

---

# **GWSumm Documentation**

***Release 2.2.6.dev1+g105a704.d20240419***

**Duncan Macleod, Alex Urban**

**Apr 19, 2024**



# CONTENTS

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Contributing</b>	<b>5</b>
	<b>Python Module Index</b>	<b>109</b>
	<b>Index</b>	<b>111</b>



The *gwsumm* package is a tool used by the [Laser Interferometer Gravitational-wave Observatory \(LIGO\)](#) to collect, aggregate, and visually summarise the sundry data produced throughout the experiment in order to characterise instrument performance.

The output of this package, known internally as the ‘summary pages’, give an archive of a number of figures or merit, including time-series amplitude trends, frequency spectra and spectrograms, and transient event triggers.

This package includes a collection of command-line utilities and a python module:

```
import gwsumm
```



## INSTALLATION

GWSumm is best installed with [conda](#):

```
conda install -c conda-forge gwsumm
```

but can also be installed with [pip](#):

```
python -m pip install gwsumm
```

Note, users with *LIGO.ORG* credentials have access to a software container with a regularly-updated build of GWSumm. For more information please refer to the [LSCSoft Conda](#) documentation.





## CONTRIBUTING

All code should follow the Python Style Guide outlined in [PEP 0008](#); users can use the [flake8](#) package to check their code for style issues before submitting.

See the [contributions guide](#) for the recommended procedure for proposing additions/changes.

The GWSumm project is hosted on GitHub:

- Issue tickets: <https://github.com/gwpy/gwsumm/issues>
- Source code: <https://github.com/gwpy/gwsumm>

## 2.1 License

GWSumm is distributed under the [GNU General Public License](#).

### 2.1.1 What is GWSumm?

The *gwsumm* package (‘the summary pages’) is a python toolbox that can be used to generate a structured HTML webpage displaying graphical data that describe any and all aspects of gravitational-wave interferometer performance. The summary pages were developed in collaboration between the LIGO Laboratory and the GEO600 project with the goal of generating an automated daily summary of laser-interferometer operations and performance.

The LIGO Summary Pages are used to characterize and monitor the status of the detectors and their subsystems. In addition, data products and webpages from other analysis tools are included in the Summary Pages.

The output acts as a kind of daily magazine, allowing instrument scientists and data analysis teams a archived, searchable summary of the key figures of merit that will determine the sensitivity and ultimately the science output of these instruments.

Those readers who are members of the LIGO Scientific Collaboration, the Virgo Collaboration, or KAGRA can view the current LIGO summary pages at the following sites:

H1	<a href="https://ldas-jobs.ligo-wa.caltech.edu/~detchar/summary/">https://ldas-jobs.ligo-wa.caltech.edu/~detchar/summary/</a>
L1	<a href="https://ldas-jobs.ligo-la.caltech.edu/~detchar/summary/">https://ldas-jobs.ligo-la.caltech.edu/~detchar/summary/</a>

## Working model

The GWSumm package provides an abstract set of classes from which any user can build their own python program to read, manipulate, and display data. However, for the specific purpose of the LIGO instrumental summary pages, the `gw_summary` command-line executable is used to read in a number of INI-format configuration files that define what data should be read, how it should be manipulated, and how it should all be displayed.

These configuration files are made up [tab-xxx] section with the following format:

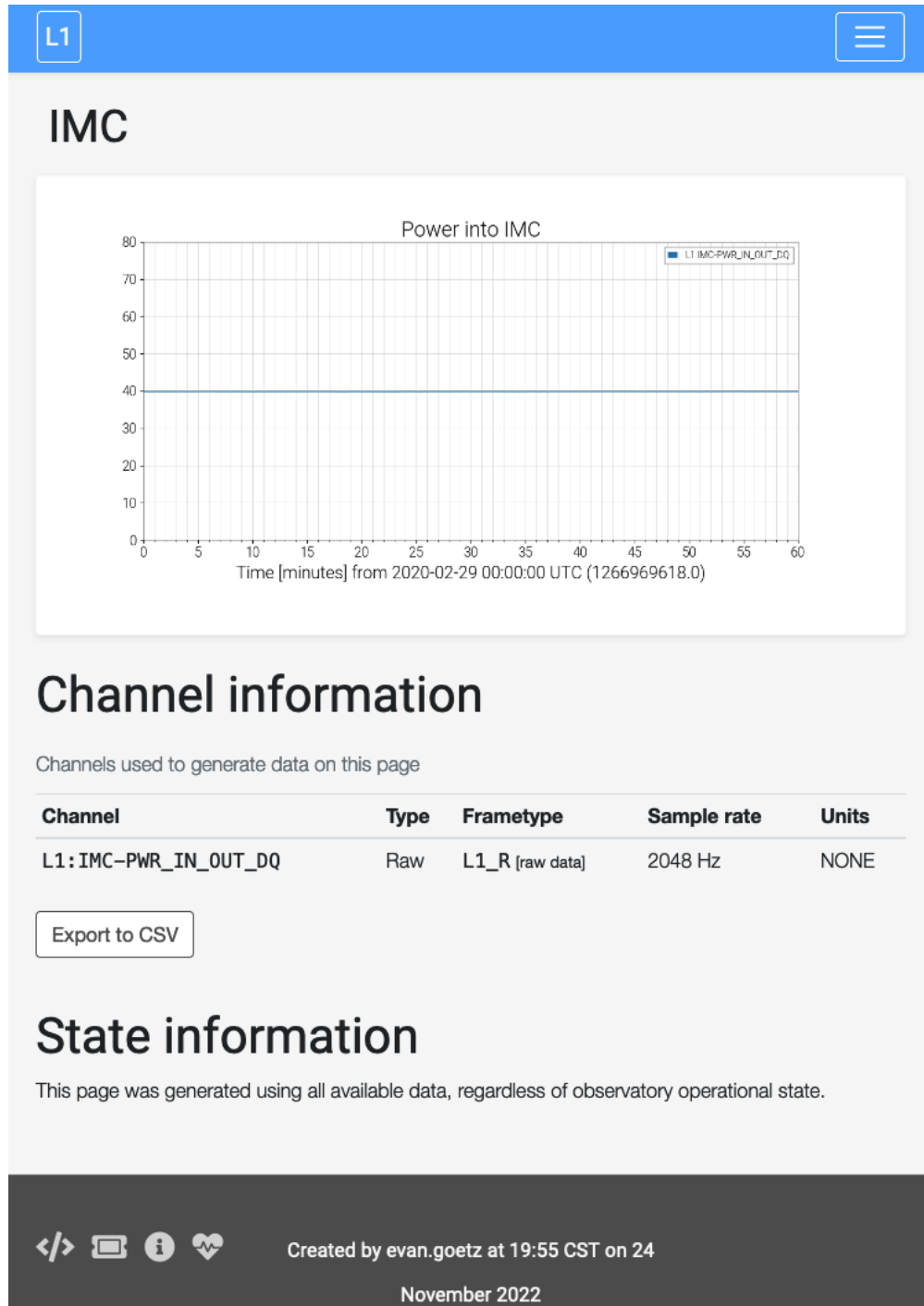
```
[tab-IMC]
name = Input mode cleaner
shortname = IMC
1 = L1:IMC-PWR_IN_OUT_DQ timeseries
1-ylim = 0,80
1-title = 'Power into IMC'
[html]
issues =
```

This block defines the IMC tab, with a `name` (and a `shortname`): a single `timeseries` plot of the L1:IMC-PWR\_IN\_OUT\_DQ channel. The plot has been customised with a y-axis limit and a title. This also defines the required [html] section, where the required key issues is defined. This example can be saved to a file called `imc.ini`.

This tab is then generated by passing it to the `gw_summary` executable, along with some GPS times over which to run:

```
$ gw_summary gps 'Feb 29 2020 00:00' 'Feb 29 2020 01:00' --config-file imc.ini
```

This minimal setup will produce the following HTML page <1266969618-1266973218/imc/index.html>:



## 2.1.2 Command-line interface

### GW Summary

The primary interface for GWSumm is a command-line utility called *gw\_summary*. For a full explanation of the available command-line arguments and options, you can run

```
$ python ../bin/gw_summary --help
python: can't open file '/home/docs/checkouts/readthedocs.org/user_builds/gwsumm/
↳checkouts/latest/docs/./bin/gw_summary': [Errno 2] No such file or directory
```

This tool can be run in four modes: daily, weekly, and monthly analyses, and a specific range of GPS times.

### Day mode

To run in daily summary mode, the following command-line options are available:

```
$ python gw_summary day --help
python: can't open file '/home/docs/checkouts/readthedocs.org/user_builds/gwsumm/
↳checkouts/latest/docs/gw_summary': [Errno 2] No such file or directory
```

### Week mode

The arguments in weekly mode are as follows:

```
$ python gw_summary week --help
python: can't open file '/home/docs/checkouts/readthedocs.org/user_builds/gwsumm/
↳checkouts/latest/docs/gw_summary': [Errno 2] No such file or directory
```

### Month mode

In monthly mode:

```
$ python gw_summary month --help
python: can't open file '/home/docs/checkouts/readthedocs.org/user_builds/gwsumm/
↳checkouts/latest/docs/gw_summary': [Errno 2] No such file or directory
```

### GPS mode

To run within a specific (but arbitrary) range of GPS seconds:

```
$ python gw_summary gps --help
python: can't open file '/home/docs/checkouts/readthedocs.org/user_builds/gwsumm/
↳checkouts/latest/docs/gw_summary': [Errno 2] No such file or directory
```

## Batch mode

To stage a batch of analyses with a large collection of configuration files, as is done in embarrassingly parallel fashion when the summary pages run online, you can use the `gw_summary_pipe` command-line utility. This tool uses [HTCondor](#) to schedule and run jobs in parallel.

To see all the available arguments and options for this tool, you can run with `-help` as usual:

```
$ python gw_summary_pipe --help
python: can't open file '/home/docs/checkouts/readthedocs.org/user_builds/gwsumm/
↳checkouts/latest/docs/gw_summary_pipe': [Errno 2] No such file or directory
```

## 2.1.3 Automatic generation for LIGO

The LIGO Detector Characterization group use the `gwsumm` package to generate daily, weekly, and monthly summaries of the performance of the LIGO detectors. These data generation runs are automatically completed using the HTCondor high-throughput job scheduler.

Members of the LIGO Scientific Collaboration or the Virgo Collaboration can view more details on the HTCondor configuration [here](#).

## 2.1.4 Introduction to Tabs

GWsumm can be used either from the command line as described in [CLI interface](#) or as a package to programmatically generate pages called “tabs”.

A *Tab* is a single, configurable page of output, containing some data. Each *Tab* is written in its own HTML page, and can be written to contain any set of data, with any format.

The basic object provided by `mod:gwsumm.tabs` is the *Tab*, which allows embedding of arbitrary HTML text into a standard surrounding HTML framework. The *Tab* also defines the API for other tabs.

### Simple *Tab* use

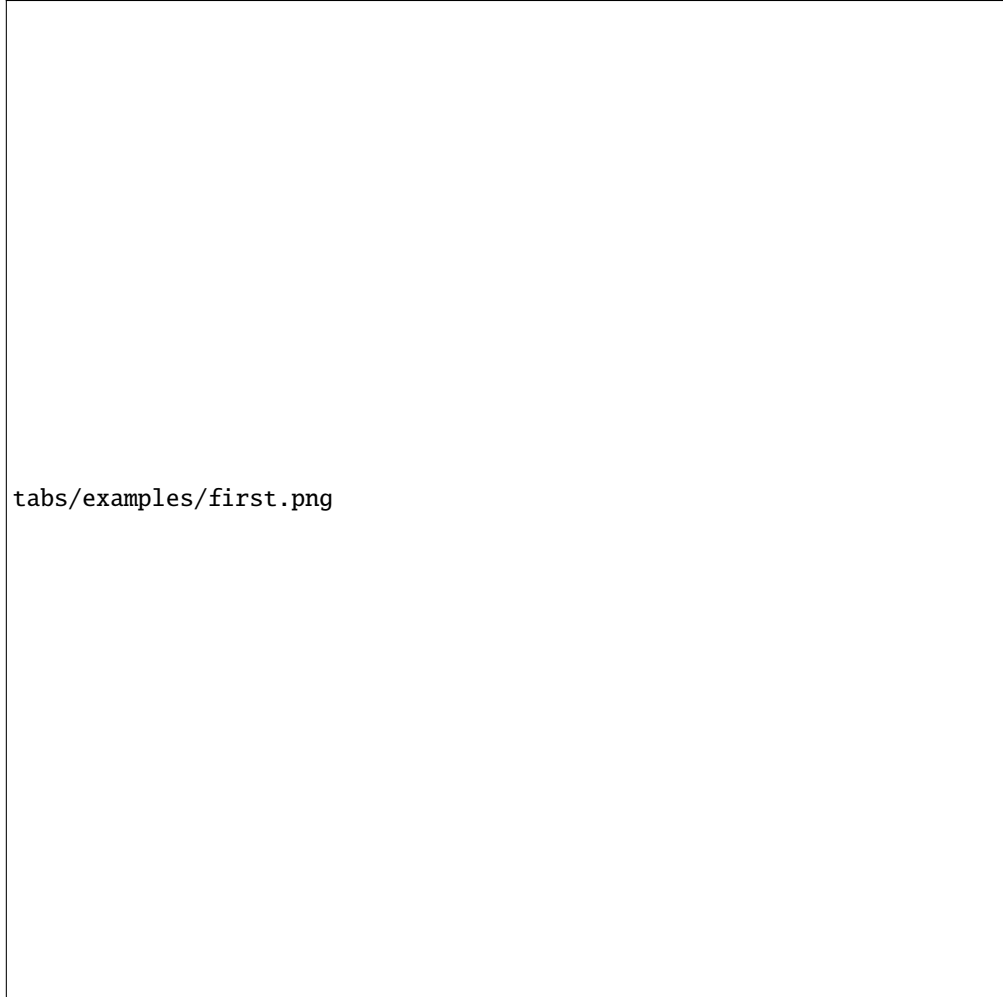
A simple *Tab* can be created in only two steps

```
from gwsumm.tabs import Tab
mytab = Tab('My first tab')
mytab.write_html("This tab doesn't do very much")
```

This will create a directory under the current one,

- `my_first_tab/` containing the HTML for the new tab

The output webpage looks like:



tabs/examples/first.png

The content given to *Tab.write\_html* is passed along untouched, so can contain any HTML you like.

## Generating websites

The *next page* will guide you through created groups of tabs and combining them to generate a fully-fledged website complete with navigation.

## 2.1.5 Generating Websites

As we have seen, generating standalone pages is trivial using GWSumm. What would be more useful would be to generate linked sets of pages, aka a website.

### Navigation

The key difference between standalone pages and a website is the ability to navigate between them. The `Tab.write_html` method will take care of that for you if you pass it all of the tabs:

```
from gwsomm.tabs import Tab
tab1 = Tab('Tab 1')
tab2 = Tab('Tab 2')
tabs = [tab1, tab2]
tab1.write_html('This is tab 1', tabs=tabs)
tab2.write_html('This is tab 2', tabs=tabs)
```

This will write each tab into its own directory, as before, but the HTML will now contain an `<nav></nav>` block above the banner to allow navigation between the pages.

