 180
FILENAME (*llama.files.coinc_significance.opa.RctSlkLmaCoincSignificanceI3Lvc attribute*), 182
FILENAME (*llama.files.coinc_significance.subthreshold.CoincSignificanceI3Lvc attribute*), 185
FILENAME (*llama.files.coinc_significance.subthreshold.RctSlkLmaCoincSignificanceI3Lvc attribute*), 187
FILENAME (*llama.files.CoincScatterI3LvcPdf attribute*), 144
FILENAME (*llama.files.CoincScatterI3LvcPng attribute*), 145
FILENAME (*llama.files.CoincScatterZtfI3LvcPdf attribute*), 145
FILENAME (*llama.files.CoincScatterZtfI3LvcPng attribute*), 146
FILENAME (*llama.files.CoincScatterZtfLVCPdf attribute*), 147
FILENAME (*llama.files.CoincScatterZtfLVCPng attribute*), 147
FILENAME (*llama.files.CoincSignificanceI3Lvc attribute*), 149
FILENAME (*llama.files.CoincSummaryI3LvcPdf attribute*), 150
FILENAME (*llama.files.CoincSummaryI3LvcTex attribute*), 152
FILENAME (*llama.files.fermi_grb.FermiGRBsJSON attribute*), 270
FILENAME (*llama.files.FermiGRBsJSON attribute*), 154
FILENAME (*llama.files.gcn_draft_o2.I3LvcGcnDraft attribute*), 272
FILENAME (*llama.files.gcn_draft_o2.RctTeamI3LvcGcnDraft attribute*), 274
FILENAME (*llama.files.gracedb.LVCGraceDbEventData attribute*), 277
FILENAME (*llama.files.gracedb.PAstro attribute*), 277
FILENAME (*llama.files.gracedb.RctSlkLmaLVCGraceDbEventData attribute*), 278
FILENAME (*llama.files.gwastro.RctGwaLVAAlertJSON attribute*), 279
FILENAME (*llama.files.gwastro.RctGwaLvcGcnXml attribute*), 280
FILENAME (*llama.files.gwastro.RctGwaLvcRetractionXml attribute*), 280
FILENAME (*llama.files.gwastro.RctGwaSkymapInfo attribute*), 281
FILENAME (*llama.files.i3.IceCubeNeutrinoList attribute*), 211
FILENAME (*llama.files.i3.IceCubeNeutrinoListCointTxt attribute*), 213
FILENAME (*llama.files.i3.IceCubeNeutrinoListTex attribute*), 214
FILENAME (*llama.files.i3.IceCubeNeutrinoListTxt attribute*), 215
FILENAME (*llama.files.i3.json.IceCubeNeutrinoList attribute*), 217
FILENAME (*llama.files.i3.json.RctGdbIceCubeNeutrinoList attribute*), 219
FILENAME (*llama.files.i3.json.RctSlkLmaIceCubeNeutrinoList attribute*), 219
FILENAME (*llama.files.i3.tex.IceCubeNeutrinoListTex attribute*), 226
FILENAME (*llama.files.i3.txt.IceCubeNeutrinoListCointTxt attribute*), 227
FILENAME (*llama.files.i3.txt.IceCubeNeutrinoList attribute*), 229
FILENAME (*llama.files.i3.txt.RctGwaIceCubeNeutrinoListTxt attribute*), 229
FILENAME (*llama.files.IceCubeNeutrinoList attribute*), 155
FILENAME (*llama.files.IceCubeNeutrinoListCointTxt attribute*), 156
FILENAME (*llama.files.IceCubeNeutrinoListTex attribute*), 157
FILENAME (*llama.files.IceCubeNeutrinoListTxt attribute*), 159
FILENAME (*llama.files.lvalert_advok.LVAAlertAdvok attribute*), 282
FILENAME (*llama.files.lvalert_json.LVAAlertJSON attribute*), 282
FILENAME (*llama.files.LVAAlertAdvok attribute*), 159
FILENAME (*llama.files.LVAAlertJSON attribute*), 159
FILENAME (*llama.files.lvc_gcn_xml.LvcGcnXml attribute*), 283
FILENAME (*llama.files.lvc_gcn_xml.LvcRetractionXml attribute*), 284
FILENAME (*llama.files.lvc_skymap.LvcDistancesJson attribute*), 232
FILENAME (*llama.files.lvc_skymap.LvcSkymapFits attribute*), 233
FILENAME (*llama.files.lvc_skymap.LvcSkymapHdf5 attribute*), 233
filename (*llama.files.lvc_skymap.utils.SkymapFilename property*), 235
FILENAME (*llama.files.lvc_skymap_mat.LvcSkymapMat attribute*), 286
FILENAME (*llama.files.lvc_skymap_txt.LvcSkymapTxt attribute*), 287
FILENAME (*llama.files.LvcDistancesJson attribute*), 161
FILENAME (*llama.files.LvcGcnXml attribute*), 162
FILENAME (*llama.files.LVCGraceDbEventData attribute*), 160
FILENAME (*llama.files.LvcRetractionXml attribute*), 163
FILENAME (*llama.files.LvcSkymapFits attribute*), 163
FILENAME (*llama.files.LvcSkymapHdf5 attribute*), 164
FILENAME (*llama.files.PAstro attribute*), 164
FILENAME (*llama.files.RctSlkI3CoincSummaryI3LvcPdf attribute*), 165
FILENAME (*llama.files.RctSlkLmaCoincScatterI3LvcPdf attribute*), 166
FILENAME (*llama.files.RctSlkLmaCoincScatterI3LvcPng attribute*), 167
FILENAME (*llama.files.RctSlkLmaCoincScatterZtfI3LvcPdf attribute*),
FILENAME (*llama.files.RctSlkLmaCoincScatterZtfI3LvcPng attribute*), 168
FILENAME (*llama.files.RctSlkLmaCoincScatterZtfLVCPdf attribute*), 169
FILENAME (*llama.files.RctSlkLmaCoincScatterZtfLVCPng attribute*), 170
FILENAME (*llama.files.RctSlkLmaCoincSignificanceI3Lvc attribute*), 170
FILENAME (*llama.files.RctSlkLmaCoincSummaryI3LvcPdf attribute*), 171
FILENAME (*llama.files.RctSlkLmaLVAAlertJSON attribute*), 172
FILENAME (*llama.files.RctSlkLmaLvcDistancesJson attribute*), 174
FILENAME (*llama.files.RctSlkLmaLvcGcnXml attribute*), 174
FILENAME (*llama.files.RctSlkLmaLVCGraceDbEventData attribute*), 174
FILENAME (*llama.files.RctSlkLmaLvcRetractionXml attribute*), 176
FILENAME (*llama.files.RctSlkLmaSkymapInfo attribute*), 176
FILENAME (*llama.files.skymap_info.SkymapInfo attribute*), 235
FILENAME (*llama.files.SkymapInfo attribute*), 176
FILENAME (*llama.files.slack.RctSlkLmaLVAAlertJSON attribute*), 238
FILENAME (*llama.files.slack.RctSlkLmaLvcGcnXml attribute*), 239
FILENAME (*llama.files.slack.RctSlkLmaLvcRetractionXml attribute*), 239
FILENAME (*llama.files.slack.RctSlkLmaSkymapInfo attribute*), 240
FILENAME (*llama.files.sms_receipts.RctSMSAdvokJSON attribute*), 289
FILENAME (*llama.files.timing_checks.TimingChecks attribute*), 290
FILENAME (*llama.files.uw_summary.UwSummary attribute*), 291
FILENAME (*llama.files.ztf_trigger_list.ZtfTriggerList attribute*), 292
FILENAME (*llama.files.ZtfTriggerList attribute*), 179
FILENAME (*llama.test.test_filehandler.MockFh1 attribute*), 331
FILENAME (*llama.test.test_filehandler.MockFh2 attribute*), 331
FILENAME (*llama.test.test_filehandler.MockFh3 attribute*), 331
FILENAME (*llama.test.test_filehandler.MockFh4 attribute*), 332
FILENAME (*llama.test.test_filehandler.MockFh5 attribute*), 333
FILENAME_FMT (*llama.com.utils.UploadReceipt attribute*), 115

FILENAME_FMT
(llama.files.coinc_plots.RctSlkI3CoincSummaryI3LvcPdf attribute), 263
FILENAME_FMT
(llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPdf attribute), 265
FILENAME_FMT
(llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPng attribute), 265
FILENAME_FMT
(llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPdf attribute), 266
FILENAME_FMT
(llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng attribute), 266
FILENAME_FMT
(llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPdf attribute), 267
FILENAME_FMT
(llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPng attribute), 268
FILENAME_FMT
(llama.files.coinc_plots.RctSlkLmaCoincSummaryI3LvcPdf attribute), 268
FILENAME_FMT
(llama.files.coinc_significance.opa.RctSlkLmaCoincSignificance attribute), 182
FILENAME_FMT
(llama.files.coinc_significance.subthreshold.RctSlkLmaCoinc attribute), 187
FILENAME_FMT (*llama.files.gracedbGraceDBReceipt attribute*), 275
FILENAME_FMT (*llama.files.gracedb.RctSlkLmaLVCGraceDbEventData attribute*), 278
FILENAME_FMT (*llama.files.gwastro.GWAstroReceipt attribute*), 279
FILENAME_FMT (*llama.files.i3.json.RctSlkLmaIceCubeNeutrinoList attribute*), 219
FILENAME_FMT (*llama.files.RctSlkI3CoincSummaryI3LvcPdf attribute*), 165
FILENAME_FMT (*llama.files.RctSlkLmaCoincScatterI3LvcPdf attribute*), 167
FILENAME_FMT (*llama.files.RctSlkLmaCoincScatterI3LvcPng attribute*), 167
FILENAME_FMT (*llama.files.RctSlkLmaCoincScatterZtfI3LvcPdf attribute*), 168
FILENAME_FMT (*llama.files.RctSlkLmaCoincScatterZtfI3LvcPng attribute*), 168
FILENAME_FMT (*llama.files.RctSlkLmaCoincScatterZtfLVCPdf attribute*), 169
FILENAME_FMT (*llama.files.RctSlkLmaCoincScatterZtfLVCPng attribute*), 170
FILENAME_FMT (*llama.files.RctSlkLmaCoincSignificanceI3Lvc attribute*), 170
FILENAME_FMT (*llama.files.RctSlkLmaCoincSummaryI3LvcPdf attribute*), 171
FILENAME_FMT (*llama.files.RctSlkLmaLVAAlertJSON attribute*), 172
FILENAME_FMT (*llama.files.RctSlkLmaLvcDistancesJson attribute*), 174
FILENAME_FMT (*llama.files.RctSlkLmaLvcGcnXml attribute*), 174
FILENAME_FMT (*llama.files.RctSlkLmaLVCGraceDbEventData attribute*), 174
FILENAME_FMT (*llama.files.RctSlkLmaLvcRetractionXml attribute*), 176
FILENAME_FMT (*llama.files.RctSlkLmaSkymapInfo attribute*), 176
FILENAME_FMT (*llama.files.slack.RctSlkLmaLVAAlertJSON attribute*), 238
FILENAME_FMT (*llama.files.slack.RctSlkLmaLvcGcnXml attribute*), 239
FILENAME_FMT (*llama.files.slack.RctSlkLmaLvcRetractionXml attribute*), 239
FILENAME_FMT (*llama.files.slack.RctSlkLmaSkymapInfo attribute*), 240
FILENAME_FMT (*llama.files.sms_receipts.SMSReceipt attribute*), 289
FILENAME_FMT (*llama.files.team_receipts.TeamEmailReceipt attribute*), 290
filename_for_download (*llama.filehandler.FileHandler property*), 136
filename_for_download() (*llama.versioning.GitHandler method*), 350
FILENAME_PREFIX
(llama.files.coinc_analyses.CoincSkymapFrmI3HDF5 attribute), 245
FILENAME_PREFIX
(llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDF5 attribute), 246
FILENAME_PREFIX
(llama.files.coinc_analyses.CoincSkymapFrmLvcHDF5 attribute), 247
FILENAME_PREFIX
(llama.files.coinc_analyses.CoincSkymapI3LvcHDF5 attribute), 248
FILENAME_PREFIX (*llama.files.healpix.HEALPixSkyMapAnalysis attribute*), 194
filenames (*llama.classes.RiderFile property*), 97
files (*llama.Event property*), 87
files (*llama.event.Event property*), 129
files() (*llama.com.gracedbGraceDb method*), 108
find_in_submodules() (*in module llama.utils*), 338
finished() (*llama.run.RunVisualization method*), 321
five() (*in module llama.files.coinc_significance.opa*), 183
five() (*in module llama.files.coinc_significance.subthreshold*), 188
five2() (*in module llama.files.coinc_significance.opa*), 183
five2() (*in module llama.files.coinc_significance.subthreshold*), 189
five2wogw() (*in module llama.files.coinc_significance.subthreshold*), 189
fivewogw() (*in module llama.files.coinc_significance.subthreshold*), 190
flag_preset_freeze_veto() (*in module llama.listen*), 301
flag_table() (*in module llama.flags*), 294
flag_update_freeze_veto() (*in module llama.listen*), 301
FlagAction (*class in llama.flags.cli*), 294
FlagDict (*class in llama.flags*), 293
FlagPreset (*class in llama.flags*), 294
flags (*llama.flags.Parsers attribute*), 294
flags (*llama.flags.FlagsMixin property*), 294
flags (*llama.meta.MetaData property*), 307
FLAGS (*llama.test.test_bin.AbstractTestSkymapInfoCli attribute*), 329
FLAGS_PRESET (*llama.test.test_bin.AbstractTestSkymapInfoCli attribute*), 329
FLAGS_PRESET (*llama.test.test_bin.TestS190412mSkymapInfoCliPublic attribute*), 329
FLAGS_PRESET (*llama.test.test_bin.TestS190412mSkymapInfoCliTest attribute*), 330
FlagsMixin (*class in llama.flags*), 294
format() (*llama.utils.ColorFormatter method*), 337
format_neutrinos() (*in module llama.files.i3.json*), 220
from_module() (*llama.cli.RecursiveCli class method*), 101
from_module() (*llama.Pipeline class method*), 89
from_module() (*llama.pipeline.Pipeline class method*), 313
fromdir() (*llama.Event class method*), 87
fromdir() (*llama.event.Event class method*), 129
fromtimestamp() (*in module llama.filehandler*), 140
fullname (*llama.detectors.DetectorTuple attribute*), 118
fullpath (*llama.filehandler.FileHandler property*), 136
fullpaths (*llama.classes.RiderFile property*), 97