### Tab parents

In the above example, each tab is included as a link in the navigation bar. However, in larger websites with many pages, the navigation can quickly become cluttered and will start to overflow the width of the page. This can be avoided by declaring `~Tab.parent` for sets of tabs:

```
tab1 = Tab('Tab 1')
tab2a = Tab('A', parent='Tab 2')
tab2b = Tab('B', parent=tab2a.parent)
tabs = [tab1, tab2a, tab2b]
tab1.write_html('This is tab 1', tabs=tabs)
tab2a.write_html('This is tab 2A', tabs=tabs)
tab2b.write_html('This is tab 2B', tabs=tabs)
```

Here we have set a *parent* tab for 2A, and used the same for 2B, which creates a dropdown menu in the navigation bar linking to these tabs. ‘Tab 2’ is never created, but is used only for navigation.

### Tab groups

For even larger websites, sets of tabs under a single parent can be further separated into *groups* `<Tab.group>`. For example, to put 2A into group 1 and 2B into group 2, we can write:

```
tab1 = Tab('Tab 1')
tab2a = Tab('A', parent='Tab 2', group='1')
tab2b = Tab('B', parent=tab2a.parent, group='2')
tabs = [tab1, tab2a, tab2b]
tab1.write_html('This is tab 1', tabs=tabs)
tab2a.write_html('This is tab 2A', tabs=tabs)
tab2b.write_html('This is tab 2B', tabs=tabs)
```

## 2.1.6 *Tab* modes

In its simplest form, the *Tab* is essentially a blank canvas on which to write whatever you want. However, the original mandate for GWSumm was to provide a framework in which to generate automatic summaries of LIGO data, over a given timescale.

To handle data processing, rather than static HTML generation, each *Tab* has a type, based on its relation to any interval in time

The type of a *Tab* is set automatically when it is created based on the value of the `:attr:~Tab.mode` attribute, so you don't need to remember the above objects.

### Modes

GWSumm currently support seven different *Tab* modes:

Mode	Enum	Description
STATIC	0	No associated time interval
EVENT	1	Associated with a single GPS time, normally around an event
GPS	2	Simple (arbitrary) GPS [start, end) interval
DAY	10	One UTC 24-hour day
WEEK	11	One 7-day week
MONTH	12	One calendar month
YEAR	13	One calendar year

### Assigning modes

Each *Tab* will be assigned a mode (unless specified as follows, the default mode is `STATIC`). The assignment can be done on a per-tab basis by passing the `~Tab.mode` keyword argument when creating a *Tab*, or globally, by using the `:meth:~gwsomm.mode.set_mode`. The latter sets the default mode for all subsequent tabs created in this session.

If a `:attr:~Tab.mode` is given that associates with a GPS time or times, these must be given via the `~IntervalTab.span` or `~EventTab.gps_time` keyword arguments, otherwise a `TypeError` will be raised. The *span* tuple is the (GPS start time, GPS end time)

```
>>> tab = Tab('My first tab', mode='day', span=(0, 100))
>>> print(tab.mode, tab.span)
(10, Segment(0, 100))
>>> tab = Tab('My first tab', mode='EVENT', gpstime=101)
>>> print(tab.mode, tab.gpstime)
(1, LIGOTimeGPS(101,0))
```



## 2.1.7 Tab API

### gwsumm.tabs Package

This module defines the *Tab* API, and all of the built-in tab objects

### Functions

<code>get_tab(name)</code>	Query the registry for the tab class registered to the given name
<code>register_tab(tab[, name, force])</code>	Register a new summary <i>Tab</i> to the given name

### get\_tab

### register\_tab

### Classes

<code>AboutTab(*args, **kwargs)</code>	Page describing how the containing HTML pages were generated
<code>AccountingTab(*args, **kwargs)</code>	Summarise the data recorded by the operating mode channels
<code>BaseTab(name[, index, shortname, parent, ...])</code>	The core <i>Tab</i> object, defining basic functionality
<code>DataTab(*args, **kwargs)</code>	A tab where plots and data summaries are built upon request
<code>Error404Tab(*args, **kwargs)</code>	Custom HTTP 404 error page
<code>EventTab(*args, **kwargs)</code>	<i>Tab</i> defined around a central GPS time
<code>EventTriggerTab(*args, **kwargs)</code>	Custom <i>DataTab</i> displaying a summary of event trigger generation
<code>ExternalTab(*args, **kwargs)</code>	A simple tab to link HTML from an external source
<code>FscanTab(*args, **kwargs)</code>	Custom tab displaying a summary of Fscan results.
<code>GpsTab(name[, index, shortname, parent, ...])</code>	Stub for GPS-related tabs
<code>GraceDbTab(*args, **kwargs)</code>	Custom tab displaying a summary of GraceDb results.
<code>GuardianTab(*args, **kwargs)</code>	Summarises the data recorded by an Advanced LIGO Guardian node.
<code>IntervalTab(*args, **kwargs)</code>	<i>Tab</i> defined within a GPS [start, end) interval
<code>ParentTab(*args, **kwargs)</code>	Dummy <i>Tab</i> only for navigation
<code>PlotTab(*args, **kwargs)</code>	A simple tab to layout some figures in the #main div.
<code>ProcessedTab()</code>	Abstract base class to detect necessity to run <code>Tab.process()</code>
<code>SEIWatchDogTab(*args, **kwargs)</code>	Summarise the WatchDog trips recorded from the SEI system.
<code>StampPEMTTab(*args, **kwargs)</code>	Custom tab displaying a summary of StampPEM results.
<code>StateTab(*args, **kwargs)</code>	Tab with multiple content pages defined via 'states'
<code>StaticTab(name[, index, shortname, parent, ...])</code>	Simple <i>Tab</i> with no GPS association
<code>Tab(*args, **kwargs)</code>	A Simple HTML tab.
<code>TabList([entries])</code>	Custom <i>list</i> of <i>Tab</i> objects with sorting and parsing
<code>UrlTab(*args, **kwargs)</code>	

**AboutTab**

**AccountingTab**

**BaseTab**

**DataTab**

**Error404Tab**

**EventTab**

**EventTriggerTab**

**ExternalTab**

**FscanTab**

**GpsTab**

**GraceDbTab**

**GuardianTab**

**IntervalTab**

**ParentTab**

**PlotTab**

**ProcessedTab**

**SEIWatchDogTab**

**StampPEMTab**

**StateTab**

**StaticTab**

**Tab**

**TabList**

**UriTab**

## Class Inheritance Diagram

### 2.1.8 States

A *SummaryState* defines a sub-set of time over which a *~gwsomm.tabs.Tab* should be processed. Each *SummaryState* is normally tied to one or more data-quality flags marking times during which each of the LIGO instruments was operating in a certain configuration, or was subject to a known noise interference.

#### The state registry

GWSumm defines a state ‘registry’, simply a record of all *SummaryState* objects that have been defined (and registered) so far in a given program. The registry just makes remembering states in complicated programs a little easier.

Any *SummaryState* can be registered with an arbitrary name as follows:

```
>>> from gwsomm.state.registry import register_state
>>> register_state(mystate, 'my state')
```

and can be recovered later:

```
>>> from gwsomm.state.registry import get_state
>>> mystate = get_state('my state')
```

#### API reference

<i>SummaryState</i> (name[, known, active, ...])	An operating state over which to process a <i>~gwsomm.tabs.DataTab</i> .
<i>get_state</i> (key)	Query the registry for the <i>SummaryState</i> registered to the given key
<i>get_states</i> ([keys])	Query the registry for a list of states (defaults to all)
<i>register_state</i> (state[, key, force])	Register a new <i>SummaryState</i> to the given key

### SummaryState

#### *get\_state*

#### *get\_states*

#### *register\_state*

### 2.1.9 Plots

A *Plot* is a representation of an image to be included in the HTML output a tab.

For simple purposes, a *Plot* is just a reference to an existing image file that can be imported into an HTML page via the `<img>` tag.

For more complicated purposes, a number of data plot classes are provided to allow users to generate images on-the-fly. The available classes are:

<code>TimeSeriesDataPlot(*args, **kwargs)</code>	DataPlot of some <i>TimeSeries</i> data.
<code>SpectrogramDataPlot(*args, **kwargs)</code>	DataPlot a Spectrogram
<code>SegmentDataPlot(flags, start, end[, state, ...])</code>	Segment plot of one or more <i>DataQualityFlags</i> < <i>DataQualityFlag</i> >.
<code>StateVectorDataPlot(*args, **kwargs)</code>	DataPlot of some <i>StateVector</i> data.
<code>SpectrumDataPlot(channels, start, end[, ...])</code>	Spectrum plot for a <i>SummaryTab</i>
<code>TimeSeriesHistogramPlot(channels, start, end)</code>	HistogramPlot from a Series
<code>TriggerTimeSeriesDataPlot(*args, **kwargs)</code>	Custom time-series plot to handle discontinuous <i>Time-Series</i> .
<code>TriggerHistogramPlot(*args, **kwargs)</code>	HistogramPlot from a LIGO_LW Table
<code>TriggerRateDataPlot(*args, **kwargs)</code>	TimeSeriesDataPlot of trigger rate.

### TimeSeriesDataPlot

### SpectrogramDataPlot

### SegmentDataPlot

### StateVectorDataPlot

### SpectrumDataPlot

### TimeSeriesHistogramPlot

### TriggerTimeSeriesDataPlot

### TriggerHistogramPlot

### TriggerRateDataPlot

## 2.1.10 gwsumm.mode Module

Job modes

### Functions

<code>get_base(date[, mode])</code>	Determine the correct base attribute for the given date and mode.
<code>get_mode([m])</code>	Return the enum for the given mode, defaults to the current mode.
<code>set_mode(m)</code>	Set the current mode.
<code>unique(enumeration)</code>	Class decorator for enumerations ensuring unique member values.

`get_base`

`get_mode`

`set_mode`

## Classes

<code>Enum(value[, names, module, qualname, type, ...])</code>	Create a collection of name/value pairs.
<code>Mode(value[, names, module, qualname, type, ...])</code>	Enumeration of valid processing 'modes'
<code>OrderedEnum(value[, names, module, ...])</code>	

**Mode**

**OrderedEnum**

**Class Inheritance Diagram**

## 2.1.11 Full API

**gwsumm package**

**Subpackages**

**gwsumm.config package**

**Module contents**

Thin wrapper around configparser

```
class gwsumm.config.GWSummConfigParser(*args, **kwargs)
```

Bases: `ConfigParser`

```
OPTCRE = re.compile('(P<option>[^\s][^=]*)\\s*(P<vi>[=])\\s*(P<value>.*)$')
```

```
finalize()
```

Finalize this *GWSummConfigParser* by running all the loaders

This method is just a shortcut to run each of the following

<code>load_plugins()</code>	Load all plugin modules as defined in the [plugins] section
<code>load_units()</code>	Load all unit definitions as defined in the [units] section
<code>load_channels()</code>	Load all channel definitions as given in the selfuration
<code>load_states([section])</code>	Read and format a list of <i>SummaryState</i> definitions from the given <code>:class:`~configparser.ConfigParser`</code>

**classmethod** `from_configparser(cp)`

Copy an existing `:class:`~configparser.ConfigParser``.

**get\_css**(`section='html'`)

**get\_javascript**(`section='html'`)

**interpolate\_section\_names**(`**kwargs`)

Interpolate a specific key in a section name using val

**load\_channels()**

Load all channel definitions as given in the selfuration

Channels are loaded from sections named [channels-...] or those sections whose name is a channel name in itself

**load\_plugins()**

Load all plugin modules as defined in the [plugins] section

**load\_rcParams**(`section='rcParams'`)

Load custom *matplotlib.rcParams* for plots in this analysis

**load\_states**(`section='states'`)

Read and format a list of *SummaryState* definitions from the given `:class:`~configparser.ConfigParser``

**load\_units()**

Load all unit definitions as defined in the [units] section

**nditems**(`section, **kwargs`)

**ndoptions**(`section, **kwargs`)

**optionxform**

alias of `str`

**read**(`filenames`)

Read and parse a filename or an iterable of filenames.

Files that cannot be opened are silently ignored; this is designed so that you can specify an iterable of potential configuration file locations (e.g. current directory, user's home directory, systemwide directory), and all existing configuration files in the iterable will be read. A single filename may also be given.

Return list of successfully read files.

**set\_date\_options**(`start, end, section='DEFAULT'`)

Set datetime options in [DEFAULT] based on the given times

The following options are set

- *gps-start-time* - the integer GPS start time of this job
- *gps-end-time* - the integer GPS end time of this job
- *yyyy* - the four-digit year of the start date
- *mm* - the two-digit month of the start date
- *dd* - the two-digit day-of-month of the start date
- *yyyymm* - the six-digit year and month of the start date
- *yyyymmdd* - the eight-digit year-month-day of the start date
- *duration* - the duration of the job (seconds)

Additionally, if LAL is available, the following extra options are also set

- *leap-seconds* - the number of leap seconds for the start date
- *gps-start-time-noleap* - the leap-corrected integer GPS start time
- *gps-end-time-noleap* - the leap-corrected integer GPS end time

**set\_ifo\_options**(*ifo*, *observatory*=None, *section*='DEFAULT')

Set configurations options in [DEFAULT] based on the given *ifo*

The following options are set

- *IFO* - the two-character interferometer prefix, e.g. L1
- ***ifo* - the two-character interferometer prefix in lower-case,**  
e.g. l1
- *SITE* - the single-character site ID, e.g. L
- *site* - the single-character site ID,n lower-case e.g. l

Additionally, if *observatory* is given, or the *ifo* matches known observatories, the following option is set

- *observatory* - the name of the observatory, e.g. LIGO Livingston

## gwsumm.data package

### Submodules

#### gwsumm.data.coherence module

Utilities for data handling and display

**gwsumm.data.coherence.add\_coherence\_component\_spectrogram**(*specgram*, *key*=None, *coalesce*=True)

Add a *Coherence spectrogram* to the global memory cache

**gwsumm.data.coherence.complex\_percentile**(*array*, *percentile*)

**gwsumm.data.coherence.get\_coherence\_spectrogram**(*channel\_pair*, *segments*, *config*=None, *cache*=None, *query*=True, *nds*=None, *return\_*=True, *frametype*=None, *nproc*=1, *datafind\_error*='raise', *return\_components*=False, *\*\*fftparams*)

Retrieve the time-series and generate a coherence spectrogram of the two given channels

```
gwsumm.data.coherence.get_coherence_spectrograms(channel_pairs, segments, config=None,
                                                  cache=None, query=True, nds=None, return_=True,
                                                  frametype=None, nproc=1, datafind_error='raise',
                                                  **fftparams)
```

Get coherence spectrograms for multiple channels

```
gwsumm.data.coherence.get_coherence_spectrum(channel_pair, segments, config=None, cache=None,
                                              query=True, nds=None, return_=True, **fftparams)
```

Retrieve the time-series and generate a coherence spectrogram of the given channel

## gwsumm.data.mathutils module

Handle arbitrary mathematical operations applied to data series

```
gwsumm.data.mathutils.get_operator(opstr)
```

```
gwsumm.data.mathutils.get_with_math(channel, segments, load_func, get_func, **ioargs)
```

Get data with optional arbitrary math definitions

### Parameters

#### channel

[*str*] name of the meta-channel to create

#### segments

[~*gwp.py.segments.SegmentList*] segments over which to create the new channel

#### load\_func

[*callable*] method to call to load data from disk

#### get\_func

[*callable*] method to call to return channel data

#### \*\*ioargs

all other kwargs are passed to the *load\_func* and *get\_func*

### Returns

#### datalist

[*TimeSeriesList*, or similar] a structured list of data objects, probably either for *TimeSeries* or *Spectrogram*

```
gwsumm.data.mathutils.parse_math_definition(definition)
```

Parse the definition for a channel combination

This method can only handle commutative operations, no fancy stuff with parentheses. Something like  $A * B$  is fine, but not  $(A + B) ^ 2$

All operands, operators, and values should be space-separated.

### Returns

#### channels

[*list of tuple*] a list of 2-tuples containing the name of each channel, and any mathematical operations to be applied to that channel only

#### operators

[*list of callable*] the list of functions that combine one channel and the previous, if *channels* is a list of length *N*, then the *operators* list will have length *N*-1



## Examples

```
>>> parse_math_definition('H1:TEST * L1:TEST^2')
[(['H1:TEST', None), ('L1:TEST', (<built-in function pow>, 2.0))],
 [<built-in function mul>]]
```

## gwsumm.data.range module

Get range data

```
gwsumm.data.range.get_range(channel, segments, config=None, cache=None, query=True, nds=None,
                             return_=True, nproc=1, datafind_error='raise', frametype=None, stride=None,
                             fftlength=None, overlap=None, method=None, **rangekwargs)
```

Calculate the sensitive distance for a given strain channel

```
gwsumm.data.range.get_range_channel(channel, **rangekwargs)
```

Return the meta-channel name used to store range data

```
gwsumm.data.range.get_range_spectrogram(channel, segments, config=None, cache=None, query=True,
                                         nds=None, return_=True, nproc=1, datafind_error='raise',
                                         frametype=None, stride=60, fftlength=None, overlap=None,
                                         method=None, **rangekwargs)
```

Estimate the spectral contribution to sensitive distance for a given strain channel

```
gwsumm.data.range.get_range_spectrum(channel, segments, config=None, cache=None, query=True,
                                     nds=None, return_=True, nproc=1, datafind_error='raise',
                                     frametype=None, stride=60, fftlength=None, overlap=None,
                                     method=None, which='all', state=None, **rangekwargs)
```

Compute percentile spectra of the range integrand from a set of spectrograms

## gwsumm.data.spectral module

Get spectrograms and spectra

```
gwsumm.data.spectral.add_spectrogram(spectrogram, key=None, coalesce=True)
```

Add a *Spectrogram* to the global memory cache

```
gwsumm.data.spectral.apply_transfer_function_series(spectrogram, tfunc)
```

Multiply a spectrogram by a transfer function *FrequencySeries*

This method interpolates the transfer function onto the frequency vector of the spectrogram, so should work regardless of the inputs

```
gwsumm.data.spectral.get_spectrogram(channel, segments, config=None, cache=None, query=True,
                                     nds=None, format='power', return_=True, frametype=None,
                                     nproc=1, datafind_error='raise', **fftparams)
```

Retrieve the time-series and generate a spectrogram of the given channel

```
gwsumm.data.spectral.get_spectrograms(channels, segments, config=None, cache=None, query=True,
                                       nds=None, format='power', return_=True, frametype=None,
                                       nproc=1, datafind_error='raise', **fftparams)
```

Get spectrograms for multiple channels

```
gwsumm.data.spectral.get_spectrum(channel, segments, config=None, cache=None, query=True, nds=None,
                                  format='power', return_=True, frametype=None, nproc=1,
                                  datafind_error='raise', state=None, **fftparams)
```

Retrieve the time-series and generate a spectrum of the given channel

```
gwsumm.data.spectral.size_for_spectrogram(size, stride, fftlength, overlap)
```

## gwsumm.data.timeseries module

Utilities for data handling and display

```
gwsumm.data.timeseries.add_timeseries(timeseries, key=None, coalesce=True)
```

Add a *TimeSeries* to the global memory cache

### Parameters

#### **timeseries**

[*TimeSeries* or *StateVector*] the data series to add

#### **key**

[*str*, optional] the key with which to store these data, defaults to the `~gwpw.timeseries.TimeSeries.name` of the series

#### **coalesce**

[*bool*, optional] coalesce contiguous series after adding, defaults to *True*

```
gwsumm.data.timeseries.all_adc(cache)
```

Returns *True* if all cache entries point to GWF file known to contain only ADC channels

This is useful to set *type='adc'* when reading with frameCPP, which can greatly speed things up.

```
gwsumm.data.timeseries.exclude_short_trend_segments(segments, ifo, frametype)
```

Remove segments from a list shorter than 1 trend sample

```
gwsumm.data.timeseries.filter_timeseries(ts, filt)
```

Filter a *TimeSeries* using a function or a ZPK definition.

```
gwsumm.data.timeseries.find_best_frames(ifo, frametype, start, end, **kwargs)
```

Find frames for the given type, replacing with a better type if needed

```
gwsumm.data.timeseries.find_frame_type(channel)
```

Find the frametype associated with the given channel

If the input channel has a *frametype* attribute, that will be used, otherwise the frametype will be guessed based on the channel name and any trend options given

```
gwsumm.data.timeseries.find_frames(ifo, frametype, gpsstart, gpsend, config=<GWSummConfigParser(>,<br>                                   urltype='file', gaps='warn', onerror='raise')
```

Query the datafind server for GWF files for the given type

### Parameters

#### **ifo**

[*str*] prefix for the IFO of interest (either one or two characters)

#### **frametype**

[*str*] name of the frametype to find

#### **gpsstart**

[*int*] GPS start time of the query

**gpsend**

[*int*] GPS end time of the query

**config**

[*~ConfigParser.ConfigParser*, optional] configuration with [*datafind*] section containing *server* specification, otherwise taken from the environment

**urltype**

[*str*, optional] what type of file paths to return, default: *file*

**gaps**

[*str*, optional] what to do when gaps are detected, one of

- *ignore* : do nothing
- *warn* : display the existence of gaps but carry on
- *raise* : raise an exception

**onerror**

[*str*, optional] what to do when the *gwdatafind* query itself fails, same options as for *gaps*

**Returns**
**cache**

[*list of str*] a list of file paths pointing at GWF files matching the request

`gwsumm.data.timeseries.frame_trend_type(ifo, frametype)`

Returns the trend type of based on the given frametype

`gwsumm.data.timeseries.get_channel_type(name)`

Returns the probable type of this channel, based on the name

**Parameters**
**name**

[*str*] the name of the channel

**Returns**
**type**

[*str*] one of 'adc', 'proc', or 'sim'

`gwsumm.data.timeseries.get_timeseries(channel, segments, config=None, cache=None, query=True, nds=None, nproc=1, frametype=None, statevector=False, return_=True, datafind_error='raise', **ioargs)`

Retrieve data for channel

**Parameters**
**channel**

[*str* or *~gwpy.detector.Channel*] the name of the channel you want

**segments**

[*~gwpy.segments.SegmentList*] the data segments of interest

**config**

[*~gwsumm.config.GWSummConfigParser*] the configuration for this analysis

**cache**

[*~glue.lal.Cache* or *list of str*] a cache of data files from which to read

**query**

[*bool*, optional] whether you want to retrieve new data from the source if it hasn't been loaded already

**nds**

[*bool*, optional] whether to try and use NDS2 for data access, default is to guess based on other arguments and the environment

**nproc**

[*int*, optional] number of parallel cores to use for file reading, default: 1

**frametype**

[*str*, optional] the frametype of the target channels, if not given, this will be guessed based on the channel name(s)

**statevector**

[*bool*, optional] whether you want to load `~gwpy.timeseries.StateVector` rather than `~gwpy.timeseries.TimeSeries` data

**datafind\_error**

[*str*, optional] what to do in the event of a datafind error, one of

- 'raise' : stop immediately upon error
- 'warn' : print warning and continue as if no frames had been found
- 'ignore' : print nothing and continue with no frames

**return\_**

[*bool*, optional] whether you actually want anything returned to you, or you are just calling this function to load data for use later

**\*\*ioargs**

all other keyword arguments are passed to the relevant data reading method (either `~gwpy.timeseries.TimeSeries.read` or `~gwpy.timeseries.TimeSeries.fetch` or state-vector equivalents)

**Returns**
**data**

[`~gwpy.timeseries.TimeSeriesList`] a list of *TimeSeries*

```
gwsumm.data.timeseries.get_timeseries_dict(channels, segments, config=<GWSummConfigParser()>,
                                           cache=None, query=True, nds=None, nproc=1,
                                           frametype=None, statevector=False, return_=True,
                                           datafind_error='raise', **ioargs)
```

Retrieve the data for a set of channels

**Parameters**
**channels**

[list of *str* or `~gwpy.detector.Channel`] the channels you want to get

**segments**

[`~gwpy.segments.SegmentList`] the data segments of interest

**config**

[`~gwsumm.config.GWSummConfigParser`] the configuration for this analysis

**query**

[*bool*, optional] whether you want to retrieve new data from the source if it hasn't been loaded already

**nds**

[*bool*, optional] whether to try and use NDS2 for data access, default is to guess based on other arguments and the environment

**nproc**

[*int*, optional] number of parallel cores to use for file reading, default: 1

**frametype**

[*str*, optional] the frametype of the target channels, if not given, this will be guessed based on the channel name(s)

**statevector**

[*bool*, optional] whether you want to load `~gwp.py.timeseries.StateVector` rather than `~gwp.py.timeseries.TimeSeries` data

**datafind\_error**

[*str*, optional] what to do in the event of a datafind error, one of

- ‘raise’ : stop immediately upon error
- ‘warn’ : print warning and continue as if no frames had been found
- ‘ignore’ : print nothing and continue with no frames

**return\_**

[*bool*, optional] whether you actually want anything returned to you, or you are just calling this function to load data for use later

**\*\*ioargs**

all other keyword arguments are passed to the relevant data reading method (either `~gwp.py.timeseries.TimeSeriesDict.read` or `~gwp.py.timeseries.TimeSeriesDict.fetch` or state-vector equivalents)

**Returns**
**datalist**

[*dict* of `~gwp.py.timeseries.TimeSeriesList`] a set of (*channel*, *TimeSeriesList*) pairs

`gwsumm.data.timeseries.locate_data(channels, segments, list_class=<class 'gwp.py.timeseries.timeseries.TimeSeriesList'>)`

Find and return available (already loaded) data

`gwsumm.data.timeseries.resample_timeseries_dict(tsd, nproc=1, **sampling_dict)`

Resample a *TimeSeriesDict*

**Parameters**
**tsd**

[`~gwp.py.timeseries.TimeSeriesDict`] the input dict to resample

**nproc**

[*int*, optional] the number of parallel processes to use

**\*\*sampling\_dict**

<name>=<sampling frequency> pairs defining new sampling frequencies for keys of *tsd*

**Returns**
**resampled**

[`~gwp.py.timeseries.TimeSeriesDict`] a new dict with the keys from *tsd* and resampled values, if that key was included in *sampling\_dict*, or the original value

`gwsumm.data.timeseries.sieve_cache(cache, ifo=None, tag=None, segment=None)`

## gwsumm.data.utils module

Utilities for data loading and pre-processing

**class** gwsumm.data.utils.FftParams(\*\*kwargs)

Bases: `object`

Convenience object to hold signal-processing parameters

**dict()**

**fftlength**

**method**

**overlap**

**scheme**

**stride**

**window**

gwsumm.data.utils.get\_fftparams(channel, \*\*defaults)

gwsumm.data.utils.make\_globalv\_key(channels, fftparams=None)

Generate a unique key for storing data in a globalv dict

### Parameters

**channels**

[*str*, *list*] one or more channels to group in this key

**fftparams**

[*FftParams*] structured set of signal-processing parameters used to generate the dataset

gwsumm.data.utils.use\_configparser(*f*)

Decorate a method to use a valid default for 'config'

This is just to allow lazy passing of *config=None*

gwsumm.data.utils.use\_segmentlist(*f*)

Decorate a method to convert incoming segments into a *SegmentList*

This assumes that the method to be decorated takes a segment list as the second positional argument.

## Module contents

Methods and classes for loading and pre-processing data

Each of the sub-modules are designed to read or create the data requested only once, with the containers from the *globalv* module used as a storage buffer for each unique data type

## gwsumm.html package

### Subpackages

## gwsumm.html.tests package

### Submodules

#### gwsumm.html.tests.test\_bootstrap module

Unit tests for gwsumm.html.bootstrap

`gwsumm.html.tests.test_bootstrap.test_banner()`

`gwsumm.html.tests.test_bootstrap.test_base_map_dropdown()`

`gwsumm.html.tests.test_bootstrap.test_calendar(mode, datefmt)`

`gwsumm.html.tests.test_bootstrap.test_calendar_no_mode()`

`gwsumm.html.tests.test_bootstrap.test_state_switcher()`

`gwsumm.html.tests.test_bootstrap.test_wrap_content()`

#### gwsumm.html.tests.test\_html5 module

Unit tests for gwsumm.html.html5

`gwsumm.html.tests.test_html5.test_comments_box()`

`gwsumm.html.tests.test_html5.test_dialog_box(tmpdir)`

`gwsumm.html.tests.test_html5.test_expand_path()`

`gwsumm.html.tests.test_html5.test_ldvw_qscan_batch()`

`gwsumm.html.tests.test_html5.test_ldvw_qscan_single()`

`gwsumm.html.tests.test_html5.test_load()`

`gwsumm.html.tests.test_html5.test_load_custom()`

`gwsumm.html.tests.test_html5.test_load_state()`

`gwsumm.html.tests.test_html5.test_overlay_canvas()`

## gwsumm.html.tests.test\_static module

Unit tests for gwsumm.html.static

gwsumm.html.tests.test\_static.test\_get\_css()

gwsumm.html.tests.test\_static.test\_get\_js()

## Module contents

Unit tests for gwsumm.html

## Submodules

### gwsumm.html.bootstrap module

Helper functions for twitter-bootstrap HTML constructs.

gwsumm.html.bootstrap.**banner**(title, subtitle=None, titleclass=None, subtitleclass=None)

Construct a banner heading in bootstrap format

#### Parameters

##### title

[*str*] name of page (<h1>)

##### subtitle

[*str*, optional] description of page (<p>)

##### titleclass

[*str*, optional] class option for <h1>

##### subtitleclass

[*str*, optional] class option for <p>

#### Returns

##### banner

[~MarkupPy.markup.page] markup.py page instance

gwsumm.html.bootstrap.**base\_map\_dropdown**(this, id\_=None, bases={})

Construct a dropdown menu that links to a version of the current page on another server, based on a new base.

gwsumm.html.bootstrap.**calendar**(date, tag='a', class\_='nav-link dropdown-toggle', id\_='calendar', dateformat=None, mode=None)

Construct a bootstrap-datepicker calendar.

#### Parameters

##### date

[**:class: `datetime.datetime`**, **:class: `datetime.date`**] active date for the calendar

##### tag

[*str*] type of enclosing HTML tag, default: <a>

#### Returns



**calendar**

[*list*] a list of three oneliner strings of HTML containing the calendar text and a triggering dropdown

`gwsumm.html.bootstrap.state_switcher(states, default=0)`

Build a state switch button, including all of the given states, with the default selected by index

`gwsumm.html.bootstrap.wrap_content(page)`

Utility to wrap some HTML into the relevant <div>s for the main body of a page in bootstrap format

**Parameters****page**

[*:class:* `~MarkupPy.markup.page`, *str*] HTML content to be wrapped

**span**

[*int*] column span of content, default: 'full' (12)

**Returns****wrappedpage**

[*:class:* `~MarkupPy.markup.page`] A new *page* with the input content wrapped as

```
<div class="container">
</div>
```

**gwsumm.html.html5 module**

HTML5-specific extensions

`gwsumm.html.html5.comments_box(name, identifier=None, title=None, url=None)`

Generate a Disqus comments box

`gwsumm.html.html5.dialog_box(content, title, id_, btntxt)`

Generate a dialog box to be loaded modal atop the main page

**Parameters****content**

[*str*] either raw markdown text or the path to a file containing markdown, this will be rendered in HTML as the contents of the dialog box

**title**

[*str*] title to display atop the dialog box

**id\_**

[*str*] unique identifier for the dialog box

**btntxt**

[*str*] text (usually a single character) to appear inside a sticky button that opens the dialog box

**Returns****page**

[*~MarkupPy.markup.page*] fully rendered HTML containing the dialog box

`gwsumm.html.html5.ldvw_qscan(channel, time, fmin=10, fmax='inf', qmin=4, qmax=100)`

Generate a Q-scan through LIGO DataViewer Web (LDVW)

`gwsumm.html.html5.load(url, id_='main', error=False, success=None)`

Construct the HTML script required to load a url into the HTML element with the given unique `id_`.