G

gcn_archive (*llama.listen.gcn.LlamaHandlersTuple attribute*), 302
gcn_archive
(llama.test.listeners.test_gcn.AbstractTestLlamaHandlers

property), 326
GcnInitialMixin (class in `llama.test.test_listeners.test_gcn`), 326
generate() (`llama.filehandler.FileHandler` method), 136
generate_from_gracedb() (`llama.files.skymap_info.SkymapInfo` method), 236
generate_from_gracedb() (`llama.files.SkymapInfo` method), 177
generate_from_gracedb_superevent() (`llama.files.skymap_info.SkymapInfo` method), 236
generate_from_gracedb_superevent() (`llama.files.SkymapInfo` method), 177
generate_from_lvc_gcn_xml() (`llama.files.skymap_info.SkymapInfo` method), 236
generate_from_lvc_gcn_xml() (`llama.files.SkymapInfo` method), 177
generated_before_all_neutrinos() (`llama.files.i3.IceCubeNeutrinoList` method), 212
generated_before_all_neutrinos() (`llama.files.i3.json.IceCubeNeutrinoList` method), 218
generated_before_all_neutrinos() (`llama.files.IceCubeNeutrinoList` method), 156
GenerateOnceMixin (class in `llama.filehandler`), 138
generation_time (`llama.meta.MetaData` property), 308
generation_uuid (`llama.meta.MetaData` property), 308
GenerationError, 337
get() (`llama.classes.ImmutableDict` method), 96
get() (`llama.com.s3.PrivateFileCacher` method), 112
get() (`llama.utils.RemoteFileCacher` method), 337
get_alert_type() (`llama.files.lvc_gcn_xml.LvcGcnXml` method), 284
get_alert_type() (`llama.files.LvcGcnXml` method), 163
get_archive() (in module `llama.files.i3.json`), 220
get_archive_and_real_neutrinos() (in module `llama.files.i3.json`), 221
get_client() (in module `llama.com.s3`), 112
get_client() (in module `llama.listen.lvalert`), 304
get_dec_filter() (in module `llama.files.i3.json`), 221
get_droplets() (in module `llama.dev.dv`), 123
get_epilog() (`llama.cli.RecursiveCli` method), 101
get_events() (in module `llama.files.i3.realtime_tools_stubs`), 224
get_filename_from_group() (in module `llama.files.lvc_gcn_xml`), 284
get_filename_from_group_name() (in module `llama.files.lvc_gcn_xml`), 285
get_filename_from_param() (in module `llama.files.lvc_gcn_xml`), 285
get_grbs_from_csv() (in module `llama.files.fermi_grb`), 270
get_grid() (in module `llama.utils`), 338
get_handle_lvc_gcn() (in module `llama.listen.gcn`), 302
get_healpix() (`llama.files.healpix.HEALPixSkyMapFileHandler` method), 194
get_healpix() (`llama.files.healpix.skymap.HEALPixPSF` method), 201
get_healpix() (`llama.files.healpix.skymap.HEALPixRepresentableFileHandler` method), 202
get_healpix_descriptions() (`llama.files.healpix.HEALPixSkyMapFileHandler` method), 195
get_healpix_descriptions() (`llama.files.healpix.skymap.HEALPixRepresentableFileHandler` method), 202
get_logging_cli() (in module `llama.cli`), 102
get_param() (`llama.files.lvc_gcn_xml.LvcGcnXml` method), 284
get_param() (`llama.files.LvcGcnXml` method), 163
get_parser() (in module `llama.dev.dv`), 123
get_parser() (`llama.cli.RecursiveCli` method), 101
get_postprocess_required_arg() (in module `llama.cli`), 102
get_ra_filter() (in module `llama.files.i3.json`), 222

get_real() (in module `llama.files.i3.json`), 223
get_skymap_filename() (in module `llama.files.lvc_gcn_xml`), 285
get_skymap_name() (in module `llama.poll.gracedb`), 315
get_ssh_keys() (in module `llama.com.do`), 105
get_subdir() (in module `llama.listen.gcn`), 302
get_tags() (in module `llama.com.do`), 105
get_test_file() (in module `llama.utils`), 339
get_users() (in module `llama.com.slack`), 114
get_voevent_gps_time() (in module `llama.utils`), 339
get_voevent_notice_time() (in module `llama.utils`), 339
get_voevent_param() (in module `llama.utils`), 339
get_voevent_role() (in module `llama.utils`), 339
get_voevent_time() (in module `llama.utils`), 339
getVOEventGPSNanoseconds() (in module `llama.utils`), 338
getVOEventGPSSeconds() (in module `llama.utils`), 338
git (`llama.versioning.GitDirMixin` property), 349
GitDirMixin (class in `llama.versioning`), 349
GitHandler (class in `llama.versioning`), 349
GitRepoUninitialized, 352
gps (`llama.files.i3.Neutrino` property), 216
gps2mjd() (in module `llama.utils`), 340
gps2utc() (in module `llama.utils`), 340
gpsnowO() (in module `llama.poll.gracedb`), 315
gpstime() (`llama.Event` method), 87
gpstime() (`llama.event.Event` method), 129
GraceDb (class in `llama.com.gracedb`), 106
gracedb_auth_wrapper() (in module `llama.com.gracedb`), 111
gracedb_human_file_url() (in module `llama.files.skymap_info.utils`), 238
gracedb_query() (in module `llama.poll.gracedb`), 315
gracedb_tags (`llama.files.gracedb.GraceDBReceipt` property), 276
gracedb_url (`llama.files.skymap_info.SkymapInfo` property), 237
gracedb_url (`llama.files.SkymapInfo` property), 177
gracedb_url() (in module `llama.files.skymap_info`), 237
GraceDPollingFileHandler (class in `llama.files.gracedb`), 275
GraceDBReceipt (class in `llama.files.gracedb`), 275
graceid (`llama.files.lvalert_json.LVALertJSON` property), 283
graceid (`llama.files.LVALertJSON` property), 160
graceid (`llama.files.lvc_gcn_xml.LvcGcnXml` property), 284
graceid (`llama.files.LvcGcnXml` property), 163
graceid (`llama.files.skymap_info.cli.Parsers` attribute), 237
graceid (`llama.files.skymap_info.SkymapInfo` property), 237
graceid (`llama.files.SkymapInfo` property), 178
graceid (`llama.listen.lvalert.Alert` property), 303
graph (`llama.filehandler.FileHandler` property), 137
GREEN (`llama.classes.Colors` attribute), 95
green (`llama.classes.Colors` attribute), 95
gui_url() (in module `llama.serve.gui`), 323
GW_DETECTORS (`llama.files.gracedb.LVCGraceDbEventData` attribute), 277

H
handle_archive() (`llama.listen.gcn.LlamaHandlers` method), 302
handle_lvc_initial_or_update (`llama.listen.gcn.LlamaHandlers` property), 302
handle_lvc_preliminary (`llama.listen.gcn.LlamaHandlers` property), 302
handle_lvc_retraction() (`llama.listen.gcn.LlamaHandlers` method), 302
handler (`llama.test.test_listeners.test_gcn.AbstractTestLlamaHandlers` property), 326
hashes() (`llama.versioning.GitHandler` method), 350
Hdf5Storage (class in `llama.classes`), 96
healpix_skymap_analysis_factory() (in module `llama.files.healpix`), 196

HEALPixPlots (*class in llama.files.healpix*), 194
HEALPixPSF (*class in llama.files.healpix.skymap*), 201
HEALPixRepresentableFileHandler (*class in llama.files.healpix.skymap*), 201
HEALPixSkyMap (*class in llama.files.healpix.skymap*), 202
HEALPixSkyMapAnalysis (*class in llama.files.healpix*), 194
HEALPixSkyMapFileHandler (*class in llama.files.healpix*), 194
HEALPixSkyMapStats (*class in llama.files.healpix*), 196
heartbeat_interval (*llama.listen.lvalert.Client attribute*), 304
heartbeat_message() (*llama.listen.lvalert.Client method*), 304
HEARTBEAT_NODE (*llama.listen.lvalert.Client attribute*), 304
heartbeat_queue (*llama.listen.lvalert.Client attribute*), 304
help (*llama.dev.dv.Command property*), 122
HENlistONSOURCEIceCubeMat (*class in llama.files.coinc_o2*), 250
html_table (*llama.filehandler.JSONFile property*), 139
human_url (*llama.files.skymap_info.SkymapInfo property*), 237
human_url (*llama.files.SkymapInfo property*), 178

I

I3LvcGcnDraft (*class in llama.files.gcn_draft_o2*), 271
iA() (*in module llama.files.coinc_significance.opa*), 184
iA() (*in module llama.files.coinc_significance.subthreshold*), 190
icecube_coords() (*in module llama.files.i3.utils*), 230
icecube_upload_flag_false() (*in module llama.files.slack*), 241
IceCubeNeutrinoList (*class in llama.files*), 154
IceCubeNeutrinoList (*class in llama.files.i3*), 211
IceCubeNeutrinoList (*class in llama.files.i3.json*), 217
IceCubeNeutrinoListCincTxt (*class in llama.files*), 156
IceCubeNeutrinoListCincTxt (*class in llama.files.i3*), 212
IceCubeNeutrinoListCincTxt (*class in llama.files.i3.txt*), 227
IceCubeNeutrinoListMat (*class in llama.files.coinc_o2*), 251
IceCubeNeutrinoListTex (*class in llama.files*), 157
IceCubeNeutrinoListTex (*class in llama.files.i3*), 214
IceCubeNeutrinoListTex (*class in llama.files.i3.tex*), 226
IceCubeNeutrinoListTxt (*class in llama.files*), 158
IceCubeNeutrinoListTxt (*class in llama.files.i3*), 215
IceCubeNeutrinoListTxt (*class in llama.files.i3.txt*), 228
ImmutableDict (*class in llama.classes*), 96
implemented_filehandler() (*in module llama.test.test_filehandler*), 334
in_progress() (*llama.intent.CoolDown method*), 299
in_progress() (*llama.intent.Intent method*), 300
included (*llama.listen.gcn.LlamaHandlers property*), 302
increment (*llama.intent.CoolDownParams attribute*), 299
indices (*llama.files.healpix.psf.Psf property*), 199
init() (*llama.Event method*), 87
init() (*llama.event.Event method*), 129
init() (*llama.versioning.GitHandler method*), 351
INPUT_RUN (*llama.test.classes.AbstractFileGenerationComparator attribute*), 327
inputevent (*llama.test.classes.AbstractFileGenerationComparator property*), 328
install_keytab() (*in module llama.com.gracedb*), 111
install_manifest() (*in module llama.install*), 297
instruments (*llama.files.gracedb.LVCGraceDbEventData property*), 277
instruments (*llama.files.LVCGraceDbEventData property*), 161
integrate() (*llama.files.healpix.skymap.SkyMap method*), 204
Intent (*class in llama.intent*), 299
introduction (*llama.files.gcn_draft_o2.I3LvcGcnDraft property*), 274
is_ancestor() (*llama.versioning.GitHandler method*), 351
is_clean() (*llama.versioning.GitHandler method*), 351
is_locked (*llama.lock.LockHandler property*), 305
is_obsolete() (*llama.filehandler.FileHandler method*), 137
is_obsolete() (*llama.filehandler.GenerateOnceMixin method*), 139
is_obsolete() (*llama.files.i3.IceCubeNeutrinoList method*), 212

isObsolete() (*llama.files.i3.json.IceCubeNeutrinoList method*), 218
isObsolete() (*llama.files.IceCubeNeutrinoList method*), 156
is_regular_graceid() (*in module llama.files.skymap_info*), 237
is_repo() (*llama.versioning.GitHandler method*), 351
is_super() (*llama.files.skymap_info.SkymapInfo property*), 237
is_super() (*llama.files.SkymapInfo property*), 178
is_super() (*in module llama.files.skymap_info.utils*), 238
is_superevent (*llama.listen.lvalert.Alert property*), 303
items() (*llama.classes.ImmutableDict method*), 96
items() (*llama.flags.FlagDict method*), 293
ivorn (*llama.files.lvc_gcn_xml.LvcGcnXml property*), 284
ivorn (*llama.files.LvcGcnXml property*), 163

J

JointSkymapScatter (*class in llama.files.healpix.plotters*), 198
json (*llama.listen.lvalert.Alert property*), 303
JSONFile (*class in llama.filehandler*), 139
JsonRiderMixin (*class in llama.classes*), 96

K

key (*llama.com.s3.PrivateFileCacherTuple attribute*), 112
keys() (*llama.classes.ImmutableDict method*), 96
keys() (*llama.flags.FlagDict method*), 293
keytab() (*in module llama.com.gracedb*), 111

L

latex_header() (*in module llama.files.i3.tex*), 227
latitude_limits (*llama.files.coinc_o2.CoincSkymapInitialIcecube attribute*), 249
listen() (*llama.listen.lvalert.Client method*), 304
llama
module, 85
llama.batch
module, 93
llama.classes
module, 95
llama.cli
module, 99
llama.com
module, 105
llama.com.dl
module, 105
llama.com.do
module, 105
llama.com.email
module, 106
llama.com.gracedb
module, 106
llama.com.s3
module, 112
llama.com.slack
module, 113
llama.com.utils
module, 114
llama.detectors
module, 117
llama.dev
module, 119
llama.dev.background
module, 119
llama.dev.background.pvalue
module, 119
llama.dev.background.table
module, 120
llama.dev.background.table_singles
module, 120

```

llama.dev.clean
    module, 120
llama.dev.data
    module, 120
llama.dev.data.i3
    module, 120
llama.dev.docs
    module, 121
llama.dev.docs.cli
    module, 121
llama.dev.dv
    module, 122
llama.dev.log
    module, 123
llama.dev.log.lvalert
    module, 123
llama.dev.upload
    module, 123
llama.event
    module, 127
llama.filehandler
    module, 131
llama.filehandler.mixins
    module, 140
llama.files
    module, 143
llama.files.advok
    module, 241
llama.files.coinc_analyses
    module, 242
llama.files.coinc_o2
    module, 248
llama.files.coinc_plots
    module, 254
llama.files.coinc_significance
    module, 179
llama.files.coinc_significance.opa
    module, 180
llama.files.coinc_significance.subthreshold
    module, 185
llama.files.coinc_significance.utils
    module, 192
llama.files.fermi_grb
    module, 269
llama.files.gcn_draft_o2
    module, 271
llama.files.gracedb
    module, 275
llama.files.gwastro
    module, 279
llama.files.healpix
    module, 194
llama.files.healpix.plotters
    module, 198
llama.files.healpix.psf
    module, 199
llama.files.healpix.skymap
    module, 201
llama.files.healpix.utils
    module, 206
llama.files.i3
    module, 211
llama.files.i3.json
    module, 217
llama.files.i3.realtime_tools_stubs
    module, 224
llama.files.i3.tex
    module, 226
llama.files.i3.txt
    module, 227
llama.files.i3.utils
    module, 230
llama.files.lvalert_advok
    module, 282
llama.files.lvalert_json
    module, 282
llama.files.lvc_gcn_xml
    module, 283
llama.files.lvc_skymap
    module, 232
llama.files.lvc_skymap.utils
    module, 234
llama.files.lvc_skymap_mat
    module, 286
llama.files.lvc_skymap_txt
    module, 287
llama.files.matlab
    module, 287
llama.files.skymap_info
    module, 235
llama.files.skymap_info.cli
    module, 237
llama.files.skymap_info.utils
    module, 238
llama.files.slack
    module, 238
llama.files.sms_receipts
    module, 288
llama.files.team_receipts
    module, 290
llama.files.timing_checks
    module, 290
llama.files.uw_summary
    module, 291
llama.files.ztf_trigger_list
    module, 291
llama.flags
    module, 293
llama.flags.cli
    module, 294
llama.install
    module, 297
llama.install.manifest
    module, 297
llama.intent
    module, 299
llama.listen
    module, 301
llama.listen.gcn
    module, 301
llama.listen.lvalert
    module, 303
llama.lock
    module, 305
llama.meta
    module, 307
llama.pipeline
    module, 309
llama.poll
    module, 315
llama.poll.gracedb
    module, 315
llama.run
    module, 317
llama.serve
    module, 323