`gwsumm.html.html5.load_state(url)`

Construct the HTML script required to load the Tab HTML for a given `:class:`~gwsumm.state.core.SummaryState``

#### Parameters

**url**

[*str*] path (relative to <base>) of HTML to load

**id\_**

[*str*, optional, default: '#main'] <div> 'id' in which to load HTML

#### Returns

**HTML**

[*str*] HTML one-liner with script loading

`gwsumm.html.html5.overlay_canvas()`

Generate a dialog box allowing users to select and overlay plots

#### Returns

**page**

[*MarkupPy.markup.page*] fully rendered HTML containing the dialog box

## **gwsumm.html.static module**

HTML <head> helpers

This module mainly declares the resources used by standard on HTML pages

`gwsumm.html.static.get_css()`

Return a *dict* of CSS files to link in the HTML <head>

`gwsumm.html.static.get_js()`

Return a *dict* of javascript files to link in the HTML <head>

## **Module contents**

HTML helpers

HTML output is built upon the [markup.py module](#) and primarily formatted to fit the [twitter bootstrap library](#).

## **gwsumm.plot package**

### **Subpackages**

#### **gwsumm.plot.guardian package**

### **Subpackages**

## gwsumm.plot.guardian.tests package

### Submodules

#### gwsumm.plot.guardian.tests.test\_main module

Tests for the *gwsumm.plot.guardian* command-line interface

```
gwsumm.plot.guardian.tests.test_main.test_main(tmpdir, caplog)
```

### Module contents

Unit tests for *gwsumm.plot.guardian*

### Submodules

#### gwsumm.plot.guardian.core module

Plots of Guardian data

```
class gwsumm.plot.guardian.core.GuardianStatePlot(*args, **kwargs)
```

Bases: *SegmentDataPlot*

```
defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'color': None,
'contour.algorithm': 'mpl2014', 'figure.figsize': [12.0, 6.0], 'figure.labelsize':
'large', 'figure.labelweight': 'normal', 'font.sans-serif': ['Roboto'],
'grid.alpha': 0.5, 'grid.linewidth': 0.5, 'insetlabels': 'inset',
'legend-bbox_to_anchor': (1.0, 1.0), 'legend-borderaxespad': 0, 'legend-fontsize':
12, 'legend-frameon': False, 'legend-handletextpad': 0.5, 'legend-loc': 'upper
left', 'legend.edgecolor': 'inherit', 'legend.fancybox': False, 'legend.fontsize':
10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2, 'legend_bbox_to_anchor':
(1.0, 1.0), 'legend_borderaxespad': 0.0, 'legend_fontsize': 12, 'legend_frameon':
False, 'legend_loc': 'upper left', 'linewidth': 0.5, 'mask': None, 'mathtext.bf':
'Roboto', 'mathtext.cal': 'Calligraffiti', 'mathtext.fontset': 'custom',
'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf': 'Roboto',
'mathtext.tt': 'Roboto Slab', 'nominalcolor': '#ffb200', 'on-is-bad': False,
'requestcolor': '#0066ff', 'savefig.transparent': True, 'svg.fonttype': 'none',
'text.parse_math': True, 'xscale': 'auto-gps', 'xtick.labelsize': 14.0, 'yscale':
'linear', 'ytick.labelsize': 10}
```

dict of default plotting kwargs

**draw()**

Read in all necessary data, and generate the figure.

**property info**

**property node**

**type = 'guardian'**

name for DataPlot subclass

## Module contents

Submodule for plots of Guardian data

### gwsumm.plot.triggers package

#### Subpackages

#### gwsumm.plot.triggers.tests package

#### Submodules

##### gwsumm.plot.triggers.tests.test\_main module

Tests for the *gwsumm.plot.triggers* command-line interface

`gwsumm.plot.triggers.tests.test_main.test_main(dqflag, tmpdir, caplog)`

`gwsumm.plot.triggers.tests.test_main.test_main_invalid_columns(capsys)`

`gwsumm.plot.triggers.tests.test_main.test_main_with_cache_and_tiles(tmpdir, caplog)`

## Module contents

Unit tests for *gwsumm.plot.triggers*

#### Submodules

##### gwsumm.plot.triggers.core module

Definitions for event trigger plots

```
class gwsumm.plot.triggers.core.TriggerDataPlot(channels, start, end, state=None, outdir='.',  
                                              etg=None, **kwargs)
```

Bases: *TriggerPlotMixin, TimeSeriesDataPlot*

Standard event trigger plot

```
add_loudest_event(ax, table, rank, *columns, **kwargs)
```

```
data = 'triggers'
```

```
defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'clim': None, 'cmap': 'YlGnBu',
'color': None, 'colorlabel': None, 'contour.algorithm': 'mpl2014', 'edgecolor':
'face', 'facecolor': None, 'figure.figsize': [12.0, 6.0], 'figure.labelsize':
'large', 'figure.labelweight': 'normal', 'font.sans-serif': ['Roboto'],
'grid.alpha': 0.5, 'grid.linewidth': 0.5, 'legend.edgecolor': 'inherit',
'legend.fancybox': False, 'legend.fontsize': 10.0, 'legend.handlelength': 1.0,
'legend.numpoints': 2, 'logcolor': False, 'marker': 'o', 'mathtext.bf':
'Roboto', 'mathtext.cal': 'Calligraphiti', 'mathtext.fontset': 'custom',
'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf': 'Roboto',
'mathtext.tt': 'Roboto Slab', 's': 20, 'savefig.transparent': True,
'svg.fonttype': 'none', 'text.parse_math': True, 'vmax': None, 'vmin': None,
'x': 'time', 'xscale': 'auto-gps', 'xtick.labelsize': 14.0, 'y': 'snr',
'yscale': 'linear', 'ytick.labelsize': 14.0}
```

dict of default plotting kwargs

**draw()**

Read in all necessary data, and generate the figure.

**property pid**

Unique identifier for this *TriggerDataPlot*.

Extends the standard *TimeSeriesDataPlot* pid with the ETG and each of the column names.

**type = 'triggers'**

name for DataPlot subclass

```
class gwsumm.plot.triggers.core.TriggerHistogramPlot(*args, **kwargs)
```

Bases: *TriggerPlotMixin*, *TimeSeriesHistogramPlot*

HistogramPlot from a LIGO\_LW Table

**data = 'triggers'**

**draw()**

Get data and generate the figure.

**property pid**

**type = 'trigger-histogram'**

name for DataPlot subclass

```
class gwsumm.plot.triggers.core.TriggerPlotMixin(*args, **kwargs)
```

Bases: *object*

Mixin to overwrite *channels* property for trigger plots

We don't need to get channel data for trigger plots.

**property allchannels**

List of all unique channels for this plot

**property pid**

```
class gwsumm.plot.triggers.core.TriggerRateDataPlot(*args, **kwargs)
```

Bases: [TriggerPlotMixin](#), [TimeSeriesDataPlot](#)

TimeSeriesDataPlot of trigger rate.

**data** = 'triggers'

```
defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'column': None,
'contour.algorithm': 'mpl2014', 'figure.figsize': [12.0, 6.0], 'figure.labelsize':
'large', 'figure.labelweight': 'normal', 'font.sans-serif': ['Roboto'],
'grid.alpha': 0.5, 'grid.linewidth': 0.5, 'legend-bbox_to_anchor': (1.0, 1.0),
'legend-frameon': False, 'legend-loc': 'upper left', 'legend-markerscale': 3,
'legend.edgecolor': 'inherit', 'legend.fancybox': False, 'legend.fontsize': 10.0,
'legend.handlelength': 1.0, 'legend.numpoints': 2, 'mathtext.bf': 'Roboto',
'mathtext.cal': 'Calligraffiti', 'mathtext.fontset': 'custom', 'mathtext.it':
'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf': 'Roboto', 'mathtext.tt':
'Roboto Slab', 'savefig.transparent': True, 'svg.fonttype': 'none',
'text.parse_math': True, 'xscale': 'auto-gps', 'xtick.labelsize': 14.0, 'ylabel':
'Rate [Hz]', 'yscale': 'linear', 'ytick.labelsize': 14.0}
```

dict of default plotting kwargs

**draw()**

Read in all necessary data, and generate the figure.

**property pid**

**type** = 'trigger-rate'

name for DataPlot subclass

```
class gwsumm.plot.triggers.core.TriggerTimeSeriesDataPlot(*args, **kwargs)
```

Bases: [TimeSeriesDataPlot](#)

Custom time-series plot to handle discontinuous *TimeSeries*.

**data** = 'triggers'

**draw()**

Read in all necessary data, and generate the figure.

**type** = 'trigger-timeseries'

name for DataPlot subclass

## Module contents

Submodule for plots of event triggers

## Submodules

### gwsumm.plot.builtin module

Definitions for the standard plots

```
class gwsumm.plot.builtin.CoherenceSpectrogramDataPlot(*args, **kwargs)
```

Bases: [SpectrogramDataPlot](#)

DataPlot a Spectrogram of the coherence between two channels

**data** = 'coherence-spectrogram'

```
defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'clim': None, 'colorlabel':
None, 'contour.algorithm': 'mpl2014', 'figure.figsize': [12.0, 6.0],
'figure.labelsize': 'large', 'figure.labelweight': 'normal', 'font.sans-serif':
['Roboto'], 'format': None, 'grid.alpha': 0.5, 'grid.linewidth': 0.5,
'legend.edgecolor': 'inherit', 'legend.fancybox': False, 'legend.fontsize': 10.0,
'legend.handlelength': 1.0, 'legend.numpoints': 2, 'logcolor': False,
'mathtext.bf': 'Roboto', 'mathtext.cal': 'Calligraffiti', 'mathtext.fontset':
'custom', 'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf':
'Roboto', 'mathtext.tt': 'Roboto Slab', 'rasterized': True, 'ratio': None,
'savefig.transparent': True, 'svg.fonttype': 'none', 'text.parse_math': True,
'xscale': 'auto-gps', 'xtick.labelsize': 14.0, 'ylabel': 'Frequency [Hz]',
'yscale': 'log', 'ytick.labelsize': 14.0}
```

dict of default plotting kwargs

**type** = 'coherence-spectrogram'

name for DataPlot subclass

```
class gwsumm.plot.builtin.CoherenceSpectrumDataPlot(channels, start, end, state=None, outdir='',
tag=None, pid=None, href=None, new=True,
all_data=False, read=True, fileformat='png',
caption=None, **pargs)
```

Bases: [SpectrumDataPlot](#)

Coherence pectrum plot for a *SummaryTab*

**data** = 'coherence-spectrogram'

```
defaults = {'alpha': 0.1, 'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'contour.algorithm': 'mpl2014',
'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large', 'figure.labelweight':
'normal', 'font.sans-serif': ['Roboto'], 'format': None, 'grid.alpha': 0.5,
'grid.linewidth': 0.5, 'legend.edgecolor': 'inherit', 'legend.fancybox': False,
'legend.fontsize': 10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2,
'mathtext.bf': 'Roboto', 'mathtext.cal': 'Calligraffiti', 'mathtext.fontset':
'custom', 'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf':
'Roboto', 'mathtext.tt': 'Roboto Slab', 'no-percentiles': False,
'reference_linestyle': '--', 'savefig.transparent': True, 'svg.fonttype': 'none',
'text.parse_math': True, 'xscale': 'log', 'xtick.labelsize': 14.0, 'yscale':
'linear', 'ytick.labelsize': 14.0, 'zorder': 1}
```

dict of default plotting kwargs

**get\_channel\_groups()**

Hi-jacked method to return pairs of channels

For the *CoherenceSpectrumDataPlot* this method is only used in determining how to separate lists of plotting argument given by the user.

**type = 'coherence-spectrum'**

name for DataPlot subclass

**class** gwsumm.plot.builtin.**RayleighSpectrogramDataPlot**(\*args, \*\*kwargs)

Bases: *SpectrogramDataPlot*

Rayleigh statistic versino of *SpectrogramDataPlot*

**data = 'rayleigh-spectrogram'**

```
defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'clim': [0.25, 4], 'cmap':
'BrBG_r', 'colorlabel': 'Rayleigh statistic', 'contour.algorithm': 'mpl2014',
'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large', 'figure.labelweight':
'normal', 'font.sans-serif': ['Roboto'], 'format': 'rayleigh', 'grid.alpha': 0.5,
'grid.linewidth': 0.5, 'legend.edgecolor': 'inherit', 'legend.fancybox': False,
'legend.fontsize': 10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2,
'mathtext.bf': 'Roboto', 'mathtext.cal': 'Calligraffiti', 'mathtext.fontset':
'custom', 'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf':
'Roboto', 'mathtext.tt': 'Roboto Slab', 'rasterized': True, 'ratio': None,
'savefig.transparent': True, 'svg.fonttype': 'none', 'text.parse_math': True,
'xscale': 'auto-gps', 'xtick.labelsize': 14.0, 'ylabel': 'Frequency [Hz]',
'yscale': 'log', 'ytick.labelsize': 14.0}
```

dict of default plotting kwargs

**type = 'rayleigh-spectrogram'**

name for DataPlot subclass

**class** gwsumm.plot.builtin.**RayleighSpectrumDataPlot**(channels, start, end, state=None, outdir='',  
tag=None, pid=None, href=None, new=True,  
all\_data=False, read=True, fileformat='png',  
caption=None, \*\*pargs)

Bases: *SpectrumDataPlot*

Rayleigh statistic versino of *SpectrumDataPlot*

**data = 'rayleigh-spectrum'**

```
defaults = {'alpha': 0.1, 'format': 'rayleigh', 'no-percentiles': True,
'reference-linestyle': '--', 'xscale': 'log', 'yscale': 'log', 'zorder': 1}
```

dict of default plotting kwargs

**type = 'rayleigh-spectrum'**

name for DataPlot subclass



```

class gwsumm.plot.builtin.SpectralVarianceDataPlot(channels, *args, **kwargs)
    Bases: SpectrumDataPlot

    SpectralVariance histogram plot for a DataTab

    data = 'spectrogram'

    defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
                'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
                4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
                'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'contour.algorithm': 'mpl2014',
                'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large', 'figure.labelweight':
                'normal', 'font.sans-serif': ['Roboto'], 'format': None, 'grid.alpha': 0.5,
                'grid.linewidth': 0.5, 'legend.edgecolor': 'inherit', 'legend.fancybox': False,
                'legend.fontsize': 10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2,
                'log': True, 'mathtext.bf': 'Roboto', 'mathtext.cal': 'Calligraffiti',
                'mathtext.fontset': 'custom', 'mathtext.it': 'Roboto:italic', 'mathtext.rm':
                'Roboto', 'mathtext.sf': 'Roboto', 'mathtext.tt': 'Roboto Slab', 'nbins': 100,
                'no-percentiles': False, 'reference-linestyle': '--', 'savefig.transparent':
                True, 'svg.fonttype': 'none', 'text.parse_math': True, 'xscale': 'log',
                'xtick.labelsize': 14.0, 'yscale': 'log', 'ytick.labelsize': 14.0, 'zorder': 1}

        dict of default plotting kwargs

    parse_variance_kwargs()

    type = 'variance'

        name for DataPlot subclass

class gwsumm.plot.builtin.SpectrogramDataPlot(*args, **kwargs)
    Bases: TimeSeriesDataPlot

    DataPlot a Spectrogram

    data = 'spectrogram'

    defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
                'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
                4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
                'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'contour.algorithm': 'mpl2014',
                'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large', 'figure.labelweight':
                'normal', 'font.sans-serif': ['Roboto'], 'format': None, 'grid.alpha': 0.5,
                'grid.linewidth': 0.5, 'legend.edgecolor': 'inherit', 'legend.fancybox': False,
                'legend.fontsize': 10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2,
                'mathtext.bf': 'Roboto', 'mathtext.cal': 'Calligraffiti', 'mathtext.fontset':
                'custom', 'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf':
                'Roboto', 'mathtext.tt': 'Roboto Slab', 'rasterized': True, 'ratio': None,
                'savefig.transparent': True, 'svg.fonttype': 'none', 'text.parse_math': True,
                'xscale': 'auto-gps', 'xtick.labelsize': 14.0, 'ylabel': 'Frequency [Hz]',
                'yscale': 'log', 'ytick.labelsize': 14.0}

        dict of default plotting kwargs

    draw()

        Read in all necessary data, and generate the figure.

    get_ratio(specgrams)

    property pid

```

**type** = 'spectrogram'

name for DataPlot subclass

```
class gwsumm.plot.builtin.SpectrumDataPlot(channels, start, end, state=None, outdir='.', tag=None,
                                             pid=None, href=None, new=True, all_data=False,
                                             read=True, fileformat='png', caption=None, **pargs)
```

Bases: [DataPlot](#)

Spectrum plot for a *SummaryTab*

**data** = 'spectrum'

```
defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'contour.algorithm': 'mpl2014',
'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large', 'figure.labelweight':
'normal', 'font.sans-serif': ['Roboto'], 'format': None, 'grid.alpha': 0.5,
'grid.linewidth': 0.5, 'legend.edgecolor': 'inherit', 'legend.fancybox': False,
'legend.fontsize': 10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2,
'mathtext.bf': 'Roboto', 'mathtext.cal': 'Calligraffiti', 'mathtext.fontset':
'custom', 'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf':
'Roboto', 'mathtext.tt': 'Roboto Slab', 'no-percentiles': False,
'reference-linestyle': '--', 'savefig.transparent': True, 'svg.fonttype': 'none',
'text.parse_math': True, 'xscale': 'log', 'xtick.labelsize': 14.0, 'yscale':
'log', 'ytick.labelsize': 14.0, 'zorder': 1}
```

dict of default plotting kwargs

**draw()**

Process all data and generate the output file for this *SummaryPlot*.

This function should be provided by all sub-classes, and should take no arguments.

**parse\_references**(*prefix*='reference(\d+)?\Z')

Parse parameters for displaying one or more reference traces

**type** = 'spectrogram'

name for DataPlot subclass

```
class gwsumm.plot.builtin.TimeSeriesDataPlot(*args, **kwargs)
```

Bases: [DataLabelSvgMixin](#), [DataPlot](#)

DataPlot of some *TimeSeries* data.

**add\_future\_shade**(*gps*=None, *facecolor*='gray', *alpha*=0.1, \*\*kwargs)

Shade those parts of the figure that display times in the future

**add\_state\_segments**(*ax*, *visible*=None, \*\*kwargs)

Add an *Axes* below the given *ax* displaying the *SummaryState* for this *TimeSeriesDataPlot*.

#### Parameters

**ax**

[*Axes*] the set of *Axes* below which to display the state segments.

**visible**

[*bool*, optional] whether or not to display the axes, or just make space for them, default is *None*, meaning a dynamic choice based on the state

**\*\*kwargs**  
 other keyword arguments will be passed to the `meth:~gwpy.plot.Plot.add_segments_bar` method.

**data = 'timeseries'**

**defaults = {'animation.convert\_args': ['-layers', 'OptimizePlus'],**  
**'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,**  
**4], 'axes.formatter.use\_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,**  
**'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'contour.algorithm': 'mpl2014',**  
**'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large', 'figure.labelweight':**  
**'normal', 'font.sans-serif': ['Roboto'], 'grid.alpha': 0.5, 'grid.linewidth':**  
**0.5, 'legend.edgecolor': 'inherit', 'legend.fancybox': False, 'legend.fontsize':**  
**10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2, 'mathtext.bf': 'Roboto',**  
**'mathtext.cal': 'Calligraffiti', 'mathtext.fontset': 'custom', 'mathtext.it':**  
**'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf': 'Roboto', 'mathtext.tt':**  
**'Roboto Slab', 'savefig.transparent': True, 'svg.fonttype': 'none',**  
**'text.parse\_math': True, 'xscale': 'auto-gps', 'xtick.labelsize': 14.0, 'yscale':**  
**'linear', 'ytick.labelsize': 14.0}**

dict of default plotting kwargs

**draw(outputfile=None)**

Read in all necessary data, and generate the figure.

**init\_plot(\*args, \*\*kwargs)**

Initialise the Figure and Axes objects for this *TimeSeriesDataPlot*.

**type = 'timeseries'**

name for DataPlot subclass

**class gwsumm.plot.builtin.TimeSeriesHistogram2dDataPlot(\*args, \*\*kwargs)**

Bases: *TimeSeriesHistogramPlot*

DataPlot of the 2D histogram of two *TimeSeries*.

**data = 'timeseries'**

**defaults = {'alpha': None, 'animation.convert\_args': ['-layers', 'OptimizePlus'],**  
**'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,**  
**4], 'axes.formatter.use\_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,**  
**'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'bins': 100, 'bottom': 1e-300,**  
**'cmap': 'inferno\_r', 'contour.algorithm': 'mpl2014', 'edgecolors': 'None',**  
**'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large', 'figure.labelweight':**  
**'normal', 'font.sans-serif': ['Roboto'], 'grid': 'both', 'grid.alpha': 0.5,**  
**'grid.linewidth': 0.5, 'histtype': 'stepfilled', 'legend.edgecolor': 'inherit',**  
**'legend.fancybox': False, 'legend.fontsize': 10.0, 'legend.handlelength': 1.0,**  
**'legend.numpoints': 2, 'log': True, 'mathtext.bf': 'Roboto', 'mathtext.cal':**  
**'Calligraffiti', 'mathtext.fontset': 'custom', 'mathtext.it': 'Roboto:italic',**  
**'mathtext.rm': 'Roboto', 'mathtext.sf': 'Roboto', 'mathtext.tt': 'Roboto Slab',**  
**'normed': True, 'rwidth': 1, 'savefig.transparent': True, 'shading': 'flat',**  
**'svg.fonttype': 'none', 'text.parse\_math': True, 'xtick.labelsize': 14.0,**  
**'ylabel': 'Rate [Hz]', 'yscale': 'linear', 'ytick.labelsize': 14.0}**

dict of default plotting kwargs

**draw(outputfile=None)**

Get data and generate the figure.

```
parse_hist_kwargs(**defaults)

parse_pcmesh_kwargs(**defaults)

type = 'histogram2d'
    name for DataPlot subclass

class gwsumm.plot.builtin.TimeSeriesHistogramPlot(channels, start, end, state=None, outdir='.',
                                                    tag=None, pid=None, href=None, new=True,
                                                    all_data=False, read=True, fileformat='png',
                                                    caption=None, **pargs)

Bases: DataPlot

HistogramPlot from a Series

data = 'timeseries'

defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'bottom': 1e-300,
'contour.algorithm': 'mpl2014', 'figure.figsize': [12.0, 6.0], 'figure.labelsize':
'large', 'figure.labelweight': 'normal', 'font.sans-serif': ['Roboto'],
'grid.alpha': 0.5, 'grid.linewidth': 0.5, 'histtype': 'stepfilled',
'legend.edgecolor': 'inherit', 'legend.fancybox': False, 'legend.fontsize': 10.0,
'legend.handlelength': 1.0, 'legend.numpoints': 2, 'log': True, 'mathtext.bf':
'Roboto', 'mathtext.cal': 'Calligraffiti', 'mathtext.fontset': 'custom',
'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf': 'Roboto',
'mathtext.tt': 'Roboto Slab', 'rwidth': 1, 'savefig.transparent': True,
'svg.fonttype': 'none', 'text.parse_math': True, 'xtick.labelsize': 14.0,
'ylabel': 'Rate [Hz]', 'ytick.labelsize': 14.0}
    dict of default plotting kwargs

draw(outputfile=None)
    Get data and generate the figure.

init_plot(geometry=None, **kwargs)
    Initialise the Figure and Axes objects for this TimeSeriesDataPlot.

parse_plot_kwargs(**defaults)
    Pop keyword arguments for Axes.plot from the pargs for this Plot

type = 'histogram'
    name for DataPlot subclass

gwsumm.plot.builtin.undo_demodulation(spec, channel, limits=None)
```

## gwsumm.plot.core module

Parse, define, and generate plots as requests through the configuration for GWSumm

```
class gwsumm.plot.core.DataPlot(channels, start, end, state=None, outdir='.', tag=None, pid=None,  
                                href=None, new=True, all_data=False, read=True, fileformat='png',  
                                caption=None, **pargs)
```

Bases: [SummaryPlot](#)

A *SummaryPlot* from instrumental data.

### Parameters

#### **channels**

[*list*] a list of channel names that define the data sources for this *DataPlot*

#### **start**

[*float*] GPS start time of this *DataPlot*.

#### **end**

[*float*] GPS end time of this *DataPlot*.

#### **tag**

[*str*] a descriptive tag for this *DataPlot*, used as part of the output file name

#### **outdir**

[*str*] output directory path for this *DataPlot*, defaults to the current directory

#### **href**

[*str*] custom URL for this plot to link towards.

#### **\*\*kwargs**

all other keyword arguments to be passed to this plot's `:meth:`process`` method.

## Notes

All sub-classes of this object must provide the following methods

<code>:meth:`add_data_source`</code>	routine for appending data sources to the plot
--------------------------------------	------------------------------------------------

```
DRAW_PARAMS = ['imshow', 'origin', 'vmin', 'vmax', 'alpha', 'color', 'normed',  
'log', 'weights', 'orientation', 'marker', 'density', 'bottom', 'label',  
'interpolation', 'rwidth', 'aspect', 'cmap', 'range', 'histtype', 'zorder', 'norm',  
's', 'linestyle', 'cumulative', 'linewidth', 'stacked', 'extent', 'bins',  
'rasterized', 'logbins', 'align']
```

list of parameters parsed for *plot()* calls

**add\_channel**(*channel*)

**add\_hvlines**()

Add horizontal and vertical lines to this *DataPlot*

These should be defined in the configuration via the *hline* and *vline* keys.

**property allchannels**

List of all unique channels for this plot

**apply\_parameters**(\*axes, \*\*pargs)

**property channels**

List of data-source :class:`Channels <~gwp.py.detector.channel.Channel>` for this *DataPlot*.

**Type**

:class:`~gwp.py.detector.channel.ChannelList`

```
defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'contour.algorithm': 'mpl2014',
'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large', 'figure.labelweight':
'normal', 'font.sans-serif': ['Roboto'], 'grid.alpha': 0.5, 'grid.linewidth':
0.5, 'legend.edgecolor': 'inherit', 'legend.fancybox': False, 'legend.fontsize':
10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2, 'mathtext.bf': 'Roboto',
'mathtext.cal': 'Calligraffiti', 'mathtext.fontset': 'custom', 'mathtext.it':
'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf': 'Roboto', 'mathtext.tt':
'Roboto Slab', 'savefig.transparent': True, 'svg.fonttype': 'none',
'text.parse_math': True, 'xtick.labelsize': 14.0, 'ytick.labelsize': 14.0}
```

dict of default plotting kwargs

**draw()**

Process all data and generate the output file for this *SummaryPlot*.

This function should be provided by all sub-classes, and should take no arguments.

**property end**

**finalize**(outputfile=None, close=True, \*\*savekwargs)

Save the plot to disk and close.

**classmethod from\_ini**(config, section, start, end, channels=None, \*\*kwargs)

Define a new *DataPlot*.

**get\_channel\_groups()**

Find and group (mean, min, max) sets of channels for plotting.

**Returns**

**groups**

[list of tuple] list of (channelname, channellist) tuples giving core channel name and an ordered list of channels. Ordering in preference of 'rms', 'mean', 'min', 'max'.

## Notes

This method used to return an *OrderedDict*, but was changed to return a *list* of *tuple* to enable plotting a channel multiple times on a plot, for whatever reason.

**property href**

HTML <img> href attribute for this *SummaryPlot*.

**property ifos**

Interferometer set for this *DataPlot*

**init\_plot**(data=[], FigureClass=<class 'gwp.py.plot.plot.Plot'>, geometry=(1, 1), projection='rectilinear', sharex=True, sharey=True, \*\*kwargs)

Initialise the Figure and Axes objects for this *DataPlot*.

**property logx**

**property logy**

**property outputfile**

Output file for this *DataPlot*.

**parse\_legend\_kwargs**(\*\*defaults)

Pop the legend arguments from the *pargs* for this *Plot*

**parse\_list**(prefix, \*\*defaults)

Parse a list of something from parameters

This enables listing `hline`s` (for example) in the config as

```
[plot-blah] hline = 100 hline-linestyle = '-' hline-color = 'red' hline2 = 200 hline2-linestyle = '-' hline2-color = 'blue'
```

Returns an *OrderedDict* with keys matching the primary parsed value, and values as everything else, e.g.

```
{100: {'linestyle': '-', 'color': 'red'},
 200: {'linestyle': '-', 'color': 'blue'},}
```

**parse\_plot\_kwargs**(\*\*defaults)

Pop keyword arguments for *Axes.plot* from the *pargs* for this *Plot*

**parse\_rcParams**(params)

Parse matplotlib rcParams settings from a dict of plot params

**property pid**

**process**(outputfile=None, close=True)

**property span**

The GPS [start, stop) interval for this *DataPlot*.

**property start**

**property state**

~gwsomm.state.SummaryState defining validity of this *DataPlot*.

**property tag**

File tag for this *DataPlot*.

**type = 'data'**

name for *DataPlot* subclass

**class** gwsomm.plot.core.SummaryPlot(href=None, src=None, new=True, caption="")

Bases: `object`

An image to displayed in GWSumm HTML output.

**Parameters**

**href**

[str, optional] The IMG URL for this *SummaryPlot*.

**new**

[*bool*, optional] *bool* flag whether this is a new plot to be processed (*True*), of that the output already exists on disk (*False*).

## Notes

This *class* is a stub, designed to make creating detailed *SummaryPlot* classes easier.

### property caption

HTML <fancybox plot> title attribute for this *SummaryPlot*.

### classmethod from\_ini(\*args, \*\*kwargs)

Define a new *SummaryPlot* from a an INI-format *ConfigParser* section.

### property href

HTML <img> href attribute for this *SummaryPlot*.

### property new

Flag whether this is a new plot or, already exists.

Set new=False to skip actually processing this *SummaryPlot*, and just link to the outputfile.