```

```

llama.serve.gui
    module, 323
llama.serve.gui.domain
    module, 324
llama.serve.gui.wsgi
    module, 323
llama.serve.jupyter
    module, 324
llama.serve.jupyter.logs
    module, 324
llama.serve.jupyter.utils
    module, 324
llama.test
    module, 325
llama.test.classes
    module, 327
llama.test.test_bin
    module, 329
llama.test.test_filehandler
    module, 330
llama.test.test_files
    module, 325
llama.test.test_listeners
    module, 325
llama.test.test_listeners.test_gcn
    module, 325
llama.test.test_pipeline
    module, 334
llama.test.test_utils
    module, 334
llama.utils
    module, 337
llama.version
    module, 347
llama.versioning
    module, 349
llama.vetoes
    module, 353
LlamaHandlers (class in llama.listen.gcn), 301
LlamaHandlersTuple (class in llama.listen.gcn), 302
load_neutrino_background() (in module
    llama.files.coinc_significance.utils), 192
load_notebook() (in module llama.serve.jupyter), 324
load_params() (in module llama.batch), 93
load_spline() (in module llama.files.i3.realtime_tools_stubs), 225
localparser (llama.cli.RecursiveCli attribute), 101
localpath (llama.com.s3.PrivateFileCacherTuple attribute), 112
localpath (llama.utils.RemoteFileCacherTuple attribute), 337
lock (llama.lock.LockMixin property), 306
lock() (llama.lock.LockHandler method), 305
lockfiles (llama.lock.LockHandler property), 305
LockHandler (class in llama.lock), 305
LockMixin (class in llama.lock), 306
lockpaths (llama.lock.LockHandler property), 305
log_exceptions_and_recover() (in module llama.cli), 102
log_message (llama.files.coinc_o2.RctGdbCoincSkymapO2Large
    property), 252
log_message (llama.files.gracedb.GraceDBReceipt property), 276
log_message (llama.files.i3.json.RctGdbIceCubeNeutrinoList
    property), 219
loglevel_from_cli_count() (in module llama.cli), 103
logs() (llama.com.gracedb.GraceDb method), 109
longitude_limits (llama.files.coinc_o2.CoincSkymapInitialIcecube
    attribute), 250
LVALertAdvok (class in llama.files), 159
LVALertAdvok (class in llama.files.lvalert_advok), 282
LVALertJSON (class in llama.files), 159
LVALertJSON (class in llama.files.lvalert_json), 282
LvcDistancesJson (class in llama.files), 161
LvcDistancesJson (class in llama.files.lvc_skymap), 232
LvcGcnXml (class in llama.files), 162
LvcGcnXml (class in llama.files.lvc_gcn_xml), 283
LVCGraceDbEventData (class in llama.files), 160
LVCGraceDbEventData (class in llama.files.gracedb), 276
LvcHEALPixSkyMapFileHandler (class in llama.files.healpix), 196
LvcRetractionXml (class in llama.files), 163
LvcRetractionXml (class in llama.files.lvc_gcn_xml), 284
LvcSkymapFits (class in llama.files), 163
LvcSkymapFits (class in llama.files.lvc_skymap), 232
LvcSkymapHdf5 (class in llama.files), 164
LvcSkymapHdf5 (class in llama.files.lvc_skymap), 233
LvcSkymapMat (class in llama.files.lvc_skymap_mat), 286
LvcSkymapTxt (class in llama.files.lvc_skymap_txt), 287

M
MAGENTA (llama.classes.Colors attribute), 95
magenta (llama.classes.Colors attribute), 95
main() (in module llama.files.i3.txt), 230
main() (llama.cli.RecursiveCli method), 101
main() (llama.test.test_filehandler.AbstractTestObsolescence method),
    330
main() (llama.test.test_filehandler.TestGenerationOrder method), 333
main() (llama.test.test_filehandler.TestObsolescenceAndLocking
    method), 333
make_mock_filehandler() (in module llama.test.test_filehandler),
    334
manifest (llama.filehandler.FileHandler property), 137
manifest (llama.vetoes.VetoHandlerTuple attribute), 354
manifest_filehandlers (llama.classes.ManifestTuple attribute), 96
manifest_filehandlers (llama.filehandler.FileHandler property),
    137
MANIFEST_TYPES (llama.filehandler.FileHandler attribute), 135
MANIFEST_TYPES (llama.files.Advok attribute), 143
MANIFEST_TYPES (llama.files.advok.Advok attribute), 242
MANIFEST_TYPES
    (llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3
        attribute), 243
MANIFEST_TYPES
    (llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3Lvc
        attribute), 243
MANIFEST_TYPES
    (llama.files.coinc_analyses.CoincLikelihoodPltsFrmLvc
        attribute), 244
MANIFEST_TYPES
    (llama.files.coinc_analyses.CoincLikelihoodPltsI3Lvc
        attribute), 244
MANIFEST_TYPES
    (llama.files.coinc_analyses.CoincSkymapFrmI3HDF5
        attribute), 245
MANIFEST_TYPES
    (llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDFS
        attribute), 246
MANIFEST_TYPES
    (llama.files.coinc_analyses.CoincSkymapFrmLvcHDF5
        attribute), 247
MANIFEST_TYPES
    (llama.files.coinc_analyses.CoincSkymapI3LvcHDF5
        attribute), 248
MANIFEST_TYPES (llama.files.coinc_o2.CoincAnalysis attribute), 248
MANIFEST_TYPES
    (llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson
        attribute), 249
MANIFEST_TYPES (llama.files.coinc_o2.CoincSkymapO2Large
        attribute), 250

```

MANIFEST_TYPES (<i>llama.files.coinc_o2.HENlistONSOURCEIceCubeMat</i> attribute), 250	MANIFEST_TYPES (<i>llama.files.CoincScatterI3LvcPng</i> attribute), 145
MANIFEST_TYPES (<i>llama.files.coinc_o2.IceCubeNeutrinoListMat</i> attribute), 251	MANIFEST_TYPES (<i>llama.files.CoincScatterZtfI3LvcPdf</i> attribute), 145
MANIFEST_TYPES (<i>llama.files.coinc_o2.RctGdbCoincSkymapO2Large</i> attribute), 251	MANIFEST_TYPES (<i>llama.files.CoincScatterZtfI3LvcPng</i> attribute), 146
MANIFEST_TYPES (<i>llama.files.coinc_o2.RctGwaCoincAnalysisInitialIcecubeJson</i> attribute), 252	MANIFEST_TYPES (<i>llama.files.CoincScatterZtfLVCPdf</i> attribute), 147
MANIFEST_TYPES (<i>llama.files.coinc_o2.RctGwaCoincSkymapO2Large</i> attribute), 253	MANIFEST_TYPES (<i>llama.files.CoincScatterZtfLVCPng</i> attribute), 148
MANIFEST_TYPES (<i>llama.files.coinc_o2.RctTeamCoincAnalysisInitialIcecubeJson</i> attribute), 253	MANIFEST_TYPES (<i>llama.files.CoincSignificanceI3Lvc</i> attribute), 149
MANIFEST_TYPES (<i>llama.files.coinc_plots.CoincScatterI3LvcPdf</i> attribute), 254	MANIFEST_TYPES (<i>llama.files.CoincSummaryI3LvcPdf</i> attribute), 150
MANIFEST_TYPES (<i>llama.files.coinc_plots.CoincScatterI3LvcPng</i> attribute), 255	MANIFEST_TYPES (<i>llama.files.CoincSummaryI3LvcTex</i> attribute), 152
MANIFEST_TYPES (<i>llama.files.coinc_plots.CoincScatterZtfI3LvcPdf</i> attribute), 256	MANIFEST_TYPES (<i>llama.files.fermi_grb.FermiGRBsJSON</i> attribute), 270
MANIFEST_TYPES (<i>llama.files.coinc_plots.CoincScatterZtfI3LvcPng</i> attribute), 257	MANIFEST_TYPES (<i>llama.files.FermiGRBsJSON</i> attribute), 154
MANIFEST_TYPES (<i>llama.files.coinc_plots.CoincScatterZtfLVCPdf</i> attribute), 258	MANIFEST_TYPES (<i>llama.files.gcn_draft_o2.I3LvcGenDraft</i> attribute), 272
MANIFEST_TYPES (<i>llama.files.coinc_plots.CoincScatterZtfLVCPng</i> attribute), 259	MANIFEST_TYPES (<i>llama.files.gracedb.LVCGraceDbEventData</i> attribute), 277
MANIFEST_TYPES (<i>llama.files.coinc_plots.CoincSummaryI3LvcPdf</i> attribute), 259	MANIFEST_TYPES (<i>llama.files.gracedb.PAstro</i> attribute), 277
MANIFEST_TYPES (<i>llama.files.coinc_plots.CoincSummaryI3LvcTex</i> attribute), 262	MANIFEST_TYPES (<i>llama.files.gracedb.RctSlkLmaLVCGraceDbEventData</i> attribute), 278
MANIFEST_TYPES (<i>llama.files.coinc_plots.RctSlkI3CoincSummaryI3LvcPdf</i> attribute), 263	MANIFEST_TYPES (<i>llama.files.gwastro.RctGwaLVAlertJSON</i> attribute), 279
MANIFEST_TYPES (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPdf</i> attribute), 265	MANIFEST_TYPES (<i>llama.files.gwastro.RctGwaLvcGcnXml</i> attribute), 280
MANIFEST_TYPES (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPng</i> attribute), 265	MANIFEST_TYPES (<i>llama.files.gwastro.RctGwaLvcRetractionXml</i> attribute), 280
MANIFEST_TYPES (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPdf</i> attribute), 266	MANIFEST_TYPES (<i>llama.files.gwastro.RctGwaSkymapInfo</i> attribute), 281
MANIFEST_TYPES (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng</i> attribute), 266	MANIFEST_TYPES (<i>llama.files.i3.IceCubeNeutrinoList</i> attribute), 211
MANIFEST_TYPES (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPdf</i> attribute), 267	MANIFEST_TYPES (<i>llama.files.i3.IceCubeNeutrinoListCoincTxt</i> attribute), 213
MANIFEST_TYPES (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPng</i> attribute), 268	MANIFEST_TYPES (<i>llama.files.i3.IceCubeNeutrinoListTex</i> attribute), 214
MANIFEST_TYPES (<i>llama.files.coinc_plots.RctSlkLmaCoincSummaryI3LvcPdf</i> attribute), 268	MANIFEST_TYPES (<i>llama.files.i3.IceCubeNeutrinoListTxt</i> attribute), 215
MANIFEST_TYPES (<i>llama.files.coinc_significance.opa.CoincSignificanceI3Lvc</i> attribute), 180	MANIFEST_TYPES (<i>llama.files.i3.json.IceCubeNeutrinoList</i> attribute), 218
MANIFEST_TYPES (<i>llama.files.coinc_significance.opa.RctSlkLmaCoincSignificance</i> attribute), 182	MANIFEST_TYPES (<i>llama.files.i3.json.RctGdbIceCubeNeutrinoList</i> attribute), 219
MANIFEST_TYPES (<i>llama.files.coinc_significance.subthreshold.CoincSignificanceSubthreshold</i> attribute), 185	MANIFEST_TYPES (<i>llama.files.i3.json.RctSlkLmaIceCubeNeutrinoList</i> attribute), 219
MANIFEST_TYPES (<i>llama.files.coinc_significance.subthreshold.RctSlkLmaCoincSignificance</i> attribute), 188	MANIFEST_TYPES (<i>llama.files.i3.tex.IceCubeNeutrinoListTex</i> attribute), 226
MANIFEST_TYPES (<i>llama.files.CoincScatterI3LvcPdf</i> attribute), 144	MANIFEST_TYPES (<i>llama.files.i3.txt.IceCubeNeutrinoListCoincTxt</i> attribute), 227
	MANIFEST_TYPES (<i>llama.files.i3.txt.IceCubeNeutrinoListTxt</i> attribute), 229
	MANIFEST_TYPES (<i>llama.files.i3.txt.RctGwaIceCubeNeutrinoListTxt</i> attribute), 229
	MANIFEST_TYPES (<i>llama.files.IceCubeNeutrinoList</i> attribute), 155
	MANIFEST_TYPES (<i>llama.files.IceCubeNeutrinoListCoincTxt</i> attribute), 156
	MANIFEST_TYPES (<i>llama.files.IceCubeNeutrinoListTex</i> attribute), 157
	MANIFEST_TYPES (<i>llama.files.IceCubeNeutrinoListTxt</i> attribute), 159
	MANIFEST_TYPES (<i>llama.files.lvalert_advok.LVAlertAdvok</i> attribute), 282
	MANIFEST_TYPES (<i>llama.files.lvalert_json.LVAlertJSON</i> attribute), 282
	MANIFEST_TYPES (<i>llama.files.LVAlertAdvok</i> attribute), 159
	MANIFEST_TYPES (<i>llama.files.LVAlertJSON</i> attribute), 159
	MANIFEST_TYPES (<i>llama.files.lvc_gcn_xml.LvcGcnXml</i> attribute), 283
	MANIFEST_TYPES (<i>llama.files.lvc_skymap.LvcDistancesJson</i> attribute), 283
	MANIFEST_TYPES (<i>llama.files.lvc_skymap.LvcSkymapFits</i> attribute), 233
	MANIFEST_TYPES (<i>llama.files.lvc_skymap.LvcSkymapHdf5</i> attribute), 233