### property src

type = None

## gwsumm.plot.mixins module

```
class gwsumm.plot.mixins.DataLabelSvgMixin(*args, **kwargs)
```

Bases: [SvgMixin](#)

```
draw_svg(outputfile)
```

```
class gwsumm.plot.mixins.SegmentLabelSvgMixin(*args, **kwargs)
```

Bases: [SvgMixin](#)

```
draw_svg(outputfile)
```

```
class gwsumm.plot.mixins.SvgMixin(*args, **kwargs)
```

Bases: [object](#)

```
abstract draw_svg(outputfile)
```

```
finalize(outputfile=None, close=True, **savekwargs)
```

```
finalize_svg(tree, outputfile, script=None)
```



## gwsumm.plot.noisebudget module

Extensions to the spectrum plot for noise budgets

```
class gwsumm.plot.noisebudget.NoiseBudgetPlot(channels, start, end, state=None, outdir='.', tag=None,
                                              pid=None, href=None, new=True, all_data=False,
                                              read=True, fileformat='png', caption=None, **pargs)
```

Bases: [SpectrumDataPlot](#)

Plot of a noise budget ASD

`data = 'spectrum'`

```
defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'contour.algorithm': 'mpl2014',
'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large', 'figure.labelweight':
'normal', 'font.sans-serif': ['Roboto'], 'format': 'asd', 'grid.alpha': 0.5,
'grid.linewidth': 0.5, 'legend.edgecolor': 'inherit', 'legend.fancybox': False,
'legend.fontsize': 10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2,
'mathtext.bf': 'Roboto', 'mathtext.cal': 'Calligraffiti', 'mathtext.fontset':
'custom', 'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf':
'Roboto', 'mathtext.tt': 'Roboto Slab', 'no-percentiles': False,
'reference-linestyle': '--', 'residual-color': 'grey', 'residual-label':
'Residual', 'residual-linestyle': ':', 'savefig.transparent': True, 'sum-color':
'black', 'sum-label': 'Sum of noises', 'sum-linestyle': '--', 'svg.fonttype':
'none', 'text.parse_math': True, 'xscale': 'log', 'xtick.labelsize': 14.0,
'yscale': 'log', 'ytick.labelsize': 14.0, 'zorder': 1}
```

dict of default plotting kwargs

`parse_residual_params(**defaults)`

`parse_sum_params(**defaults)`

`type = 'noise-budget'`

name for DataPlot subclass

```
class gwsumm.plot.noisebudget.RelativeNoiseBudgetPlot(channels, start, end, state=None, outdir='.',
                                                       tag=None, pid=None, href=None, new=True,
                                                       all_data=False, read=True, fileformat='png',
                                                       caption=None, **pargs)
```

Bases: [SpectrumDataPlot](#)

Spectrum plot for a *SummaryTab*

`data = 'spectrum'`

```
defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'contour.algorithm': 'mpl2014',
'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large', 'figure.labelweight':
'normal', 'font.sans-serif': ['Roboto'], 'format': 'asd', 'grid.alpha': 0.5,
'grid.linewidth': 0.5, 'legend.edgecolor': 'inherit', 'legend.fancybox': False,
'legend.fontsize': 10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2,
'mathtext.bf': 'Roboto', 'mathtext.cal': 'Calligraffiti', 'mathtext.fontset':
'custom', 'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf':
'Roboto', 'mathtext.tt': 'Roboto Slab', 'no-percentiles': False,
'reference-linestyle': '--', 'savefig.transparent': True, 'svg.fonttype': 'none',
'text.parse_math': True, 'xscale': 'log', 'xtick.labelsize': 14.0, 'yscale':
'log', 'ytick.labelsize': 14.0, 'zorder': 1}
```

dict of default plotting kwargs

**type** = 'noise-budget-ratio'

name for DataPlot subclass

## gwsumm.plot.range module

Definitions for range plots

**class** gwsumm.plot.range.GWpyTimeVolumeDataPlot(\*args, \*\*kwargs)

Bases: [RangePlotMixin](#), [SimpleTimeVolumeDataPlot](#)

TimeVolumeDataPlot where the range is calculated on-the-fly

**type** = 'strain-time-volume'

name for DataPlot subclass

**class** gwsumm.plot.range.RangeCumulativeHistogramPlot(\*args, \*\*kwargs)

Bases: [RangePlotMixin](#), [TimeSeriesHistogramPlot](#)

```
defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'bottom': 1e-300,
'contour.algorithm': 'mpl2014', 'cumulative': True, 'density': True, 'fftlength':
8, 'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large',
'figure.labelweight': 'normal', 'fmin': 10, 'font.sans-serif': ['Roboto'],
'grid.alpha': 0.5, 'grid.linewidth': 0.5, 'histtype': 'stepfilled',
'legend.edgecolor': 'inherit', 'legend.fancybox': False, 'legend.fontsize': 10.0,
'legend.handlelength': 1.0, 'legend.numpoints': 2, 'log': False, 'mathtext.bf':
'Roboto', 'mathtext.cal': 'Calligraffiti', 'mathtext.fontset': 'custom',
'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf': 'Roboto',
'mathtext.tt': 'Roboto Slab', 'overlap': 4, 'range': (1, 'max'), 'rwidth': 1,
'savefig.transparent': True, 'snr': 8.0, 'stride': 60.0, 'svg.fonttype': 'none',
'text.parse_math': True, 'xlabel': 'Angle-averaged range [Mpc]',
'xtick.labelsize': 14.0, 'ylabel': 'Cumulative time duration', 'ytick.labelsize':
14.0}
```

dict of default plotting kwargs

**type** = 'range-cumulative-histogram'

name for DataPlot subclass

```

class gwsumm.plot.range.RangeCumulativeSpectrumDataPlot(*args, **kwargs)
    Bases: RangePlotMixin, SpectrumDataPlot

    data = 'spectrum'

    defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
                'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
                4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
                'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'contour.algorithm': 'mpl2014',
                'fftlength': 8, 'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large',
                'figure.labelweight': 'normal', 'fmin': 10, 'font.sans-serif': ['Roboto'],
                'format': None, 'grid.alpha': 0.5, 'grid.linewidth': 0.5, 'legend.edgecolor':
                'inherit', 'legend.fancybox': False, 'legend.fontsize': 10.0,
                'legend.handlelength': 1.0, 'legend.numpoints': 2, 'mathtext.bf': 'Roboto',
                'mathtext.cal': 'Calligraffiti', 'mathtext.fontset': 'custom', 'mathtext.it':
                'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf': 'Roboto', 'mathtext.tt':
                'Roboto Slab', 'no-percentiles': False, 'overlap': 4, 'reference-linestyle':
                '--', 'savefig.transparent': True, 'snr': 8.0, 'stride': 60.0, 'svg.fonttype':
                'none', 'text.parse_math': True, 'xlabel': 'Frequency [Hz]', 'xscale': 'log',
                'xtick.labelsize': 14.0, 'ylabel': 'Cumulative fraction of range [%]', 'ylim':
                [0, 100], 'yscale': 'linear', 'ytick.labelsize': 14.0, 'ytickmarks': [0, 20, 40,
                60, 80, 100], 'yticks': [0, 20, 40, 60, 80, 100], 'zorder': 1}

        dict of default plotting kwargs

    type = 'cumulative-range-spectrum'
        name for DataPlot subclass

class gwsumm.plot.range.RangeDataHistogramPlot(*args, **kwargs)
    Bases: RangePlotMixin, TimeSeriesHistogramPlot

    defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
                'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
                4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
                'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'bottom': 1e-300,
                'contour.algorithm': 'mpl2014', 'fftlength': 8, 'figure.figsize': [12.0, 6.0],
                'figure.labelsize': 'large', 'figure.labelweight': 'normal', 'fmin': 10,
                'font.sans-serif': ['Roboto'], 'grid.alpha': 0.5, 'grid.linewidth': 0.5,
                'histtype': 'stepfilled', 'legend.edgecolor': 'inherit', 'legend.fancybox':
                False, 'legend.fontsize': 10.0, 'legend.handlelength': 1.0, 'legend.numpoints':
                2, 'log': True, 'mathtext.bf': 'Roboto', 'mathtext.cal': 'Calligraffiti',
                'mathtext.fontset': 'custom', 'mathtext.it': 'Roboto:italic', 'mathtext.rm':
                'Roboto', 'mathtext.sf': 'Roboto', 'mathtext.tt': 'Roboto Slab', 'overlap': 4,
                'rwidth': 1, 'savefig.transparent': True, 'snr': 8.0, 'stride': 60.0,
                'svg.fonttype': 'none', 'text.parse_math': True, 'xlabel': 'Sensitive distance
                [Mpc]', 'xtick.labelsize': 14.0, 'ylabel': 'Rate [Hz]', 'ytick.labelsize': 14.0}

        dict of default plotting kwargs

    type = 'range-histogram'
        name for DataPlot subclass

class gwsumm.plot.range.RangeDataPlot(*args, **kwargs)
    Bases: RangePlotMixin, TimeSeriesDataPlot

```

```
defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'contour.algorithm': 'mpl2014',
'fftlength': 8, 'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large',
'figure.labelweight': 'normal', 'fmin': 10, 'font.sans-serif': ['Roboto'],
'grid.alpha': 0.5, 'grid.linewidth': 0.5, 'legend.edgecolor': 'inherit',
'legend.fancybox': False, 'legend.fontsize': 10.0, 'legend.handlelength': 1.0,
'legend.numpoints': 2, 'mathtext.bf': 'Roboto', 'mathtext.cal': 'Calligraffiti',
'mathtext.fontset': 'custom', 'mathtext.it': 'Roboto:italic', 'mathtext.rm':
'Roboto', 'mathtext.sf': 'Roboto', 'mathtext.tt': 'Roboto Slab', 'overlap': 4,
'savefig.transparent': True, 'snr': 8.0, 'stride': 60.0, 'svg.fonttype': 'none',
'text.parse_math': True, 'xscale': 'auto-gps', 'xtick.labelsize': 14.0, 'ylabel':
'Sensitive distance [Mpc]', 'yscale': 'linear', 'ytick.labelsize': 14.0}

    dict of default plotting kwargs

type = 'range'
    name for DataPlot subclass

class gwsumm.plot.range.RangePlotMixin(*args, **kwargs)
    Bases: object

    data = 'spectrogram'

    defaults = {'fftlength': 8, 'fmin': 10, 'overlap': 4, 'snr': 8.0, 'stride':
60.0}

    draw()
        Read in all necessary data and generate a figure

class gwsumm.plot.range.RangeSpectrogramDataPlot(*args, **kwargs)
    Bases: RangePlotMixin, SpectrogramDataPlot

    defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'cmap': 'inferno',
'contour.algorithm': 'mpl2014', 'fftlength': 8, 'figure.figsize': [12.0, 6.0],
'figure.labelsize': 'large', 'figure.labelweight': 'normal', 'fmin': 10,
'font.sans-serif': ['Roboto'], 'format': None, 'grid.alpha': 0.5,
'grid.linewidth': 0.5, 'legend.edgecolor': 'inherit', 'legend.fancybox': False,
'legend.fontsize': 10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2,
'mathtext.bf': 'Roboto', 'mathtext.cal': 'Calligraffiti', 'mathtext.fontset':
'custom', 'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf':
'Roboto', 'mathtext.tt': 'Roboto Slab', 'norm': 'linear', 'overlap': 4,
'rasterized': True, 'ratio': None, 'savefig.transparent': True, 'snr': 8.0,
'stride': 60.0, 'svg.fonttype': 'none', 'text.parse_math': True, 'xscale':
'auto-gps', 'xtick.labelsize': 14.0, 'ylabel': 'Frequency [Hz]', 'yscale': 'log',
'ytick.labelsize': 14.0}

    dict of default plotting kwargs

type = 'range-spectrogram'
    name for DataPlot subclass

class gwsumm.plot.range.RangeSpectrumDataPlot(*args, **kwargs)
    Bases: RangePlotMixin, SpectrumDataPlot
```

```

data = 'spectrum'

defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'contour.algorithm': 'mpl2014',
'fftlength': 8, 'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large',
'figure.labelweight': 'normal', 'fmin': 10, 'font.sans-serif': ['Roboto'],
'format': None, 'grid.alpha': 0.5, 'grid.linewidth': 0.5, 'legend.edgecolor':
'inherit', 'legend.fancybox': False, 'legend.fontsize': 10.0,
'legend.handlelength': 1.0, 'legend.numpoints': 2, 'mathtext.bf': 'Roboto',
'mathtext.cal': 'Calligraffiti', 'mathtext.fontset': 'custom', 'mathtext.it':
'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf': 'Roboto', 'mathtext.tt':
'Roboto Slab', 'no-percentiles': False, 'overlap': 4, 'reference-linestyle':
'--', 'savefig.transparent': True, 'snr': 8.0, 'stride': 60.0, 'svg.fonttype':
'none', 'text.parse_math': True, 'xlabel': 'Frequency [Hz]', 'xscale': 'log',
'xtick.labelsize': 14.0, 'yscale': 'linear', 'ytick.labelsize': 14.0, 'zorder':
1}

    dict of default plotting kwargs

type = 'range-spectrum'

    name for DataPlot subclass

class gwsumm.plot.range.SimpleTimeVolumeDataPlot(sources, *args, **kwargs)
    Bases: SegmentDataPlot

    Time-series of the time-volume searched by an interferometer

    DRAW_PARAMS = ['imshow', 'origin', 'vmin', 'vmax', 'alpha', 'color', 'normed',
'log', 'weights', 'orientation', 'marker', 'density', 'bottom', 'label',
'interpolation', 'rwidth', 'aspect', 'cmap', 'range', 'histtype', 'zorder', 'norm',
's', 'linestyle', 'cumulative', 'linewidth', 'stacked', 'extent', 'bins',
'rasterized', 'logbins', 'align']

        list of parameters parsed for plot() calls

    static calculate_time_volume(segments, range)

    combined_time_volume(allsegments, allranges)

    data = 'timeseries'

    defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'contour.algorithm': 'mpl2014',
'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large', 'figure.labelweight':
'normal', 'font.sans-serif': ['Roboto'], 'grid.alpha': 0.5, 'grid.linewidth':
0.5, 'legend.edgecolor': 'inherit', 'legend.fancybox': False, 'legend.fontsize':
10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2, 'mathtext.bf': 'Roboto',
'mathtext.cal': 'Calligraffiti', 'mathtext.fontset': 'custom', 'mathtext.it':
'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf': 'Roboto', 'mathtext.tt':
'Roboto Slab', 'savefig.transparent': True, 'svg.fonttype': 'none',
'text.parse_math': True, 'xscale': 'auto-gps', 'xtick.labelsize': 14.0, 'yscale':
'linear', 'ytick.labelsize': 14.0}

        dict of default plotting kwargs

```

**draw**(*outputfile=None*)

Generate the figure for this plot

**classmethod from\_ini**(*\*args, \*\*kwargs*)

Define a new *DataPlot*.

**parse\_plot\_kwargs**(*\*\*defaults*)

Pop keyword arguments for *Axes.plot* from the *pargs* for this Plot

**property pid**

File pid for this *DataPlot*.

**type** = 'time-volume'

name for *DataPlot* subclass

## gwsumm.plot.registry module

Registry for GWSumm output plot types

All plot types should be registered for easy identification from the configuration INI files

**gwsumm.plot.registry.get\_plot**(*name*)

Query the registry for the plot class registered to the given name

**gwsumm.plot.registry.register\_plot**(*plot, name=None, force=False*)

Register a new summary *Plot* to the given name

### Parameters

**name**

[*str*] unique descriptive name for this type of plot, must not contain any spaces, e.g. 'time-series'

**plotclass**

[*type*] defining Class for this plot type

**force**

[*bool*] overwrite existing registration for this type

### Raises

**ValueError**

if name is already registered and *force* not given as *True*

## gwsumm.plot.segments module

Definitions for the standard plots

**class** gwsumm.plot.segments.**DutyDataPlot**(*flags, start, end, state=None, outdir='.', bins=None, \*\*kwargs*)

Bases: [SegmentDataPlot](#)

*DataPlot* of the duty-factor for a *SegmentList*

**calculate\_duty\_factor**(*segments, bins=None, cumulative=False, normalized=None*)

**data** = 'segments'

```
defaults = {'alpha': 0.8, 'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'contour.algorithm': 'mpl2014',
'cumulative': False, 'figure.figsize': [12.0, 6.0], 'figure.labelsize': 'large',
'figure.labelweight': 'normal', 'font.sans-serif': ['Roboto'], 'grid.alpha': 0.5,
'grid.linewidth': 0.5, 'legend.edgecolor': 'inherit', 'legend.fancybox': False,
'legend.fontsize': 10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2,
'mathtext.bf': 'Roboto', 'mathtext.cal': 'Calligraffiti', 'mathtext.fontset':
'custom', 'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf':
'Roboto', 'mathtext.tt': 'Roboto Slab', 'normalized': None, 'savefig.transparent':
True, 'sep': False, 'side_by_side': False, 'stacked': False, 'svg.fonttype':
'none', 'text.parse_math': True, 'xscale': 'auto-gps', 'xtick.labelsize': 14.0,
'ylabel': 'Duty factor [%]', 'ylim': (0, 100), 'yscale': 'linear',
'ytick.labelsize': 14.0}
```

dict of default plotting kwargs

**draw**(*outputfile=None*)

Read in all necessary data, and generate the figure.

**get\_bins**()

Work out the correct histogram binning for this *DutyDataPlot*

**parse\_plot\_kwargs**(\*args, \*\*kwargs)

Pop keyword arguments for *Axes.plot* from the *pargs* for this Plot

**property pid**

File pid for this *DataPlot*.

**type = 'duty'**

name for DataPlot subclass

**class** gwsumm.plot.segments.**NetworkDutyBarPlot**(*flags, start, end, state=None, outdir='.', \*\*kwargs*)

Bases: [SegmentBarPlot](#)

Special case of the *SegmentPiePlot* for network duty factors.

```
NETWORK_COLOR = {'G1': '#222222', 'H1': '#ee0000', 'I1': '#b0dd8b', 'K1':
'#ffb200', 'L1': '#4ba6ff', 'V1': '#9b59b6'}
```

```
NETWORK_NAME = {0: 'no', 1: 'single', 2: 'double', 3: 'triple', 4: 'quadruple',
5: 'quintuple', 6: 'sextuple'}
```

```
defaults = {'alpha': 0.6, 'scale': 'percent', 'title': 'Network duty factor',
'ylabel': 'Duty factor [%]'}
```

dict of default plotting kwargs

**draw**()

Process all data and generate the output file for this *SummaryPlot*.

This function should be provided by all sub-classes, and should take no arguments.

**type = 'network-duty-segment-bar'**

name for DataPlot subclass



```
class gwsumm.plot.segments.NetworkDutyPiePlot(flags, start, end, state=None, outdir='.', **kwargs)
```

Bases: [SegmentPiePlot](#)

Special case of the *SegmentPiePlot* for network duty factors

```
NETWORK_COLOR = {'G1': '#222222', 'H1': '#ee0000', 'I1': '#b0dd8b', 'K1':  
                  '#ffb200', 'L1': '#4ba6ff', 'V1': '#9b59b6', 'double': (0.0, 0.4, 1.0), 'no':  
                  'black', 'quadruple': (1.0, 0.4, 0.0), 'single': (1.0, 0.7, 0.0), 'triple':  
                  'pink'}
```

```
NETWORK_NAME = {0: 'no', 1: 'single', 2: 'double', 3: 'triple', 4: 'quadruple',  
                 5: 'quintuple', 6: 'sextuple'}
```

```
defaults = {'legend-bbox_to_anchor': (0.8, 0.5), 'legend-fontsize': 24,  
            'legend-frameon': False, 'legend-loc': 'center left', 'wedge-edgecolor': 'white',  
            'wedge-width': 0.55}
```

dict of default plotting kwargs

```
draw()
```

Process all data and generate the output file for this *SummaryPlot*.

This function should be provided by all sub-classes, and should take no arguments.

```
type = 'network-duty-pie'
```

name for DataPlot subclass

```
class gwsumm.plot.segments.ODCDataPlot(*args, **kwargs)
```

Bases: [SegmentLabelSvgMixin](#), [StateVectorDataPlot](#)

Custom *StateVectorDataPlot* for ODCs with bitmasks

```
data = 'odc'
```

```
defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],  
            'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,  
            4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,  
            'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'color': None,  
            'contour.algorithm': 'mpl2014', 'figure.figsize': [12.0, 6.0], 'figure.labelsize':  
            'large', 'figure.labelweight': 'normal', 'font.sans-serif': ['Roboto'],  
            'grid.alpha': 0.5, 'grid.linewidth': 0.5, 'in_mask_color': (0.0, 0.4, 1.0),  
            'insetlabels': 'inset', 'legend-bbox_to_anchor': (1.01, 1),  
            'legend-borderaxespad': 0.0, 'legend-fontsize': 10, 'legend-frameon': False,  
            'legend-handletextpad': 0.5, 'legend-loc': 'upper left', 'legend.edgecolor':  
            'inherit', 'legend.fancybox': False, 'legend.fontsize': 10.0,  
            'legend.handlelength': 1.0, 'legend.numpoints': 2, 'mask': None,  
            'masked_off_color': 'red', 'mathtext.bf': 'Roboto', 'mathtext.cal':  
            'Calligraffiti', 'mathtext.fontset': 'custom', 'mathtext.it': 'Roboto:italic',  
            'mathtext.rm': 'Roboto', 'mathtext.sf': 'Roboto', 'mathtext.tt': 'Roboto Slab',  
            'no_summary_bit': False, 'on-is-bad': False, 'savefig.transparent': True,  
            'svg.fonttype': 'none', 'text.parse_math': True, 'unmasked_off_color': (1.0, 0.7,  
            0.0), 'xscale': 'auto-gps', 'xtick.labelsize': 14.0, 'yscale': 'linear',  
            'ytick.labelsize': 14.0}
```

dict of default plotting kwargs

```
draw()
```

Read in all necessary data, and generate the figure.



```
get_bitmask_channels()

property pid

type = 'odc'
    name for DataPlot subclass

class gwsumm.plot.segments.SegmentBarPlot(flags, start, end, state=None, outdir='.', **kwargs)
    Bases: BarPlot, SegmentDataPlot

    SCALE_UNIT = {'percent': '%', 1: 'seconds', 3600: 'hours', 60: 'minutes', None:
    'seconds'}

    defaults = {'alpha': 0.6, 'color': '#33cc33', 'edgecolor': 'green', 'scale':
    'percent'}
        dict of default plotting kwargs

    draw(outputfile=None)
        Process all data and generate the output file for this SummaryPlot.

        This function should be provided by all sub-classes, and should take no arguments.

    type = 'segment-bar'
        name for DataPlot subclass

class gwsumm.plot.segments.SegmentDataPlot(flags, start, end, state=None, outdir='.', **kwargs)
    Bases: SegmentLabelSvgMixin, TimeSeriesDataPlot

    Segment plot of one or more DataQualityFlags <DataQualityFlag>.

    DRAW_PARAMS = ['imshow', 'origin', 'vmin', 'vmax', 'alpha', 'color', 'normed',
    'log', 'weights', 'orientation', 'marker', 'density', 'bottom', 'label',
    'interpolation', 'rwidth', 'aspect', 'cmap', 'range', 'histtype', 'zorder', 'norm',
    's', 'linestyle', 'cumulative', 'linewidth', 'stacked', 'extent', 'bins',
    'rasterized', 'logbins', 'align', 'known', 'height', 'y', 'facecolor', 'edgecolor']
        list of parameters parsed for plot() calls

    add_flag(f)

    add_legend(ax, colors, **kwargs)

    property allflags

    data = 'segments'
```

```
defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'color': None,
'contour.algorithm': 'mpl2014', 'figure.figsize': [12.0, 6.0], 'figure.labelsize':
'large', 'figure.labelweight': 'normal', 'font.sans-serif': ['Roboto'],
'grid.alpha': 0.5, 'grid.linewidth': 0.5, 'insetlabels': 'inset',
'legend-bbox_to_anchor': (1.0, 1.0), 'legend-borderaxespad': 0, 'legend-fontsize':
12, 'legend-frameon': False, 'legend-handletextpad': 0.5, 'legend-loc': 'upper
left', 'legend.edgecolor': 'inherit', 'legend.fancybox': False, 'legend.fontsize':
10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2, 'mask': None,
'mathtext.bf': 'Roboto', 'mathtext.cal': 'Calligraffiti', 'mathtext.fontset':
'custom', 'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf':
'Roboto', 'mathtext.tt': 'Roboto Slab', 'on-is-bad': False, 'savefig.transparent':
True, 'svg.fonttype': 'none', 'text.parse_math': True, 'xscale': 'auto-gps',
'xtick.labelsize': 14.0, 'yscale': 'linear', 'ytick.labelsize': 14.0}
```

dict of default plotting kwargs

### **draw()**

Read in all necessary data, and generate the figure.

### **property flags**

**classmethod from\_ini**(*config, section, start, end, flags=None, state='all', \*\*kwargs*)

Define a new *DataPlot*.

**get\_channel\_groups**(*\*args, \*\*kwargs*)

Find and group (mean, min, max) sets of channels for plotting.

### **Returns**

#### **groups**

[*list of tuple*] list of (channelname, channellist) tuples giving core channel name and an ordered list of channels. Ordering in preference of 'rms', 'mean', 'min', 'max'.

### **Notes**

This method used to return an *OrderedDict*, but was changed to return a *list of tuple* to enable plotting a channel multiple times on a plot, for whatever reason.

**get\_segment\_color()**

Parse the configured pargs and determine the colors for active and valid segments.

### **property ifos**

Interferometer set for this *SegmentDataPlot*

**init\_plot**(*projection='segments', \*\*kwargs*)

Initialise the Figure and Axes objects for this *TimeSeriesDataPlot*.

### **property padding**

**parse\_plot\_kwargs**(*\*args, \*\*kwargs*)

Pop keyword arguments for *Axes.plot* from the *pargs* for this Plot

### **property pid**

File pid for this *DataPlot*.

```

    type = 'segments'
        name for DataPlot subclass

class gwsumm.plot.segments.SegmentHistogramPlot(flags, start, end, state=None, outdir='.', **kwargs)
    Bases: TimeSeriesHistogramPlot, SegmentDataPlot

    Histogram of segment duration

    data = 'segments'

    defaults = {'bottom': 0, 'histtype': 'stepfilled', 'log': False, 'rwidth': 1,
        'ylabel': 'Number of segments'}
        dict of default plotting kwargs

    draw(outputfile=None)
        Get data and generate the figure.

    parse_plot_kwargs(**defaults)
        Pop keyword arguments for Axes.plot from the pargs for this Plot

    type = 'segment-histogram'
        name for DataPlot subclass

class gwsumm.plot.segments.SegmentPiePlot(flags, start, end, state=None, outdir='.', **kwargs)
    Bases: PiePlot, SegmentDataPlot

    defaults = {'legend-bbox_to_anchor': (0.8, 0.5), 'legend-fontsize': 14,
        'legend-frameon': False, 'legend-loc': 'center left', 'wedge-edgecolor': 'white',
        'wedge-width': 0.55}
        dict of default plotting kwargs

    draw(outputfile=None)
        Process all data and generate the output file for this SummaryPlot.

        This function should be provided by all sub-classes, and should take no arguments.

    parse_plot_kwargs(**defaults)
        Pop keyword arguments for Axes.plot from the pargs for this Plot

    parse_wedge_kwargs(defaults={})

    type = 'segment-pie'
        name for DataPlot subclass

class gwsumm.plot.segments.StateVectorDataPlot(*args, **kwargs)
    Bases: TimeSeriesDataPlot

    DataPlot of some StateVector data.

    While technically a sub-class of the TimeSeriesDataPlot, for data access and processing reasons, the output
    shadows that of the SegmentDataPlot more closely.

    DRAW_PARAMS = ['imshow', 'origin', 'vmin', 'vmax', 'alpha', 'color', 'normed',
        'log', 'weights', 'orientation', 'marker', 'density', 'bottom', 'label',
        'interpolation', 'rwidth', 'aspect', 'cmap', 'range', 'histtype', 'zorder', 'norm',
        's', 'linestyle', 'cumulative', 'linewidth', 'stacked', 'extent', 'bins',
        'rasterized', 'logbins', 'align', 'known', 'height', 'y', 'facecolor', 'edgecolor']
        list of parameters parsed for plot() calls

```

```
data = 'statevector'

defaults = {'animation.convert_args': ['-layers', 'OptimizePlus'],
'axes.axisbelow': False, 'axes.edgecolor': 'gray', 'axes.formatter.limits': [-3,
4], 'axes.formatter.use_mathtext': True, 'axes.grid': True, 'axes.labelpad': 5.0,
'axes.labelsize': 18.0, 'axes.titlesize': 22.0, 'color': None,
'contour.algorithm': 'mpl2014', 'figure.figsize': [12.0, 6.0], 'figure.labelsize':
'large', 'figure.labelweight': 'normal', 'font.sans-serif': ['Roboto'],
'grid.alpha': 0.5, 'grid.linewidth': 0.5, 'insetlabels': 'inset',
'legend-bbox_to_anchor': (1.0, 1.0), 'legend-borderaxespad': 0, 'legend-fontsize':
12, 'legend-frameon': False, 'legend-handletextpad': 0.5, 'legend-loc': 'upper
left', 'legend.edgecolor': 'inherit', 'legend.fancybox': False, 'legend.fontsize':
10.0, 'legend.handlelength': 1.0, 'legend.numpoints': 2, 'mask': None,
'mathtext.bf': 'Roboto', 'mathtext.cal': 'Calligraffiti', 'mathtext.fontset':
'custom', 'mathtext.it': 'Roboto:italic', 'mathtext.rm': 'Roboto', 'mathtext.sf':
'Roboto', 'mathtext.tt': 'Roboto Slab', 'on-is-bad': False, 'savefig.transparent':
True, 'svg.fonttype': 'none', 'text.parse_math': True, 'xscale': 'auto-gps',
'xtick.labelsize': 14.0, 'yscale': 'linear', 'ytick.labelsize': 14.0}
```

dict of default plotting kwargs

**draw()**

Read in all necessary data, and generate the figure.

**property flag**

List of flags generated for this *StateVectorDataPlot*.

**get\_segment\_color()**

Parse the configured pargs and determine the colors for active and valid segments.

**init\_plot(\*args, \*\*kwargs)**

Initialise the Figure and Axes objects for this *TimeSeriesDataPlot*.

**parse\_plot\_kwargs(\*args, \*\*kwargs)**

Pop keyword arguments for *Axes.plot* from the *pargs* for this Plot

**property pid**

**type = 'statevector'**

name for DataPlot subclass

**gwsumm.plot.segments.common\_limits(datasets, default\_min=0, default\_max=0)**

Find the global maxima and minima of a list of datasets.