MANIFEST_TYPES (*llama.files.lvc_skymap_mat.LvcSkymapMat attribute*), 286
MANIFEST_TYPES (*llama.files.LvcDistancesJson attribute*), 162
MANIFEST_TYPES (*llama.files.LvcGcnXml attribute*), 162
MANIFEST_TYPES (*llama.files.LVCGraceDbEventData attribute*), 161
MANIFEST_TYPES (*llama.files.LvcSkymapFits attribute*), 163
MANIFEST_TYPES (*llama.files.LvcSkymapHdf5 attribute*), 164
MANIFEST_TYPES (*llama.files.PAstro attribute*), 164
MANIFEST_TYPES (*llama.files.RctSlkI3CoincSummaryI3LvcPdf attribute*), 165
MANIFEST_TYPES (*llama.files.RctSlkLmaCoincScatterI3LvcPdf attribute*), 167
MANIFEST_TYPES (*llama.files.RctSlkLmaCoincScatterI3LvcPng attribute*), 167
MANIFEST_TYPES (*llama.files.RctSlkLmaCoincScatterZtfI3LvcPdf attribute*), 168
MANIFEST_TYPES (*llama.files.RctSlkLmaCoincScatterZtfI3LvcPng attribute*), 168
MANIFEST_TYPES (*llama.files.RctSlkLmaCoincScatterZtfLVCPdf attribute*), 169
MANIFEST_TYPES (*llama.files.RctSlkLmaCoincScatterZtfLVCPng attribute*), 170
MANIFEST_TYPES (*llama.files.RctSlkLmaCoincSignificanceI3Lvc attribute*), 170
MANIFEST_TYPES (*llama.files.RctSlkLmaCoincSummaryI3LvcPdf attribute*), 171
MANIFEST_TYPES (*llama.files.RctSlkLmaLVAAlertJSON attribute*), 172
MANIFEST_TYPES (*llama.files.RctSlkLmaLvcDistancesJson attribute*), 174
MANIFEST_TYPES (*llama.files.RctSlkLmaLvcGcnXml attribute*), 174
MANIFEST_TYPES (*llama.files.RctSlkLmaLvcGraceDbEventData attribute*), 174
MANIFEST_TYPES (*llama.files.RctSlkLmaLvcRetractionXml attribute*), 176
MANIFEST_TYPES (*llama.files.RctSlkLmaSkymapInfo attribute*), 176
MANIFEST_TYPES (*llama.files.skymap_info.SkymapInfo attribute*), 236
MANIFEST_TYPES (*llama.files.SkymapInfo attribute*), 176
MANIFEST_TYPES (*llama.files.slack.RctSlkLmaLVAAlertJSON attribute*), 238
MANIFEST_TYPES (*llama.files.slack.RctSlkLmaLvcGcnXml attribute*), 239
MANIFEST_TYPES (*llama.files.slack.RctSlkLmaLvcRetractionXml attribute*), 239
MANIFEST_TYPES (*llama.files.slack.RctSlkLmaSkymapInfo attribute*), 240
MANIFEST_TYPES (*llama.files.sms_receipts.RctSMSAdvokJSON attribute*), 289
MANIFEST_TYPES (*llama.files.uw_summary.UwSummary attribute*), 291
MANIFEST_TYPES (*llama.files.ztf_trigger_list.ZtfTriggerList attribute*), 292
MANIFEST_TYPES (*llama.files.ZtfTriggerList attribute*), 179
MANIFEST_TYPES (*llama.test.test_filehandler.MockFh1 attribute*), 331
MANIFEST_TYPES (*llama.test.test_filehandler.MockFh2 attribute*), 331
MANIFEST_TYPES (*llama.test.test_filehandler.MockFh3 attribute*), 331
MANIFEST_TYPES (*llama.test.test_filehandler.MockFh4 attribute*), 332
MANIFEST_TYPES (*llama.test.test_filehandler.MockFh5 attribute*), 333
ManifestTuple (*class in llama.classes*), 96
manually_included (*llama.listen.gcn.LlamaHandlersTuple attribute*), 302
map() (*llama.files.healpix.skymap.SkyMap method*), 205
matlab_command (*llama.files.coinc_o2.CoincAnalysis property*), 249
matlab_command (*llama.files.coinc_o2.CoincSkymapInitialIcecube property*), 250
matlab_command (*llama.files.matlab.MATLABCallingFileHandler property*), 287
matlab_eval() (*in module llama.files.matlab*), 287
matlab_options (*llama.files.coinc_o2.CoincAnalysis attribute*), 249
matlab_options (*llama.files.matlab.MATLABCallingFileHandler attribute*), 287
MATLABCallingFileHandler (*class in llama.files.matlab*), 287
matlabpath() (*in module llama.files.matlab*), 288
max_nside (*llama.files.healpix.psf.Psf property*), 199
max_scale (*llama.files.healpix.psf.Psf property*), 199
max_scale (*llama.files.healpix.psf.PsfGaussian property*), 201
max_scale_indices (*llama.files.healpix.psf.Psf property*), 199
maximum (*llama.intent.CoolDownParams attribute*), 299
memoize() (*in module llama.utils*), 340
memoize_func_helper() (*in module llama.test.test_utils*), 334
memoize_helper() (*in module llama.test.test_utils*), 334
message_for_team (*llama.files.sms_receipts.RctSMSAdvokJSON property*), 289
message_for_team (*llama.files.sms_receipts.SMSReceipt property*), 289
MessageClient (*class in llama.files.sms_receipts*), 289
MetaClassFactory() (*in module llama.classes*), 96
MetaData (*class in llama.meta*), 307
min_nside (*llama.files.healpix.psf.Psf property*), 199
min_scale (*llama.files.healpix.psf.Psf property*), 199
min_scale (*llama.files.healpix.psf.PsfGaussian property*), 201
mjd (*llama.files.i3.Neutrino property*), 216
mjd2gps() (*in module llama.utils*), 340
mjd2utc() (*in module llama.utils*), 340
MockFh1 (*class in llama.test.test_filehandler*), 330
MockFh2 (*class in llama.test.test_filehandler*), 331
MockFh3 (*class in llama.test.test_filehandler*), 331
MockFh4 (*class in llama.test.test_filehandler*), 332
MockFh5 (*class in llama.test.test_filehandler*), 332
modification_time() (*llama.Event method*), 87
modification_time() (*llama.event.Event method*), 129
modtime() (*llama.filehandler.FileHandler method*), 138
module
 llama, 85
 llama.batch, 93
 llama.classes, 95
 llama.cli, 99
 llama.com, 105
 llama.com.dl, 105
 llama.com.do, 105
 llama.com.email, 106
 llama.com.gracedb, 106
 llama.com.s3, 112
 llama.com.slack, 113
 llama.com.utils, 114
 llama.detectors, 117
 llama.dev, 119
 llama.dev.background, 119
 llama.dev.background.pvalue, 119
 llama.dev.background.table, 120
 llama.dev.background.table_singles, 120
 llama.dev.clean, 120
 llama.dev.data, 120
 llama.dev.data.i3, 120
 llama.dev.docs, 121
 llama.dev.docs.cli, 121
 llama.dev.dv, 122
 llama.dev.log, 123
 llama.dev.log.lvalert, 123
 llama.dev.upload, 123
 llama.event, 127
 llama.filehandler, 131
 llama.filehandler.mixins, 140
 llama.files, 143
 llama.files.advok, 241
 llama.files.coinc_analyses, 242
 llama.files.coinc_o2, 248

llama.files.coinc_plots, 254
 llama.files.coinc_significance, 179
 llama.files.coinc_significance.opa, 180
 llama.files.coinc_significance.subthreshold, 185
 llama.files.coinc_significance.utils, 192
 llama.files.fermi_grb, 269
 llama.files.gcn_draft_o2, 271
 llama.files.gracedb, 275
 llama.files.gwastro, 279
 llama.files.healpix, 194
 llama.files.healpix.plotters, 198
 llama.files.healpix.psf, 199
 llama.files.healpix.skymap, 201
 llama.files.healpix.utils, 206
 llama.files.i3, 211
 llama.files.i3.json, 217
 llama.files.i3.realtime_tools_stubs, 224
 llama.files.i3.tex, 226
 llama.files.i3.txt, 227
 llama.files.i3.utils, 230
 llama.files.lvalert_advok, 282
 llama.files.lvalert_json, 282
 llama.files.lvc_gcn_xml, 283
 llama.files.lvc_skymap, 232
 llama.files.lvc_skymap.utils, 234
 llama.files.lvc_skymap_mat, 286
 llama.files.lvc_skymap_txt, 287
 llama.files.matlab, 287
 llama.files.skymap_info, 235
 llama.files.skymap_info.cli, 237
 llama.files.skymap_info.utils, 238
 llama.files.slack, 238
 llama.files.sms_receipts, 288
 llama.files.team_receipts, 290
 llama.files.timing_checks, 290
 llama.files.uw_summary, 291
 llama.files.ztf_trigger_list, 291
 llama.flags, 293
 llama.flags.cli, 294
 llama.install, 297
 llama.install.manifest, 297
 llama.intent, 299
 llama.listen, 301
 llama.listen.gcn, 301
 llama.listen.lvalert, 303
 llama.lock, 305
 llama.meta, 307
 llama.pipeline, 309
 llama.poll, 315
 llama.poll.gracedb, 315
 llama.run, 317
 llama.serve, 323
 llama.serve.gui, 323
 llama.serve.gui.domain, 324
 llama.serve.gui.wsgi, 323
 llama.serve.jupyter, 324
 llama.serve.jupyter.logs, 324
 llama.serve.jupyter.utils, 324
 llama.test, 325
 llama.test.classes, 327
 llama.test.test_bin, 329
 llama.test.test_filehandler, 330
 llama.test.test_files, 325
 llama.test.test_listeners, 325
 llama.test.test_listeners.test_gcn, 325
 llama.test.test_pipeline, 334
 llama.test.test_utils, 334
 llama.utils, 337

llama.version, 347
 llama.versioning, 349
 llama.vetoes, 353
 mollview() (*llama.files.healpix.plotters.JointSkymapScatter method*), 198
 mollview() (*llama.files.healpix.skymap.HEALPixSkyMap method*), 203
 mollview() (*llama.files.healpix.skymap.MollviewMixin method*), 204
 MollviewMixin (*class in llama.files.healpix.skymap*), 204
 most_likely_population (*llama.files.gracedb.PAstro property*), 277
 most_likely_population (*llama.files.PAstro property*), 164
 MS181101abMixin (*class in llama.test.classes*), 328

N

name (*llama.detectors.DetectorTuple attribute*), 118
 NamespaceMappable (*class in llama.classes*), 97
 nest2uniq() (*in module llama.files.healpix.utils*), 207
 Neutrino (*class in llama.files.i3*), 216
 neutrino_archive_available_years() (*in module llama.files.i3.json*), 224
 neutrino_table (*llama.files.gcn_draft_o2.I3LvcGcnDraft property*), 274
 neutrinos (*llama.files.i3.IceCubeNeutrinoList property*), 212
 neutrinos (*llama.files.i3.json.IceCubeNeutrinoList property*), 218
 neutrinos (*llama.files.IceCubeNeutrinoList property*), 156
 new_droplet() (*in module llama.com.do*), 105
 new_skymap (*llama.listen.lvalert.Alert property*), 303
 nodes (*llama.listen.lvalert.Client attribute*), 304
 NON_DETERMINISTIC_JSON_FIELDS
 (*llama.test.test_bin.AbstractTestSkymapInfoCli attribute*), 329
 normalize() (*llama.files.healpix.skymap.SkyMap method*), 205
 notice_time_str (*llama.files.lvalert_json.LVAlertJSON property*), 283
 notice_time_str (*llama.files.LVAlertJSON property*), 160
 notice_time_str (*llama.files.lvc_gcn_xml.LvcGcnXml property*), 284
 notice_time_str (*llama.files.LvcGcnXml property*), 163
 notice_time_str (*llama.files.skymap_info.SkymapInfo property*), 237
 notice_time_str (*llama.files.SkymapInfo property*), 178
 NOW() (*in module llama.event*), 130
 NOW() (*in module llama.filehandler*), 139
 npix (*llama.files.healpix.skymap.SkyMap property*), 205
 nside (*llama.files.healpix.skymap.HEALPixSkyMap property*), 203
 nside2ang() (*llama.files.healpix.skymap.HEALPixSkyMap class method*), 203
 num_triggers (*llama.filehandler.TriggerList property*), 140
 num_triggers (*llama.files.fermi_grb.FermiGRBs.JSON property*), 270
 num_triggers (*llama.files.FermiGRBs.JSON property*), 154
 num_triggers (*llama.files.healpix.HEALPixSkyMapFileHandler property*), 195
 num_triggers (*llama.files.i3.IceCubeNeutrinoList property*), 212
 num_triggers (*llama.files.i3.json.IceCubeNeutrinoList property*), 218
 num_triggers (*llama.files.IceCubeNeutrinoList property*), 156
 num_triggers (*llama.files.ztf_trigger_list.ZtfTriggerList property*), 292
 num_triggers (*llama.files.ZtfTriggerList property*), 179

O

ObservingVetoMixin (*class in llama.filehandler.mixins*), 140
 obsolescence (*llama.lock.LockHandler property*), 305
 obsolescence_files (*llama.lock.LockHandler property*), 305
 obsolescence_paths (*llama.lock.LockHandler property*), 305
 odds() (*llama.files.coinc_significance.opa.CoincSignificanceI3Lvc method*), 181
 odds() (*llama.files.CoincSignificanceI3Lvc method*), 149

odds_ratio() (in module `llama.files.coinc_significance.opa`), 184
odds_ratio() (in module `llama.files.coinc_significance.subthreshold`), 190
offline (`llama.files.gracedb.LVCGraceDbEventData` property), 277
offline (`llama.files.LVCGraceDbEventData` property), 161
online_flag_false() (in module `llama.filehandler.mixins`), 141
OnlineVetoMixin (class in `llama.filehandler.mixins`), 140
open() (`llama.filehandler.FileHandler` method), 138
optional_env_var() (in module `llama.classes`), 97
OptionalFeatureWarning, 97
ORGANIZATION (`llama.files.slack.SlackReceipt` attribute), 241
ORGANIZATION (`llama.files.slack.SlackReceiptIcecube` attribute), 241
ORGANIZATION (`llama.files.slack.SlackReceiptLlama` attribute), 241
outdir (`llama.cli.Parsers` attribute), 100
outfile (`llama.cli.Parsers` attribute), 100
outline_effect() (in module `llama.files.healpix.plotters`), 198

P

p_bbh (`llama.files.gracedb.PAstro` property), 277
p_bbh (`llama.files.PAstro` property), 165
p_bns (`llama.files.gracedb.PAstro` property), 277
p_bns (`llama.files.PAstro` property), 165
p_mass_gap (`llama.files.gracedb.PAstro` property), 278
p_mass_gap (`llama.files.PAstro` property), 165
p_nsbh (`llama.files.gracedb.PAstro` property), 278
p_nsbh (`llama.files.PAstro` property), 165
p_terr (`llama.files.gracedb.PAstro` property), 278
p_terr (`llama.files.PAstro` property), 165
p_value (`llama.files.uw_summary.UwSummary` property), 291
p_value() (in module `llama.files.coinc_significance.utils`), 192
p_values (`llama.files.coinc_significance.opa.CoincSignificanceI3Lvc` property), 181
p_values (`llama.files.CoincSignificanceI3Lvc` property), 149
parameter_factory() (in module `llama.utils`), 340
parent (`llama.filehandler.FileHandler` property), 138
parse_atom() (in module `llama.cli`), 103
parse_ivorn() (in module `llama.files.lvc_gcn_xml`), 285
parse_known_args() (`llama.cli.CliParser` method), 99
parse_neutrino_event_stream() (in module `llama.files.i3.realtime_tools_stubs`), 225
Parsers (class in `llama.cli`), 100
Parsers (class in `llama.files.skymap_info.cli`), 237
Parsers (class in `llama.flags.cli`), 294
Parsers (class in `llama.listen`), 301
Parsers (class in `llama.pipeline`), 309
Parsers (class in `llama.run`), 317
ParseRunsAction (class in `llama.run`), 317
past_runs() (in module `llama.run`), 321
PAstro (class in `llama.files`), 164
PAstro (class in `llama.files.gracedb`), 277
PEgw() (in module `llama.files.coinc_significance.opa`), 181
PEgw() (in module `llama.files.coinc_significance.subthreshold`), 186
Pempgw() (in module `llama.files.coinc_significance.utils`), 192
PEn() (in module `llama.files.coinc_significance.subthreshold`), 187
PEnu() (in module `llama.files.coinc_significance.opa`), 181
permanently_vetoed() (`llama.vetoes.VetoHandler` method), 353
pidfile() (in module `llama.cli`), 103
Pipeline (class in `llama`), 87
Pipeline (class in `llama.pipeline`), 309
pipeline (`llama.event.EventTuple` attribute), 130
pipeline (`llama.filehandler.FileGraphTuple` attribute), 134
pipeline (`llama.files.lvc_gcn_xml.LvcGcnXml` property), 284
pipeline (`llama.files.LvcGcnXml` property), 163
pipeline (`llama.files.skymap_info.SkymapInfo` property), 237
pipeline (`llama.files.SkymapInfo` property), 178
pipeline (`llama.pipeline.Parsers` attribute), 309
pipeline (`llama.run.RunTuple` attribute), 321

pipeline (`llama.test.classes.AbstractFileGenerationComparator` property), 328
pipeline (`llama.test.test_bin.AbstractTestSkymapInfoCli` property), 329
pipeline (`llama.test.test_listeners.test_gcn.AbstractTestLlamaHandlers` property), 326
pipeline (`llama.test.test_listeners.test_gcn.GcnInitialMixin` property), 326
PipelineAction (class in `llama.pipeline`), 313
pix_area (`llama.files.healpix.Psf` property), 200
pixel_radii_rad() (in module `llama.files.healpix`), 197
placeholderclass() (in module `llama.classes`), 98
plot_graphviz() (in module `llama.utils`), 341
plot_healpix() (in module `llama.files.healpix.plotters`), 199
POINT_SOURCES (`llama.files.coinc_plots.CoincScatterI3Lvc` attribute), 254
POINT_SOURCES (`llama.files.coinc_plots.CoincScatterZtfI3Lvc` attribute), 256
POINT_SOURCES (`llama.files.coinc_plots.CoincScatterZtfLVC` attribute), 258
POINT_SOURCES (`llama.files.healpix.plotters.JointSkymapScatter` attribute), 198
postprocess() (`llama.cli.CliParser` method), 99
postprocess_dev_mode() (in module `llama.cli`), 103
postprocess_downselect() (in module `llama.run`), 322
postprocess_dry_run() (in module `llama.run`), 322
postprocess_flags() (in module `llama.flags.cli`), 294
postprocess_flags_dry_run() (in module `llama.flags.cli`), 295
postprocess_logging() (in module `llama.cli`), 103
postprocess_param_iterator() (in module `llama.batch`), 93
postprocess_pipeline_dry_run() (in module `llama.pipeline`), 313
postprocess_pipeline_selection() (in module `llama.pipeline`), 313
postprocess_version() (in module `llama.cli`), 103
POSTPROCESSORS (`llama.cli.CliParser` attribute), 99
postscript (`llama.files.gcn_draft_o2.I3LvcGcnDraft` property), 274
preprocessor (`llama.cli.RecursiveCli` attribute), 102
PRESETS (`llama.flags.FlagDict` attribute), 293
print_defaults() (in module `llama.dev.dv`), 123
print_descriptions() (in module `llama.dev.dv`), 123
print_droplets() (in module `llama.com.do`), 105
print_help() (`llama.cli.CliParser` method), 100
print_help_if_no_cli_args() (`llama.cli.RecursiveCli` method), 102
print_running_procs_action() (in module `llama.cli`), 103
print_ssh_keys() (in module `llama.com.do`), 106
PrintDefaultPipeline (class in `llama.pipeline`), 313
PrintDownselectionsAction (class in `llama.run`), 318
PrintFlagsAction (class in `llama.flags.cli`), 294
printprocs() (in module `llama.cli`), 103
printstatus() (`llama.Event` method), 87
printstatus() (`llama.event.Event` method), 129
PrivateFileCacher (class in `llama.com.s3`), 112
PrivateFileCacherTuple (class in `llama.com.s3`), 112
proc_formatter() (in module `llama.cli`), 104
proc_printer() (in module `llama.cli`), 104
process_alert_json() (in module `llama.listen.lvalert`), 304
processor (`llama.listen.lvalert.Client` attribute), 304
production (`llama.listen.lvalert.Alert` property), 303
prog (`llama.cli.RecursiveCli` attribute), 102
provision() (`llama.test.classes.AbstractFileGenerationComparator` method), 328
Psf (class in `llama.files.healpix.psf`), 199
psf (`llama.files.i3.Neutrino` property), 216
psf_gaussian() (in module `llama.files.healpix.skymap`), 205
psf_gaussian() (`llama.files.healpix.skymap.HEALPixSkyMap` method), 203

`psf_gaussian_scales()` (in module `llama.files.healpix.skymap`), 205
`PsfGaussian` (class in `llama.files.healpix.psf`), 200
`pull_correct_online_2017()` (in module
`llama.files.i3.realtime_tools_stubs`), 225
`pull_correction()` (in module `llama.files.i3.realtime_tools_stubs`),
225
`put_file()` (in module `llama.batch`), 94

R

`r0()` (in module `llama.files.coinc_significance.utils`), 193
`ra` (`llama.files.healpix.Psf` property), 200
`ra` (`llama.files.healpix.skymap.HEALPixSkyMap` property), 204
`ra` (`llama.files.healpix.skymap.SkyMap` property), 205
`ra` (`llama.files.i3.Neutrino` property), 216
`ra_dec2zen_az()` (in module `llama.files.i3.utils`), 230
`random()` (in module `llama.files.i3.json`), 224
`RctGdbCoincSkymapO2Large` (class in `llama.files.coinc_o2`), 251
`RctGdbIceCubeNeutrinoList` (class in `llama.files.i3.json`), 218
`RctGwaCoincAnalysisInitialIcecubeJson` (class in
`llama.files.coinc_o2`), 252
`RctGwaCoincSkymapO2Large` (class in `llama.files.coinc_o2`), 253
`RctGwaIceCubeNeutrinoListTxt` (class in `llama.files.i3.txt`), 229
`RctGwaLVAAlertJSON` (class in `llama.files.gwastro`), 279
`RctGwaLvcGcnXml` (class in `llama.files.gwastro`), 280
`RctGwaLvcRetractionXml` (class in `llama.files.gwastro`), 280
`RctGwaSkymapInfo` (class in `llama.files.gwastro`), 281
`RctSlkI3CoincSummaryI3LvcPdf` (class in `llama.files`), 165
`RctSlkI3CoincSummaryI3LvcPdf` (class in `llama.files.coinc_plots`),
263
`RctSlkLmaCoincScatterI3LvcPdf` (class in `llama.files`), 166
`RctSlkLmaCoincScatterI3LvcPdf` (class in `llama.files.coinc_plots`),
264
`RctSlkLmaCoincScatterI3LvcPng` (class in `llama.files`), 167
`RctSlkLmaCoincScatterI3LvcPng` (class in `llama.files.coinc_plots`),
265
`RctSlkLmaCoincScatterZtfI3LvcPdf` (class in `llama.files`), 167
`RctSlkLmaCoincScatterZtfI3LvcPdf` (class in
`llama.files.coinc_plots`), 265
`RctSlkLmaCoincScatterZtfI3LvcPng` (class in `llama.files`), 168
`RctSlkLmaCoincScatterZtfI3LvcPng` (class in
`llama.files.coinc_plots`), 266
`RctSlkLmaCoincScatterZtfLVCPdf` (class in `llama.files`), 169
`RctSlkLmaCoincScatterZtfLVCPdf` (class in
`llama.files.coinc_plots`), 267
`RctSlkLmaCoincScatterZtfLVCPng` (class in `llama.files`), 170
`RctSlkLmaCoincScatterZtfLVCPng` (class in
`llama.files.coinc_plots`), 268
`RctSlkLmaCoincSignificanceI3Lvc` (class in `llama.files`), 170
`RctSlkLmaCoincSignificanceI3Lvc` (class in
`llama.files.coinc_significance.opa`), 181
`RctSlkLmaCoincSignificanceSubthresholdI3Lvc` (class in
`llama.files.coinc_significance.subthreshold`), 187
`RctSlkLmaCoincSummaryI3LvcPdf` (class in `llama.files`), 171
`RctSlkLmaCoincSummaryI3LvcPdf` (class in `llama.files.coinc_plots`),
268
`RctSlkLmaIceCubeNeutrinoList` (class in `llama.files.i3.json`), 219
`RctSlkLmaLVAAlertJSON` (class in `llama.files`), 172
`RctSlkLmaLVAAlertJSON` (class in `llama.files.slack`), 238
`RctSlkLmaLvcDistancesJson` (class in `llama.files`), 174
`RctSlkLmaLvcGcnXml` (class in `llama.files`), 174
`RctSlkLmaLvcGcnXml` (class in `llama.files.slack`), 239
`RctSlkLmaLVCGraceDbEventData` (class in `llama.files`), 172
`RctSlkLmaLVCGraceDbEventData` (class in `llama.files.gracedb`), 278
`RctSlkLmaLvcRetractionXml` (class in `llama.files`), 175
`RctSlkLmaLvcRetractionXml` (class in `llama.files.slack`), 239
`RctSlkLmaSkymapInfo` (class in `llama.files`), 176
`RctSlkLmaSkymapInfo` (class in `llama.files.slack`), 240

`RctSMSAdvokJSON` (class in `llama.files.sms_receipts`), 289
`RctTeamCoincAnalysisInitialIcecubeJson` (class in
`llama.files.coinc_o2`), 253
`RctTeamI3LvcGcnDraft` (class in `llama.files.gcn_draft_o2`), 274
`read_file()` (in module `llama.batch`), 94
`read_healpix_from_file()`
(`llama.files.healpix.HEALPixSkyMapFileHandler` static
method), 195
`read_healpix_from_file()`
(`llama.files.healpix.LvcHEALPixSkyMapFileHandler` static
method), 196
`read_json()` (`llama.classes.JsonRiderMixin` method), 96
`read_json()` (`llama.filehandler.JSONFile` method), 139
`read_json()` (`llama.vetoes.VetoHandler` method), 353
`read_partial_skymap()` (in module `llama.files.healpix.utils`), 207
`recipients` (`llama.com.email.EmailReceipt` property), 106
`recipients` (`llama.files.team_receipts.TeamEmailReceipt` property),
290
`reconstructed_neutrino_table`
(`llama.files.gcn_draft_o2.I3LvcGcnDraft` property), 274
`reconstructed_neutrinos`
(`llama.files.coinc_o2.CoincAnalysisInitialIcecubeJson`
property), 249
`reconstructed_neutrinos`
(`llama.files.gcn_draft_o2.I3LvcGcnDraft` property), 274
`record_obsobescence()` (`llama.lock.LockHandler` method), 305
`recursive_obsobescence()` (in module `llama.filehandler`), 140
`RecursiveCli` (class in `llama.cli`), 100
`RED` (`llama.classes.Colors` attribute), 95
`red` (`llama.classes.Colors` attribute), 95
`register_exit_handler()` (in module `llama.cli`), 104
`registerstub()` (in module `llama.classes`), 98
`remote_filename` (`llama.files.gracedb.GraceDBPollingFileHandler`
property), 275
`remote_filename` (`llama.files.gracedb.PAstro` property), 278
`remote_filename` (`llama.files.PAstro` property), 165
`REMOTE_RUNDIR` (`llama.files.gwastro.GWAstroReceipt` attribute), 279
`REMOTE_URL` (`llama.files.gwastro.GWAstroReceipt` attribute), 279
`REMOTE_USER` (`llama.files.gwastro.GWAstroReceipt` attribute), 279
`RemoteFileCacher` (class in `llama.utils`), 337
`RemoteFileCacherTuple` (class in `llama.utils`), 337
`remove()` (`llama.versioning.GitHandler` method), 351
`remove_all_obsobescence()` (`llama.lock.LockHandler` method),
305
`remove_obsobescence()` (`llama.lock.LockHandler` method), 305
`required_attributes()` (`llama.classes.RequiredAttributeMixin`
class method), 97
`RequiredAttributeMixin` (class in `llama.classes`), 97
`reset_hard()` (`llama.versioning.GitHandler` method), 351
`resolutions()` (in module `llama.files.healpix.psf`), 201
`rider_fmt` (`llama.classes.RiderFile` property), 97
`rider_fmt` (`llama.intent.CoolDown` attribute), 299
`rider_fmt` (`llama.intent.Intent` attribute), 300
`rider_fmt` (`llama.meta.MetaData` attribute), 308
`rider_mixin_factory()` (in module `llama.classes`), 98
`RiderFile` (class in `llama.classes`), 97
`role` (`llama.files.lvc_gcn_xml.LvcGcnXml` property), 284
`role` (`llama.files.LvcGcnXml` property), 163
`role_flag_not_observation()` (in module
`llama.filehandler.mixins`), 141
`rotate_angs2angs()` (in module `llama.utils`), 342
`rotate_angs2vec()` (in module `llama.utils`), 342
`RSYNC_DEST`
(`llama.files.coinc_o2.RctGwaCoincAnalysisInitialIcecubeJson`
attribute), 252
`RSYNC_DEST` (`llama.files.coinc_o2.RctGwaCoincSkymapO2Large`
attribute), 253
`RSYNC_DEST` (`llama.files.gwastro.RctGwaLVAAlertJSON` attribute), 279

RSYNC_DEST (*llama.files.gwastro.RctGwaLvcGcnXml attribute*), 280
RSYNC_DEST (*llama.files.gwastro.RctGwaLvcRetractionXml attribute*), 280
RSYNC_DEST (*llama.files.gwastro.RctGwaSkymapInfo attribute*), 281
RSYNC_DEST (*llama.files.i3.txt.RctGwaIceCubeNeutrinoListTxt attribute*), 229
rsync_source (*llama.files.gwastro.GWAstroReceipt property*), 279
Run (*class in llama*), 89
Run (*class in llama.run*), 318
run (*llama.test.classes.AbstractFileGenerationComparator property*), 328
run () (*llama.dev.dv.Command method*), 122
run_on_voevent_file() (*in module llama.listen.gcn*), 302
run_workers() (*in module llama.dev.dv*), 123
rundir (*llama.classes.FileHandlerTuple attribute*), 96
rundir (*llama.event.EventTuple attribute*), 130
rundir (*llama.filehandler.FileGraphTuple attribute*), 134
rundir (*llama.listen.gcn.LlamaHandlersTuple attribute*), 302
rundir (*llama.listen.Parsers attribute*), 301
rundir (*llama.run.RunTuple attribute*), 321
rundir (*llama.test.test_filehandler.AbstractTestObsolescence property*), 330
running_pids() (*in module llama.cli*), 104
running_servers() (*in module llama.serve.jupyter*), 324
RunTuple (*class in llama.run*), 321
RunVisualization (*class in llama.run*), 321

S

S190412mMixin (*class in llama.test.classes*), 328
S190425zMixin (*class in llama.test.classes*), 328
S190521gMixin (*class in llama.test.classes*), 328
safe_launch_daemon() (*in module llama.cli*), 104
save_tarball() (*llama.Event method*), 87
save_tarball() (*llama.event.Event method*), 129
scale2nside() (*in module llama.files.healpix*), 197
scripts() (*in module llama.dev.dv*), 123
search_parameters() (*in module llama.files.coinc_significance.utils*), 193
search_users() (*in module llama.com.slack*), 114
secure_transfer_to_s3() (*in module llama.dev.data.i3*), 121
send_as_attachment (*llama.com.email.EmailReceipt property*), 106
send_email() (*in module llama.utils*), 343
send_message() (*in module llama.com.slack*), 114
send_message() (*llama.files.sms_receipts.MessageClient method*), 289
send_message_to_team() (*llama.files.sms_receipts.SMSReceipt class method*), 289
serial_version() (*llama.versioning.GitHandler method*), 351
server (*llama.listen.lvalert.Client attribute*), 304
set_class_attributes() (*llama.com.utils.UploadReceipt class method*), 115
set_class_attributes() (*llama.filehandler.FileHandler class method*), 138
set_class_attributes() (*llama.files.gracedb.GraceDBReceipt class method*), 276
set_class_attributes() (*llama.files.gwastro.GWAstroReceipt class method*), 279
set_class_attributes() (*llama.files.healpix.HEALPixPlots class method*), 194
set_class_attributes()
 (*llama.files.healpix.HEALPixSkyMapAnalysis class method*), 194
set_class_attributes() (*llama.files.healpix.HEALPixSkyMapStats class method*), 196
set_class_attributes()
 (*llama.files.healpix.plotters.JointSkymapScatter class method*), 198

set_class_attributes() (*llama.files.slack.SlackReceipt class method*), 241
set_class_attributes() (*llama.files.sms_receipts.SMSReceipt class method*), 289
set_flags_oldevents() (*in module llama.dev.clean*), 120
setup() (*llama.dev.dv.Command method*), 122
setup_logger() (*in module llama.utils*), 343
setup_tail_widgets() (*in module llama.serve.jupyter.logs*), 324
show_log() (*llama.versioning.GitHandler method*), 351
sigma (*llama.files.i3.Neutrino property*), 216
SIGMA (*llama.files.ztf_trigger_list.ZtfTriggerList attribute*), 292
SIGMA (*llama.files.ZtfTriggerList attribute*), 179
size() (*llama.filehandler.FileHandler method*), 138
sizeof_fmt() (*in module llama.utils*), 344
sky_area() (*in module llama.files.healpix.utils*), 208
sky_area_deg() (*in module llama.files.healpix.utils*), 208
SkyMap (*class in llama.files.healpix.skymap*), 204
skymap (*llama.files.skymap_info.cli.Parsers attribute*), 237
skymap_filename (*llama.files.lvalert_json.LVAlertJSON property*), 283
skymap_filename (*llama.files.LVAlertJSON property*), 160
skymap_filename (*llama.files.lvc_gcn_xml.LvcGcnXml property*), 284
skymap_filename (*llama.files.LvcGcnXml property*), 163
skymap_filename (*llama.files.skymap_info.SkymapInfo property*), 237
skymap_filename (*llama.files.SkymapInfo property*), 178

SKYMAP_FILENAME

skymap_filenames() (*in module llama.files.lvc_skymap*), 233
skymap_filenames() (*in module llama.files.lvc_skymap.utils*), 235
skymap_product() (*llama.files.healpix.psf.Psf method*), 200
skymap_version (*llama.files.lvalert_json.LVAlertJSON property*), 283
skymap_version (*llama.files.LVAlertJSON property*), 160
SkymapFilename (*class in llama.files.lvc_skymap.utils*), 234
SkymapInfo (*class in llama.files*), 176
SkymapInfo (*class in llama.files.skymap_info*), 235
SkymapNotFoundError, 233
SlackReceipt (*class in llama.files.slack*), 241
SlackReceiptIcecube (*class in llama.files.slack*), 241
SlackReceiptLlama (*class in llama.files.slack*), 241

SLEEP_INTERVAL

(llama.test.test_listeners.test_gcn.AbstractGcndTester attribute), 325

SMSReceipt (*class in llama.files.sms_receipts*), 289

snr (*llama.files.gracedb.LVCGraceDbEventData property*), 277
snr (*llama.files.LVCGraceDbEventData property*), 161

sort_files_by_modification_time() (*in module llama.utils*), 344
sorted_by_generation_time()
 (*llama.test.test_filehandler.AbstractTestObsolescence method*), 330

source_locations() (*llama.files.fermi_grb.FermiGRBsJSON method*), 270
source_locations() (*llama.files.FermiGRBsJSON method*), 154
source_locations() (*llama.files.healpix.skymap.HEALPixPSF method*), 201
source_locations() (*llama.files.i3.IceCubeNeutrinoList method*), 212
source_locations() (*llama.files.i3.json.IceCubeNeutrinoList method*), 218
source_locations() (*llama.files.IceCubeNeutrinoList method*), 156
source_locations() (*llama.files.ztf_trigger_list.ZtfTriggerList method*), 292
source_locations() (*llama.files.ZtfTriggerList method*), 179

spawn_daemon() (*in module llama.cli*), 104

s
spec_flags (*llama.test.test_bin.AbstractTestSkymapInfoCli* property), 329
STARTING_MANIFEST
(llama.test.classes.AbstractFileGenerationComparator property), 327
STARTING_MANIFEST (*llama.test.test_bin.AbstractTestSkymapInfoCli* attribute), 329
STARTING_MANIFEST
(llama.test.test_listeners.test_gcn.AbstractTestLlamaHandlers attribute), 325
s
stats (*llama.filehandler.Status* attribute), 140
Status (class in *llama.filehandler*), 139
status (*llama.filehandler.Status* attribute), 140
status() (*llama.filehandler.FileGraph* method), 133
status() (*llama.filehandler.FileHandler* method), 138
subcommands (*llama.cli.RecursiveCli* attribute), 102
subgraph (*llama.filehandler.FileHandler* property), 138
subject (*llama.com.email.EmailReceipt* property), 106
subject
(llama.files.coinc_o2.RctTeamCoincAnalysisInitialIcecubeJson property), 254
subject (*llama.files.gcn_draft_o2.I3LvcGcnDraft* property), 274
subject (*llama.files.gcn_draft_o2.RctTeamI3LvcGcnDraft* property), 275
subject (*llama.files.team_receipts.TeamEmailReceipt* property), 290
subpixels() (in module *llama.files.healpix*), 197
summary (*llama.detectors.DetectorTuple* attribute), 118
superevent() (*llama.com.gracedb.GraceDb* method), 109
superevents() (in module *llama.poll.gracedb*), 315
superevents() (*llama.com.gracedb.GraceDb* method), 110
superid (*llama.listen.lvalert.Alert* property), 303
sync_shared_manifests() (*llama.filehandler.FileHandler* class method), 138

T

t_overlap() (in module *llama.files.coinc_significance.utils*), 193
TABLE_COLUMN_WIDTHS (*llama.detectors.TableRowRepresentable* attribute), 118
table_legend (*llama.files.gcn_draft_o2.I3LvcGcnDraft* property), 274
TableRowRepresentable (class in *llama.detectors*), 118
tag_users() (in module *llama.com.slack*), 114
tbbhighlight() (in module *llama.utils*), 344
TeamEmailReceipt (class in *llama.files.team_receipts*), 290
template_skymap_filehandler
(llama.files.fermi_grb.FermiGRBsJSON property), 270
template_skymap_filehandler (*llama.files.FermiGRBsJSON* property), 154
template_skymap_filehandler
(llama.files.healpix.skymap.HEALPixPSF property), 201
template_skymap_filehandler (*llama.files.i3.IceCubeNeutrinoList* property), 212
template_skymap_filehandler
(llama.files.i3.json.IceCubeNeutrinoList property), 218
template_skymap_filehandler (*llama.files.IceCubeNeutrinoList* property), 156
template_skymap_filehandler
(llama.files.ztf_trigger_list.ZtfTriggerList property), 292
template_skymap_filehandler (*llama.files.ZtfTriggerList* property), 179
temporally_coincident_neutrinos
(llama.files.gcn_draft_o2.I3LvcGcnDraft property), 274
test() (*llama.test.classes.AbstractFileGenerationComparator* method), 328
test() (*llama.test.test_filehandler*.*AbstractTestObsolescence* method), 330

t
test() (*llama.test.test_listeners.test_gcn*.*AbstractTestLlamaHandlers* method), 326
test_default_pipeline_consistency() (in module *llama.test.test_pipeline*), 334
test_filehandler_definition_consistency() (in module *llama.test.test_filehandler*), 334
test_get_grid_make_grid() (in module *llama.test.test_utils*), 334
test_get_grid_north_pole() (in module *llama.test.test_utils*), 334
test_llama_pipeline_parser() (in module *llama.test.test_bin*), 330
test_llama_run_parser() (in module *llama.test.test_bin*), 330
test_memoize() (in module *llama.test.test_utils*), 335
test_memoize_class() (in module *llama.test.test_utils*), 335
test_memoize_helper() (in module *llama.test.test_utils*), 335
test_mjd_interval() (in module *llama.test.test_bin*), 330
test_rotate_angs2vec() (in module *llama.test.test_utils*), 335
test_write_gzip() (in module *llama.test.test_utils*), 335
TestGcndS190425z (class in *llama.test.test_listeners.test_gcn*), 326
TestGcndS190521g (class in *llama.test.test_listeners.test_gcn*), 326
TestGenerationOrder (class in *llama.test.test_filehandler*), 333
TestLlamaHandlerMS181101ab (class in *llama.test.test_listeners.test_gcn*), 326
TestLlamaHandlerS190425z (class in *llama.test.test_listeners.test_gcn*), 326
TestLlamaHandlerS190521g (class in *llama.test.test_listeners.test_gcn*), 326
TestObsolescenceAndLocking (class in *llama.test.test_filehandler*), 333
TestS190412mSkymapInfoCliPublic (class in *llama.test.test_bin*), 329
TestS190412mSkymapInfoCliTest (class in *llama.test.test_bin*), 329
text_graph() (*llama.versioning.GitHandler* method), 351
time (*llama.files.Advok* property), 143
time (*llama.files.advok.Advok* property), 242
TIMEOUT (*llama.filehandler.FileHandler* attribute), 135
TIMEOUT (*llama.files.coinc_significance.opa.CoincSignificanceI3Lvc* attribute), 180
TIMEOUT
(llama.files.coinc_significance.subthreshold.CoincSignificanceSubthresholdI3Lvc attribute), 185
TIMEOUT (*llama.files.CoincSignificanceI3Lvc* attribute), 149
TIMEOUT (*llama.test.test_listeners.test_gcn*.*AbstractGcndTester* attribute), 325
TimingChecks (class in *llama.files.timing_checks*), 290
to_datestring() (in module *llama.files.i3.realtime_tools_stubs*), 225
to_nest() (*llama.files.healpix.skymap.HEALPixSkyMap* method), 204
to_ring() (*llama.files.healpix.skymap.HEALPixSkyMap* method), 204
total_mass (*llama.files.gracedb.LVCGraceDbEventData* property), 277
total_mass (*llama.files.LVCGraceDbEventData* property), 161
traceback_alert_maintainers() (in module *llama.cli*), 104
trash_obsolescence() (in module *llama.lock*), 306
TriggerList (class in *llama.filehandler*), 140
triggers (*llama.files.ztf_trigger_list.ZtfTriggerList* property), 292
triggers (*llama.files.ZtfTriggerList* property), 179

U

underline (*llama.classes.Colors* attribute), 95
underline (*llama.classes.Colors* attribute), 95
uninstall() (in module *llama.com.gracedb*), 111
uniq2nest_and_nsides() (in module *llama.files.healpix.utils*), 208
uniq2nsides() (in module *llama.files.healpix.utils*), 208
uniq2uniq_lowres() (in module *llama.files.healpix.utils*), 208
uniq_intersection() (in module *llama.files.healpix.utils*), 209
uniq_rasterize() (in module *llama.files.healpix.utils*), 209
unit (*llama.files.healpix.skymap.SkyMap* property), 205
unlock() (*llama.lock.LockHandler* method), 305

unveto() (*llama.vetoes.VetoHandler method*), 353
update() (*llama.Event method*), 87
update() (*llama.event.Event method*), 129
update() (*llama.filehandler.FileGraph method*), 133
update() (*llama.flags.FlagDict method*), 293
update() (*llama.Run method*), 91
update() (*llama.run.Run method*), 321
UPLOAD (*llama.com.utils.UploadReceipt attribute*), 115
UPLOAD (*llama.files.coinc_o2.RctGdbCoincSkymapO2Large attribute*), 251
UPLOAD (*llama.files.coinc_o2.RctGwaCoi...InitialIcecubeJson attribute*), 252
UPLOAD (*llama.files.coinc_o2.RctGwaCoi...Large attribute*), 253
UPLOAD (*llama.files.coinc_o2.RctTeamCoi...InitialIcecubeJson attribute*), 253
UPLOAD (*llama.files.coinc_plots.RctSlkI3Coi...SummaryI3LvcPdf attribute*), 263
UPLOAD (*llama.files.coinc_plots.RctSlkLmaCoi...ScatterI3LvcPdf attribute*), 265
UPLOAD (*llama.files.coinc_plots.RctSlkLmaCoi...ScatterI3LvcPng attribute*), 265
UPLOAD (*llama.files.coinc_plots.RctSlkLmaCoi...ScatterZtfI3LvcPdf attribute*), 266
UPLOAD (*llama.files.coinc_plots.RctSlkLmaCoi...ScatterZtfI3LvcPng attribute*), 266
UPLOAD (*llama.files.coinc_plots.RctSlkLmaCoi...ScatterZtfLVCPdf attribute*), 267
UPLOAD (*llama.files.coinc_plots.RctSlkLmaCoi...ScatterZtfLVCPng attribute*), 268
UPLOAD (*llama.files.coinc_plots.RctSlkLmaCoi...SummaryI3LvcPdf attribute*), 269
UPLOAD
(llama.files.coinc_significance.opa.RctSlkLmaCoi...SignificanceI3Lvc attribute), 182
UPLOAD
(llama.files.coinc_significance.subthreshold.RctSlkLmaCoi...SignificanceI3Lvc attribute), 188
UPLOAD (*llama.files.gcn_draft_o2.RctTeamI3LvcGcnDraft attribute*), 274
UPLOAD (*llama.files.gracedb.RctSlkLmaLVCGraceDbEventData attribute*), 278
UPLOAD (*llama.files.gwastro.RctGwaLVAlertJSON attribute*), 280
UPLOAD (*llama.files.gwastro.RctGwaLvcGcnXml attribute*), 280
UPLOAD (*llama.files.gwastro.RctGwaLvcRetractionXml attribute*), 280
UPLOAD (*llama.files.gwastro.RctGwaSkymapInfo attribute*), 281
UPLOAD (*llama.files.i3.json.RctGdbIceCubeNeutrinoList attribute*), 219
UPLOAD (*llama.files.i3.json.RctSlkLmaIceCubeNeutrinoList attribute*), 219
UPLOAD (*llama.files.i3.txt.RctGwaIceCubeNeutrinoListTxt attribute*), 229
UPLOAD (*llama.files.RctSlkI3Coi...SummaryI3LvcPdf attribute*), 165
UPLOAD (*llama.files.RctSlkLmaCoi...ScatterI3LvcPdf attribute*), 167
UPLOAD (*llama.files.RctSlkLmaCoi...ScatterI3LvcPng attribute*), 167
UPLOAD (*llama.files.RctSlkLmaCoi...ScatterZtfI3LvcPdf attribute*), 168
UPLOAD (*llama.files.RctSlkLmaCoi...ScatterZtfI3LvcPng attribute*), 168
UPLOAD (*llama.files.RctSlkLmaCoi...ScatterZtfLVCPdf attribute*), 169
UPLOAD (*llama.files.RctSlkLmaCoi...ScatterZtfLVCPng attribute*), 170
UPLOAD (*llama.files.RctSlkLmaCoi...SignificanceI3Lvc attribute*), 171
UPLOAD (*llama.files.RctSlkLmaCoi...SummaryI3LvcPdf attribute*), 171
UPLOAD (*llama.files.RctSlkLmaLVAlertJSON attribute*), 172
UPLOAD (*llama.files.RctSlkLmaLvcDistancesJson attribute*), 174
UPLOAD (*llama.files.RctSlkLmaLvcGcnXml attribute*), 175
UPLOAD (*llama.files.RctSlkLmaLVCGraceDbEventData attribute*), 174
UPLOAD (*llama.files.RctSlkLmaLvcRetractionXml attribute*), 176
UPLOAD (*llama.files.RctSlkLmaSkymapInfo attribute*), 176
UPLOAD (*llama.files.slack.RctSlkLmaLVAlertJSON attribute*), 238
UPLOAD (*llama.files.slack.RctSlkLmaLvcGcnXml attribute*), 239
UPLOAD (*llama.files.slack.RctSlkLmaLvcRetractionXml attribute*), 240
UPLOAD (*llama.files.slack.RctSlkLmaSkymapInfo attribute*), 240
UPLOAD (*llama.files.sms_receipts.RctSMSAdvokJSON attribute*), 289
upload_and_get_manifest() (*in module llama.dev.upload*), 123
upload_dict (*llama.files.gracedb.GraceDBReceipt property*), 276
upload_file() (*in module llama.com.s3*), 113
upload_flag_false() (*in module llama.com.utils*), 115
upload_this() (*llama.com.utils.UploadReceipt class method*), 115
upload_title (*llama.files.slack.SlackReceipt property*), 241
uploaded_file_links (*llama.files.gcn_draft_o2.I3LvcGcnDraft property*), 274
UploadReceipt (*class in llama.com.utils*), 114
upres_nest() (*in module llama.files.healpix.utils*), 210
UR_DEPENDENCIES (*llama.filehandler.FileHandler attribute*), 135
UR_DEPENDENCIES (*llama.files.Advok attribute*), 143
UR_DEPENDENCIES (*llama.files.advok.Advok attribute*), 242
UR_DEPENDENCIES
(llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3 attribute), 243
UR_DEPENDENCIES
(llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3Lvc attribute), 243
UR_DEPENDENCIES
(llama.files.coinc_analyses.CoincLikelihoodPltsFrmLvc attribute), 244
UR_DEPENDENCIES
(llama.files.coinc_analyses.CoincLikelihoodPltsI3Lvc attribute), 244
UR_DEPENDENCIES
(llama.files.coinc_analyses.CoincSkymapFrmI3HDF5 attribute), 245
UR_DEPENDENCIES
(llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDFS attribute), 246
UR_DEPENDENCIES
(llama.files.coinc_analyses.CoincSkymapFrmLvcHDF5 attribute), 248
UR_DEPENDENCIES (*llama.files.coinc_o2.CoincAnalysis attribute*), 248
UR_DEPENDENCIES (*llama.files.coinc_o2.CoincSkymapO2Large attribute*), 250
UR_DEPENDENCIES
(llama.files.coinc_o2.RctGdbCoincSkymapO2Large attribute), 252
UR_DEPENDENCIES
(llama.files.coinc_o2.RctGwaCoi...InitialIcecubeJson attribute), 252
UR_DEPENDENCIES
(llama.files.coinc_o2.RctTeamCoi...InitialIcecubeJson attribute), 253
UR_DEPENDENCIES (*llama.files.coinc_plots.CoincScatterI3LvcPdf attribute*), 254
UR_DEPENDENCIES (*llama.files.coinc_plots.CoincScatterI3LvcPng attribute*), 255
UR_DEPENDENCIES (*llama.files.coinc_plots.CoincScatterZtfI3LvcPdf attribute*), 256
UR_DEPENDENCIES (*llama.files.coinc_plots.CoincScatterZtfI3LvcPng attribute*), 257
UR_DEPENDENCIES (*llama.files.coinc_plots.CoincScatterZtfLVCPdf attribute*), 258
UR_DEPENDENCIES (*llama.files.coinc_plots.CoincScatterZtfLVCPng attribute*), 259

UR_DEPENDENCIES (<i>llama.files.coinc_plots.CoincSummaryI3LvcPdf</i> attribute), 259	UR_DEPENDENCIES (<i>llama.files.gwastro.RctGwaLvcRetractionXml</i> attribute), 281
UR_DEPENDENCIES (<i>llama.files.coinc_plots.CoincSummaryI3LvcTex</i> attribute), 262	UR_DEPENDENCIES (<i>llama.files.gwastro.RctGwaSkymapInfo</i> attribute), 281
UR_DEPENDENCIES (<i>llama.files.coinc_plots.RctSlkI3CoincSummaryI3LvcPdf</i> attribute), 263	UR_DEPENDENCIES (<i>llama.files.i3.IceCubeNeutrinoList</i> attribute), 211
UR_DEPENDENCIES (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPdf</i> attribute), 265	UR_DEPENDENCIES (<i>llama.files.i3.IceCubeNeutrinoListCoincTxt</i> attribute), 213
UR_DEPENDENCIES (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPng</i> attribute), 265	UR_DEPENDENCIES (<i>llama.files.i3.IceCubeNeutrinoListTex</i> attribute), 214
UR_DEPENDENCIES (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPdf</i> attribute), 266	UR_DEPENDENCIES (<i>llama.files.i3.IceCubeNeutrinoListTxt</i> attribute), 215
UR_DEPENDENCIES (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng</i> attribute), 266	UR_DEPENDENCIES (<i>llama.files.i3.json.IceCubeNeutrinoList</i> attribute), 218
UR_DEPENDENCIES (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPdf</i> attribute), 267	UR_DEPENDENCIES (<i>llama.files.i3.json.RctGdbIceCubeNeutrinoList</i> attribute), 219
UR_DEPENDENCIES (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPng</i> attribute), 268	UR_DEPENDENCIES (<i>llama.files.i3.tex.IceCubeNeutrinoList</i> attribute), 219
UR_DEPENDENCIES (<i>llama.files.coinc_plots.RctSlkLmaCoincSummaryI3LvcPdf</i> attribute), 269	UR_DEPENDENCIES (<i>llama.files.i3.tex.IceCubeNeutrinoListCoincTxt</i> attribute), 226
UR_DEPENDENCIES (<i>llama.files.coinc_significance.opa.CoincSignificanceI3Lvc</i> attribute), 180	UR_DEPENDENCIES (<i>llama.files.i3.tex.IceCubeNeutrinoListTxt</i> attribute), 227
UR_DEPENDENCIES (<i>llama.files.coinc_significance.opa.RctSlkLmaCoincSignificanc</i> attribute), 182	UR_DEPENDENCIES (<i>llama.files.i3.txt.IceCubeNeutrinoList</i> attribute), 229
UR_DEPENDENCIES (<i>llama.files.coinc_significance.subthreshold.CoincSignificanc</i> attribute), 185	UR_DEPENDENCIES (<i>llama.files.IceCubeNeutrinoList</i> attribute), 155
UR_DEPENDENCIES (<i>llama.files.coinc_significance.subthreshold.RctSlkLmaCoinc</i> attribute), 188	UR_DEPENDENCIES (<i>llama.files.IceCubeNeutrinoListCoincTxt</i> attribute), 156
UR_DEPENDENCIES (<i>llama.files.coinc_significance.ZtfI3LvcPng</i> attribute), 146	UR_DEPENDENCIES (<i>llama.files.IceCubeNeutrinoListTex</i> attribute), 157
UR_DEPENDENCIES (<i>llama.files.CoincScatterI3LvcPdf</i> attribute), 144	UR_DEPENDENCIES (<i>llama.files.IceCubeNeutrinoListTxt</i> attribute), 159
UR_DEPENDENCIES (<i>llama.files.CoincScatterI3LvcPng</i> attribute), 145	UR_DEPENDENCIES (<i>llama.files.lvalert_advok.LVAlertAdvok</i> attribute), 282
UR_DEPENDENCIES (<i>llama.files.CoincScatterZtfI3LvcPdf</i> attribute), 145	UR_DEPENDENCIES (<i>llama.files.lvalert_json.LVAlertJSON</i> attribute), 282
UR_DEPENDENCIES (<i>llama.files.CoincScatterZtfI3LvcPng</i> attribute), 146	UR_DEPENDENCIES (<i>llama.files.LVAAlertAdvok</i> attribute), 159
UR_DEPENDENCIES (<i>llama.files.CoincScatterZtfLVCPdf</i> attribute), 147	UR_DEPENDENCIES (<i>llama.files.LVAAlertJSON</i> attribute), 159
UR_DEPENDENCIES (<i>llama.files.CoincScatterZtfLVCPng</i> attribute), 148	UR_DEPENDENCIES (<i>llama.files.lvc_gcn_xml.LvcGcnXml</i> attribute), 284
UR_DEPENDENCIES (<i>llama.files.CoincSignificanceI3Lvc</i> attribute), 149	UR_DEPENDENCIES (<i>llama.files.lvc_skymap.LvcDistancesJson</i> attribute), 232
UR_DEPENDENCIES (<i>llama.files.CoincSummaryI3LvcPdf</i> attribute), 150	UR_DEPENDENCIES (<i>llama.files.lvc_skymap.LvcSkymapFits</i> attribute), 233
UR_DEPENDENCIES (<i>llama.files.CoincSummaryI3LvcTex</i> attribute), 152	UR_DEPENDENCIES (<i>llama.files.lvc_skymap.LvcSkymapHdf5</i> attribute), 233
UR_DEPENDENCIES (<i>llama.files.fermi_grb.FermiGRBsJSON</i> attribute), 270	UR_DEPENDENCIES (<i>llama.files.lvc_skymap_mat.LvcSkymapMat</i> attribute), 286
UR_DEPENDENCIES (<i>llama.files.FermiGRBsJSON</i> attribute), 154	UR_DEPENDENCIES (<i>llama.files.LvcDistancesJson</i> attribute), 162
UR_DEPENDENCIES (<i>llama.files.gcn_draft_o2.I3LvcGcnDraft</i> attribute), 272	UR_DEPENDENCIES (<i>llama.files.LvcGcnXml</i> attribute), 162
UR_DEPENDENCIES (<i>llama.files.gcn_draft_o2.RctTeamI3LvcGcnDraft</i> attribute), 274	UR_DEPENDENCIES (<i>llama.files.LVCGraceDbEventData</i> attribute), 161
UR_DEPENDENCIES (<i>llama.files.gracedb.LVCGraceDbEventData</i> attribute), 277	UR_DEPENDENCIES (<i>llama.files.LvcSkymapFits</i> attribute), 163
UR_DEPENDENCIES (<i>llama.files.gracedb.PAstro</i> attribute), 277	UR_DEPENDENCIES (<i>llama.files.LvcSkymapHdf5</i> attribute), 164
UR_DEPENDENCIES (<i>llama.files.gracedb.RctSlkLmaLVCGraceDbEventData</i> attribute), 278	UR_DEPENDENCIES (<i>llama.files.PAstro</i> attribute), 164
UR_DEPENDENCIES (<i>llama.files.gwastro.RctGwaLVAAlertJSON</i> attribute), 280	UR_DEPENDENCIES (<i>llama.files.RctSlkI3CoincSummaryI3LvcPdf</i> attribute), 165
UR_DEPENDENCIES (<i>llama.files.gwastro.RctGwaLvcGcnXml</i> attribute), 280	UR_DEPENDENCIES (<i>llama.files.RctSlkLmaCoincScatterI3LvcPdf</i> attribute), 167
	UR_DEPENDENCIES (<i>llama.files.RctSlkLmaCoincScatterI3LvcPng</i> attribute), 167
	UR_DEPENDENCIES (<i>llama.files.RctSlkLmaCoincScatterZtfI3LvcPng</i> attribute), 168
	UR_DEPENDENCIES (<i>llama.files.RctSlkLmaCoincScatterZtfLVCPdf</i> attribute), 169
	UR_DEPENDENCIES (<i>llama.files.RctSlkLmaCoincScatterZtfLVCPng</i> attribute), 170
	UR_DEPENDENCIES (<i>llama.files.RctSlkLmaCoincSignificanceI3Lvc</i> attribute), 171

UR_DEPENDENCIES (<i>llama.files.RctSlkLmaCoincSummaryI3LvcPdf</i> attribute), 171	UR_DEPENDENCY_TREE (<i>llama.files.coinc_o2.RctGwaCoincAnalysisInitialIcecubeJson</i> attribute), 253
UR_DEPENDENCIES (<i>llama.files.RctSlkLmaLVAAlertJSON</i> attribute), 172	UR_DEPENDENCY_TREE (<i>llama.files.coinc_o2.RctGwaCoincSkymapO2Large</i> attribute), 253
UR_DEPENDENCIES (<i>llama.files.RctSlkLmaLvcDistancesJson</i> attribute), 174	UR_DEPENDENCY_TREE (<i>llama.files.coinc_o2.RctTeamCoincAnalysisInitialIcecubeJson</i> attribute), 254
UR_DEPENDENCIES (<i>llama.files.RctSlkLmaLvcGcnXml</i> attribute), 175	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.CoincScatterI3LvcPdf</i> attribute), 254
UR_DEPENDENCIES (<i>llama.files.RctSlkLmaLvcGraceDbEventData</i> attribute), 174	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.CoincScatterI3LvcPng</i> attribute), 255
UR_DEPENDENCIES (<i>llama.files.RctSlkLmaLvcRetractionXml</i> attribute), 176	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.CoincScatterZtfI3LvcPdf</i> attribute), 257
UR_DEPENDENCIES (<i>llama.files.RctSlkLmaSkymapInfo</i> attribute), 176	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.CoincScatterZtfI3LvcPng</i> attribute), 257
UR_DEPENDENCIES (<i>llama.files.skymap_info.SkymapInfo</i> attribute), 236	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.CoincScatterZtfI3LvcPng</i> attribute), 257
UR_DEPENDENCIES (<i>llama.files.SkymapInfo</i> attribute), 176	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.CoincScatterZtfI3LvcPng</i> attribute), 257
UR_DEPENDENCIES (<i>llama.files.slack.RctSlkLmaLVAAlertJSON</i> attribute), 239	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.CoincScatterZtfI3LvcPng</i> attribute), 257
UR_DEPENDENCIES (<i>llama.files.slack.RctSlkLmaLvcGcnXml</i> attribute), 239	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.CoincScatterZtfI3LvcPng</i> attribute), 257
UR_DEPENDENCIES (<i>llama.files.slack.RctSlkLmaLvcRetractionXml</i> attribute), 240	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.CoincScatterZtfI3LvcPng</i> attribute), 257
UR_DEPENDENCIES (<i>llama.files.slack.RctSlkLmaSkymapInfo</i> attribute), 240	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.CoincScatterZtfI3LvcPng</i> attribute), 257
UR_DEPENDENCIES (<i>llama.files.sms_receipts.RctSMSAdvokJSON</i> attribute), 289	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.CoincScatterZtfI3LvcPng</i> attribute), 259
UR_DEPENDENCIES (<i>llama.files.uw_summary.UwSummary</i> attribute), 291	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.CoincSummaryI3LvcPdf</i> attribute), 260
UR_DEPENDENCIES (<i>llama.files.ztf_trigger_list.ZtfTriggerList</i> attribute), 292	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.CoincSummaryI3LvcTex</i> attribute), 262
UR_DEPENDENCIES (<i>llama.files.ZtfTriggerList</i> attribute), 179	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.RctSlkI3CoincSummaryI3LvcPdf</i> attribute), 264
UR_DEPENDENCIES (<i>llama.test.test_filehandler.MockFh1</i> attribute), 331	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPdf</i> attribute), 265
UR_DEPENDENCIES (<i>llama.test.test_filehandler.MockFh2</i> attribute), 331	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterI3LvcPng</i> attribute), 265
UR_DEPENDENCIES (<i>llama.test.test_filehandler.MockFh3</i> attribute), 331	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPdf</i> attribute), 266
UR_DEPENDENCIES (<i>llama.test.test_filehandler.MockFh4</i> attribute), 332	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng</i> attribute), 267
UR_DEPENDENCIES (<i>llama.test.test_filehandler.MockFh5</i> attribute), 333	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterZtfI3LvcPng</i> attribute), 268
UR_DEPENDENCY_TREE (<i>llama.filehandler.FileHandler</i> attribute), 135	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPdf</i> attribute), 268
UR_DEPENDENCY_TREE (<i>llama.files.Advok</i> attribute), 143	UR_DEPENDENCY_TREE (<i>llama.files.coinc_plots.RctSlkLmaCoincScatterZtfLVCPdf</i> attribute), 268
UR_DEPENDENCY_TREE (<i>llama.files.advok.Advok</i> attribute), 242	UR_DEPENDENCY_TREE (<i>llama.files.coinc_scatter.RctSlkLmaCoincScatterI3LvcPng</i> attribute), 265
UR_DEPENDENCY_TREE (<i>llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3</i> attribute), 243	UR_DEPENDENCY_TREE (<i>llama.files.coinc_scatter.RctSlkLmaCoincScatterZtfI3LvcPng</i> attribute), 266
UR_DEPENDENCY_TREE (<i>llama.files.coinc_analyses.CoincLikelihoodPltsFrmI3Lvc</i> attribute), 243	UR_DEPENDENCY_TREE (<i>llama.files.coinc_scatter.RctSlkLmaCoincScatterZtfI3LvcPng</i> attribute), 267
UR_DEPENDENCY_TREE (<i>llama.files.coinc_analyses.CoincLikelihoodPltsFrmLvc</i> attribute), 244	UR_DEPENDENCY_TREE (<i>llama.files.coinc_scatter.RctSlkLmaCoincScatterZtfI3LvcPng</i> attribute), 268
UR_DEPENDENCY_TREE (<i>llama.files.coinc_analyses.CoincLikelihoodPltsI3Lvc</i> attribute), 244	UR_DEPENDENCY_TREE (<i>llama.files.coinc_scatter.RctSlkLmaCoincScatterZtfLVCPdf</i> attribute), 268
UR_DEPENDENCY_TREE (<i>llama.files.coinc_analyses.CoincSkymapFrmI3HDF5</i> attribute), 245	UR_DEPENDENCY_TREE (<i>llama.files.coinc_scatter.RctSlkLmaCoincScatterZtfLVCPdf</i> attribute), 268
UR_DEPENDENCY_TREE (<i>llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDF5</i> attribute), 246	UR_DEPENDENCY_TREE (<i>llama.files.coinc_scatter.RctSlkLmaCoincScatterZtfLVCPdf</i> attribute), 269
UR_DEPENDENCY_TREE (<i>llama.files.coinc_analyses.CoincSkymapFrmI3LvcHDF5</i> attribute), 247	UR_DEPENDENCY_TREE (<i>llama.files.coinc_scatter.RctSlkLmaCoincScatterZtfLVCPdf</i> attribute), 269
UR_DEPENDENCY_TREE (<i>llama.files.coinc_analyses.CoincSkymapI3LvcHDF5</i> attribute), 248	UR_DEPENDENCY_TREE (<i>llama.files.coinc_scatter.RctSlkLmaCoincScatterZtfLVCPdf</i> attribute), 269
UR_DEPENDENCY_TREE (<i>llama.files.coinc_o2.CoincAnalysis</i> attribute), 248	UR_DEPENDENCY_TREE (<i>llama.files.coinc_scatter.RctSlkLmaCoincScatterZtfLVCPdf</i> attribute), 269
UR_DEPENDENCY_TREE (<i>llama.files.coinc_o2.CoincSkymapO2Large</i> attribute), 250	UR_DEPENDENCY_TREE (<i>llama.files.coinc_scatter.RctSlkLmaCoincScatterZtfLVCPdf</i> attribute), 269
UR_DEPENDENCY_TREE (<i>llama.files.coinc_o2.RctGdbCoincSkymapO2Large</i> attribute), 252	UR_DEPENDENCY_TREE (<i>llama.files.coinc_scatter.RctSlkLmaCoincScatterZtfLVCPdf</i> attribute), 269
	UR_DEPENDENCY_TREE (<i>llama.files.CoincScatterI3LvcPng</i> attribute), 144

UR_DEPENDENCY_TREE (*llama.files.CoincScatterI3LvcPng attribute*),
145
 UR_DEPENDENCY_TREE (*llama.files.CoincScatterZtfI3LvcPdf attribute*),
146
 UR_DEPENDENCY_TREE (*llama.files.CoincScatterZtfI3LvcPng attribute*),
146
 UR_DEPENDENCY_TREE (*llama.files.CoincScatterZtfLVCPdf attribute*),
147
 UR_DEPENDENCY_TREE (*llama.files.CoincScatterZtfLVCPng attribute*),
148
 UR_DEPENDENCY_TREE (*llama.files.CoincSignificanceI3Lvc attribute*),
149
 UR_DEPENDENCY_TREE (*llama.files.CoincSummaryI3LvcPdf attribute*),
150
 UR_DEPENDENCY_TREE (*llama.files.CoincSummaryI3LvcTex attribute*),
152
 UR_DEPENDENCY_TREE (*llama.files.fermi_grb.FermiGRBsJSON attribute*),
270
 UR_DEPENDENCY_TREE (*llama.files.FermiGRBsJSON attribute*), 154
 UR_DEPENDENCY_TREE (*llama.files.gcn_draft_o2.I3LvcGcnDraft attribute*), 272
 UR_DEPENDENCY_TREE
 (*llama.files.gcn_draft_o2.RctTeamI3LvcGcnDraft attribute*), 275
 UR_DEPENDENCY_TREE (*llama.files.gracedb.LVCGraceDbEventData attribute*), 277
 UR_DEPENDENCY_TREE (*llama.files.gracedb.PAstro attribute*), 277
 UR_DEPENDENCY_TREE
 (*llama.files.gracedb.RctSlkLmaLVCGraceDbEventData attribute*), 278
 UR_DEPENDENCY_TREE (*llama.files.gwastro.RctGwaLVAAlertJSON attribute*), 280
 UR_DEPENDENCY_TREE (*llama.files.gwastro.RctGwaLvcGcnXml attribute*), 280
 UR_DEPENDENCY_TREE (*llama.files.gwastro.RctGwaLvcRetractionXml attribute*), 281
 UR_DEPENDENCY_TREE (*llama.files.gwastro.RctGwaSkymapInfo attribute*), 281
 UR_DEPENDENCY_TREE (*llama.files.i3.IceCubeNeutrinoList attribute*),
211
 UR_DEPENDENCY_TREE (*llama.files.i3.IceCubeNeutrinoListCoincTxt attribute*), 213
 UR_DEPENDENCY_TREE (*llama.files.i3.IceCubeNeutrinoListTex attribute*), 215
 UR_DEPENDENCY_TREE (*llama.files.i3.IceCubeNeutrinoListTxt attribute*), 215
 UR_DEPENDENCY_TREE (*llama.files.i3.json.IceCubeNeutrinoList attribute*),
218
 UR_DEPENDENCY_TREE
 (*llama.files.i3.json.RctGdbIceCubeNeutrinoList attribute*),
219
 UR_DEPENDENCY_TREE
 (*llama.files.i3.json.RctSlkLmaIceCubeNeutrinoList attribute*), 220
 UR_DEPENDENCY_TREE (*llama.files.i3.tex.IceCubeNeutrinoListTex attribute*), 226
 UR_DEPENDENCY_TREE (*llama.files.i3.txt.IceCubeNeutrinoListCoincTxt attribute*), 228
 UR_DEPENDENCY_TREE (*llama.files.i3.txt.IceCubeNeutrinoListTxt attribute*), 229
 UR_DEPENDENCY_TREE
 (*llama.files.i3.txt.RctGwaIceCubeNeutrinoListTxt attribute*),
230
 UR_DEPENDENCY_TREE (*llama.files.IceCubeNeutrinoList attribute*), 155
 UR_DEPENDENCY_TREE (*llama.files.IceCubeNeutrinoListCoincTxt attribute*), 156
 UR_DEPENDENCY_TREE (*llama.files.IceCubeNeutrinoListTex attribute*),
158
 UR_DEPENDENCY_TREE (*llama.files.IceCubeNeutrinoListTxt attribute*),
159
 UR_DEPENDENCY_TREE (*llama.files.Ivalert_advok.LVAlertAdvok attribute*), 282
 UR_DEPENDENCY_TREE (*llama.files.Ivalert_json.LVAlertJSON attribute*), 282
 UR_DEPENDENCY_TREE (*llama.files.LVAlertAdvok attribute*), 159
 UR_DEPENDENCY_TREE (*llama.files.LVAlertJSON attribute*), 160
 UR_DEPENDENCY_TREE (*llama.files.lvc_gcn_xml.LvcGcnXml attribute*),
284
 UR_DEPENDENCY_TREE (*llama.files.lvc_skymap.LvcDistancesJson attribute*), 232
 UR_DEPENDENCY_TREE (*llama.files.lvc_skymap.LvcSkymapFits attribute*), 233
 UR_DEPENDENCY_TREE (*llama.files.lvc_skymap.LvcSkymapHdf5 attribute*), 233
 UR_DEPENDENCY_TREE (*llama.files.lvc_skymap_mat.LvcSkymapMat attribute*), 286
 UR_DEPENDENCY_TREE (*llama.files.LvcDistancesJson attribute*), 162
 UR_DEPENDENCY_TREE (*llama.files.LvcGcnXml attribute*), 162
 UR_DEPENDENCY_TREE (*llama.files.LVCGraceDbEventData attribute*),
161
 UR_DEPENDENCY_TREE (*llama.files.LvcSkymapFits attribute*), 163
 UR_DEPENDENCY_TREE (*llama.files.LvcSkymapHdf5 attribute*), 164
 UR_DEPENDENCY_TREE (*llama.files.PAstro attribute*), 164
 UR_DEPENDENCY_TREE (*llama.files.RctSlkI3CoincSummaryI3LvcPdf attribute*), 166
 UR_DEPENDENCY_TREE (*llama.files.RctSlkLmaCoincScatterI3LvcPdf attribute*), 167
 UR_DEPENDENCY_TREE (*llama.files.RctSlkLmaCoincScatterI3LvcPng attribute*), 167
 UR_DEPENDENCY_TREE (*llama.files.RctSlkLmaCoincScatterZtfI3LvcPdf attribute*), 168
 UR_DEPENDENCY_TREE
 (*llama.files.RctSlkLmaCoincScatterZtfI3LvcPng attribute*),
169
 UR_DEPENDENCY_TREE (*llama.files.RctSlkLmaCoincScatterZtfLVCPdf attribute*), 170
 UR_DEPENDENCY_TREE (*llama.files.RctSlkLmaCoincScatterZtfLVCPng attribute*), 170
 UR_DEPENDENCY_TREE (*llama.files.RctSlkLmaCoincSignificanceI3Lvc attribute*), 171
 UR_DEPENDENCY_TREE (*llama.files.RctSlkLmaCoincSummaryI3LvcPdf attribute*), 172
 UR_DEPENDENCY_TREE (*llama.files.RctSlkLmaLVAAlertJSON attribute*),
172
 UR_DEPENDENCY_TREE (*llama.files.RctSlkLmaLvcDistancesJson attribute*), 174
 UR_DEPENDENCY_TREE (*llama.files.RctSlkLmaLvcGcnXml attribute*),
175
 UR_DEPENDENCY_TREE (*llama.files.RctSlkLmaLVCGraceDbEventData attribute*), 174
 UR_DEPENDENCY_TREE (*llama.files.RctSlkLmaLvcRetractionXml attribute*), 176
 UR_DEPENDENCY_TREE (*llama.files.RctSlkLmaSkymapInfo attribute*),
176
 UR_DEPENDENCY_TREE (*llama.files.skymap_info.SkymapInfo attribute*),
236
 UR_DEPENDENCY_TREE (*llama.files.SkymapInfo attribute*), 176
 UR_DEPENDENCY_TREE (*llama.files.slack.RctSlkLmaLVAAlertJSON attribute*), 239
 UR_DEPENDENCY_TREE (*llama.files.slack.RctSlkLmaLvcGcnXml attribute*),
239
 UR_DEPENDENCY_TREE (*llama.files.slack.RctSlkLmaLvcRetractionXml attribute*), 240
 UR_DEPENDENCY_TREE (*llama.files.slack.RctSlkLmaSkymapInfo attribute*),
241

UR_DEPENDENCY_TREE (*llama.files.sms_receipts.RctSMSAdvokJSON attribute*), 289
 UR_DEPENDENCY_TREE (*llama.files.uw_summary.UwSummary attribute*), 291
 UR_DEPENDENCY_TREE (*llama.files.ztf_trigger_list.ZtfTriggerList attribute*), 292
 UR_DEPENDENCY_TREE (*llama.files.ZtfTriggerList attribute*), 179
 UR_DEPENDENCY_TREE (*llama.test.test_filehandler.MockFh1 attribute*), 331
 UR_DEPENDENCY_TREE (*llama.test.test_filehandler.MockFh2 attribute*), 331
 UR_DEPENDENCY_TREE (*llama.test.test_filehandler.MockFh3 attribute*), 331
 UR_DEPENDENCY_TREE (*llama.test.test_filehandler.MockFh4 attribute*), 332
 UR_DEPENDENCY_TREE (*llama.test.test_filehandler.MockFh5 attribute*), 333
url (*llama.detectors.DetectorTuple attribute*), 118
url (*llama.utils.RemoteFileCacherTuple attribute*), 338
utc2gps() (*in module llama.utils*), 344
utc2mjd() (*in module llama.utils*), 344
utcnow() (*in module llama.com.email*), 106
utcnow() (*in module llama.filehandler*), 140
utcnow() (*in module llama.intent*), 300
utcnow() (*in module llama.vetoes*), 355
UwSummary (*class in llama.files.uw_summary*), 291

V

v0ts() (*in module llama.run*), 322
values() (*llama.classes.ImmutableDict method*), 96
values() (*llama.flags.FlagDict method*), 293
veccls() (*in module llama.utils*), 344
vecstr() (*in module llama.utils*), 344
vectypes() (*in module llama.utils*), 344
version (*llama.cli.Parsers attribute*), 100
version (*llama.files.lvc_skymap.utils.SkymapFilename property*), 235
veto (*llama.vetoes.VetoMixin property*), 354
vetoes (*llama.vetoes.VetoHandlerTuple attribute*), 354
vetoes() (*llama.vetoes.VetoMixin class method*), 354
VetoException, 353
veto_filenames (*llama.vetoes.VetoHandler property*), 353
veto_filepaths (*llama.vetoes.VetoHandler property*), 353
VetoHandler (*class in llama.vetoes*), 353
VetoHandlerTuple (*class in llama.vetoes*), 354
VetoMixin (*class in llama.vetoes*), 354
vis (*llama.Run property*), 91
vis (*llama.run.Run property*), 321

W

wall_time (*llama.meta.MetaData property*), 308
wall_times() (*llama.run.RunVisualization method*), 321
write() (*llama.intent.CoolDown method*), 299
write() (*llama.intent.Intent method*), 300
write() (*llama.meta.MetaData method*), 308
write_gzip() (*in module llama.utils*), 344
write_healpix() (*llama.files.healpix.HEALPixSkyMapAnalysis method*), 194
write_healpix() (*llama.files.healpix.HEALPixSkyMapFileHandler method*), 195
write_healpix_to_file()
(llama.files.healpix.HEALPixSkyMapFileHandler static method), 195
write_json() (*llama.classes.JsonRiderMixin method*), 96
write_to_zip() (*in module llama.utils*), 345
writeLog() (*llama.com.gracedbGraceDb method*), 110

Y

YELLOW (*llama.classes.Colors attribute*), 95
yellow (*llama.classes.Colors attribute*), 95

Z

zen_az2ra_dec() (*in module llama.files.i3.utils*), 231
zenith (*llama.files.i3.Neutrino property*), 216
ZtfTriggerList (*class in llama.files*), 178
ZtfTriggerList (*class in llama.files.ztf_trigger_list*), 291