**Parameters**

**datasets**

[*iterable*] list (or any other iterable) of data arrays to analyse.

**default\_min**

[*float*, optional] fall-back minimum value if datasets are all empty.

**default\_max**

[*float*, optional] fall-back maximum value if datasets are all empty.

**Returns**

(**min**, **max**)

[*float*] 2-tuple of common minimum and maximum over all datasets.

**gwsumm.plot.segments.tint\_hex(\*args, \*\*kwargs)**

## gwsumm.plot.sei module

*SummaryTab* for seismic watchdog monitoring

```
class gwsumm.plot.sei.SeiWatchDogPlot(gpstime, chamber, sensor, config, outfile, ifo=None, duration=30,  
                                     nds=False, datacache=None)
```

Bases: *DataPlot*

Plot a specific SEI WatchDog trip

**data** = 'watchdog'

**draw()**

Process all data and generate the output file for this *SummaryPlot*.

This function should be provided by all sub-classes, and should take no arguments.

**property** **outputfile**

Output file for this *DataPlot*.

**type** = 'watchdog'

name for *DataPlot* subclass

## gwsumm.plot.utils module

Utilities for GWSumm plotting

```
gwsumm.plot.utils.get_column_label(column)
```

```
gwsumm.plot.utils.get_column_string(column)
```

Format the string *columnName* (e.g. xml table column) into latex format for an axis label.

**Parameters**

**column**

[*str*] string to format

## Examples

```
>>> get_column_string('snr')  
'SNR'  
>>> get_column_string('bank_chisq_dof')  
r'Bank  $\chi^2$  DOF'
```

```
gwsumm.plot.utils.hash(string, num=6)
```

Generate an N-character hash string based using *string* to initialise

**Parameters**

**string**

[*str*] the initialisation string

**num**

[*int*, optional] the length of the hash to produce

**Returns**

**hash**

[*str*] the new hash

## Examples

```
>>> from gwsumm.plot.utils import hash
>>> print(hash("I love gravitational waves"))
80c897
```

## Module contents

A *Plot* is a representation of an image to be included in the HTML output a tab.

For simple purposes, a *Plot* is just a reference to an existing image file that can be imported into an HTML page via the `<img>` tag.

For more complicated purposes, a number of data plot classes are provided to allow users to generate images on-the-fly. The available classes are:

<code>TimeSeriesDataPlot(*args, **kwargs)</code>	DataPlot of some <i>TimeSeries</i> data.
<code>SpectrogramDataPlot(*args, **kwargs)</code>	DataPlot a Spectrogram
<code>SegmentDataPlot(flags, start, end[, state, ...])</code>	Segment plot of one or more <i>DataQualityFlags</i> <code>&lt;DataQualityFlag&gt;</code> .
<code>StateVectorDataPlot(*args, **kwargs)</code>	DataPlot of some <i>StateVector</i> data.
<code>SpectrumDataPlot(channels, start, end[, ...])</code>	Spectrum plot for a <i>SummaryTab</i>
<code>TimeSeriesHistogramPlot(channels, start, end)</code>	HistogramPlot from a Series
<code>TriggerTimeSeriesDataPlot(*args, **kwargs)</code>	Custom time-series plot to handle discontinuous <i>Time-Series</i> .
<code>TriggerHistogramPlot(*args, **kwargs)</code>	HistogramPlot from a LIGO_LW Table
<code>TriggerRateDataPlot(*args, **kwargs)</code>	TimeSeriesDataPlot of trigger rate.

## gwsumm.state package

### Submodules

#### gwsumm.state.all module

Definition of the ‘All’ state.

This is a special *SummaryState* that has valid and active segments spanning the full analysis interval.

`gwsumm.state.all.generate_all_state(start, end, register=True, **kwargs)`

Build a new *SummaryState* for the given [start, end) interval.

#### Parameters

**start**

[~*gwpy.time.LIGOTimeGPS*, float] the GPS start time of the current analysis

**end**

[~*gwpy.time.LIGOTimeGPS*, float] the GPS end time of the current analysis

**register**

[*bool*, optional] should the new *SummaryState* be registered, default *True*

**\*\*kwargs**

other keyword arguments passed to the *SummaryState* constructor

**Returns**
**allstate**

[*SummaryState*] the newly created 'All' *SummaryState*

## gwsumm.state.core module

Definition of the *SummaryState* class.

```
class gwsumm.state.core.SummaryState(name, known=[], active=[], description=None, definition=None,
                                     hours=None, key=None, filename=None, url=None)
```

Bases: [DataQualityFlag](#)

An operating state over which to process a *~gwsumm.tabs.DataTab*.

**Parameters**
**name**

[*str*] name for this state

**known**

[*~gwp.py.segments.SegmentList*, optional] list of known segments

**active**

[*~gwp.py.segments.SegmentList*, optional] list of active segments

**description**

[*str*, optional] text describing what this state means

**definition**

[*str*, optional] logical combination of flags that define known and active segments for this state (see [:attr:~gwsumm.state.core.SummaryState.definition](#) for details)

**key**

[*str*, optional] registry key for this state, defaults to [:attr:~gwsumm.state.core.SummaryState.name](#)

```
MATH_DEFINITION = re.compile('<|<=|=|>=|>|==|!=|')
```

**copy()**

Build an exact copy of this flag.

**Returns**
**flag2**

[*DataQualityFlag*] a copy of the original flag, but with a fresh memory address.

**property definition**

The combination of data-quality flags that define this *SummaryState*

For example:

```
>>> state = SummaryState(definition='L1:DMT-SCIENCE:1')
```

would define a *SummaryState* based on the validity and activity of the single flag 'L1:DMT-SCIENCE:1' (the science-mode flag for the LIGO Livingston Observatory interferometer). Similarly:

```
>>> state = SummaryState(
    definition='L1:DMT-SCIENCE:1&!L1:DMT-LIGHTDIP_10_PERCENT:1')
```

would define a *SummaryState* as active when the 'L1:DMT-SCIENCE:1' flag was active and the 'L1:DMT-LIGHTDIP\_10\_PERCENT:1' flag was not active.

The following logical identifiers are acceptable:

&	Union (i.e. flag1 <b>and</b> flag2 must be active)
	Intersection (i.e. flag1 <b>or</b> flag2 must be active)
&!	One-sided difference (i.e. flag1 is active and flag2 <b>is not</b> active)
!=	Two-sided difference (i.e. flag1 is active and flag2 is not <b>OR</b> flag2 is active and flag2 is not)

### Type

*str*

### property end

GPS end time of this state's validity

**fetch**(*config*=<GWSummConfigParser()>, *segmentcache*=None, *segdb\_error*='raise', *datacache*=None, *datafind\_error*='raise', *nproc*=1, *nds*=None, *\*\*kwargs*)

Finalise this state by fetching its defining segments, either from global memory, or from the segment database

**classmethod from\_ini**(*config*, *section*)

Create a new *SummaryState* from a section in a *ConfigParser*.

### Parameters

#### **config**

[**:class:~gwsumm.config.GWConfigParser**] customised configuration parser containing given section

#### **section**

[*str*] name of section to parse

### Returns

#### *SummaryState*

a new state, with attributes set from the options in the configuration

### property key

The registry key for this *SummaryState*.

### Type

*str*

### property name

Name of this state

### property start

GPS start time of this state's validity.

### property tag

File tag for images generated using this state



## gwsumm.state.registry module

Registry for *states* <*SummaryState*>.

`gwsumm.state.registry.get_state(key)`

Query the registry for the *SummaryState* registered to the given key

### Parameters

#### key

[*str*] registered key of desired *SummaryState*. This may not match the `~SummaryState.name` attribute if the state was registered with a different key.

### Returns

#### state

[*SummaryState*] the *SummaryState* registered with the given key

### Raises

#### ValueError:

if the *key* doesn't map to a registered *SummaryState*

`gwsumm.state.registry.get_states(keys={})`

Query the registry for a list of states (defaults to all)

### Parameters

#### keys

[*set of str*] the set of state keys to query in the registry

### Returns

#### states

[*dict*] a *dict* of (*key*, *SummaryState*) pairs

### Raises

#### ValueError:

if any of the *keys* doesn't map to a registered *SummaryState*

`gwsumm.state.registry.register_state(state, key=None, force=False)`

Register a new *SummaryState* to the given *key*

### Parameters

#### state

[*SummaryState*] defining Class for this state type.

#### key

[*str*, optional] unique descriptive name for the *SummaryState* to be registered. If *key=None*, the `:attr:`~SummaryState.key`` attribute of the given state will be used.

#### force

[*bool*] overwrite existing registration for this key

### Raises

#### ValueError

if *key* is already registered and *force* not given as *True*

## Module contents

A *SummaryState* defines a sub-set of time over which a *~gwsomm.tabs.Tab* should be processed. Each *SummaryState* is normally tied to one or more data-quality flags marking times during which each of the LIGO instruments was operating in a certain configuration, or was subject to a known noise interference.

## The state registry

GWSumm defines a state ‘registry’, simply a record of all *SummaryState* objects that have been defined (and registered) so far in a given program. The registry just makes remembering states in complicated programs a little easier.

Any *SummaryState* can be registered with an arbitrary name as follows:

```
>>> from gwsomm.state.registry import register_state
>>> register_state(mystate, 'my state')
```

and can be recovered later:

```
>>> from gwsomm.state.registry import get_state
>>> mystate = get_state('my state')
```

## API reference

<code>SummaryState(name[, known, active, ...])</code>	An operating state over which to process a <i>~gwsomm.tabs.DataTab</i> .
<code>get_state(key)</code>	Query the registry for the <i>SummaryState</i> registered to the given key
<code>get_states([keys])</code>	Query the registry for a list of states (defaults to all)
<code>register_state(state[, key, force])</code>	Register a new <i>SummaryState</i> to the given key

```
class gwsomm.state.SummaryState(name, known=[], active=[], description=None, definition=None,
                                hours=None, key=None, filename=None, url=None)
```

Bases: [DataQualityFlag](#)

An operating state over which to process a *~gwsomm.tabs.DataTab*.

### Parameters

#### name

[*str*] name for this state

#### known

[*~gwpy.segments.SegmentList*, optional] list of known segments

#### active

[*~gwpy.segments.SegmentList*, optional] list of active segments

#### description

[*str*, optional] text describing what this state means

#### definition

[*str*, optional] logical combination of flags that define known and active segments for this state (see [:attr:~gwsomm.state.SummaryState.definition](#) for details)

**key**

[*str*, optional] registry key for this state, defaults to `:attr:`~SummaryState.name``

**MATH\_DEFINITION** = `re.compile('<|<=|=|>|=|==|!=|')`

**copy()**

Build an exact copy of this flag.

**Returns**

**flag2**

[*DataQualityFlag*] a copy of the original flag, but with a fresh memory address.

**property definition**

The combination of data-quality flags that define this *SummaryState*

For example:

```
>>> state = SummaryState(definition='L1:DMT-SCIENCE:1')
```

would define a *SummaryState* based on the validity and activity of the single flag 'L1:DMT-SCIENCE:1' (the science-mode flag for the LIGO Livingston Observatory interferometer). Similarly:

```
>>> state = SummaryState(
    definition='L1:DMT-SCIENCE:1&!L1:DMT-LIGHTDIP_10_PERCENT:1')
```

would define a *SummaryState* as active when the 'L1:DMT-SCIENCE:1' flag was active and the 'L1:DMT-LIGHTDIP\_10\_PERCENT:1' flag was not active.

The following logical identifiers are acceptable:

&	Union (i.e. flag1 <b>and</b> flag2 must be active)
	Intersection (i.e. flag1 <b>or</b> flag2 must be active)
&!	One-sided difference (i.e. flag1 is active and flag2 <b>is not</b> active)
!=	Two-sided difference (i.e. flag1 is active and flag2 is not <b>OR</b> flag2 is active and flag1 is not)

**Type**

*str*

**property end**

GPS end time of this state's validity

**fetch**(*config*=<*GWSummConfigParser*()>, *segmentcache*=None, *segdb\_error*='raise', *datacache*=None, *datafind\_error*='raise', *nproc*=1, *nds*=None, *\*\*kwargs*)

Finalise this state by fetching its defining segments, either from global memory, or from the segment database

**classmethod from\_ini**(*config*, *section*)

Create a new *SummaryState* from a section in a *ConfigParser*.

**Parameters**

**config**

[`:class:`~gwsumm.config.GWConfigParser``] customised configuration parser containing given section

**section**

[*str*] name of section to parse

## Returns

### *SummaryState*

a new state, with attributes set from the options in the configuration

### property key

The registry key for this *SummaryState*.

### Type

*str*

### property name

Name of this state

### property start

GPS start time of this state's validity.

### property tag

File tag for images generated using this state

`gwsumm.state.generate_all_state(start, end, register=True, **kwargs)`

Build a new *SummaryState* for the given [start, end) interval.

## Parameters

### start

[~*gwp.py.time.LIGOTimeGPS*, float] the GPS start time of the current analysis

### end

[~*gwp.py.time.LIGOTimeGPS*, float] the GPS end time of the current analysis

### register

[*bool*, optional] should the new *SummaryState* be registered, default *True*

### \*\*kwargs

other keyword arguments passed to the *SummaryState* constructor

## Returns

### allstate

[*SummaryState*] the newly created 'All' *SummaryState*

`gwsumm.state.get_state(key)`

Query the registry for the *SummaryState* registered to the given key

## Parameters

### key

[*str*] registered key of desired *SummaryState*. This may not match the *~SummaryState.name* attribute` if the state was registered with a different key.

## Returns

### state

[*SummaryState*] the *SummaryState* registered with the given key

## Raises

### ValueError:

if the key doesn't map to a registered *SummaryState*

`gwsumm.state.get_states(keys={})`

Query the registry for a list of states (defaults to all)

#### Parameters

##### **keys**

[*set of str*] the set of state keys to query in the registry

#### Returns

##### **states**

[*dict*] a *dict* of (*key*, *SummaryState*) pairs

#### Raises

##### **ValueError:**

if any of the keys doesn't map to a registered *SummaryState*

`gwsumm.state.register_state(state, key=None, force=False)`

Register a new *SummaryState* to the given *key*

#### Parameters

##### **state**

[*SummaryState*] defining Class for this state type.

##### **key**

[*str*, optional] unique descriptive name for the *SummaryState* to be registered. If *key=None*, the **:attr:`~SummaryState.key`** attribute of the given state will be used.

##### **force**

[*bool*] overwrite existing registration for this key

#### Raises

##### **ValueError**

if key is already registered and *force* not given as *True*

## gwsumm.tabs package

### Submodules

#### gwsumm.tabs.builtin module

This module defines a number of *Tab* subclasses.

The *builtin* classes provide interfaces for simple operations including

- *ExternalTab*: embedding an existing webpage
- *PlotTab*: scaffolding a collection of existing images
- *StateTab*: scaffolding plots split into a number of states, with a button to switch between states in the HTML

**class** `gwsumm.tabs.builtin.ExternalTab(*args, **kwargs)`

Bases: [Tab](#)

A simple tab to link HTML from an external source

#### Parameters

**name**

[*str*] name of this tab (required)

**url**

[*str*] URL of the external content to be linked into this tab.

**index**

[*str*] HTML file in which to write. By default each tab is written to an `index.html` file in its own directory. Use `:attr:~Tab.index`` to find out the default index, if not given.

**shortname**

[*str*] shorter name for this tab to use in the navigation bar. By default the regular name is used

**parent**

[`:class:~gwsumm.tabs.Tab``] parent of this tab. This is used to position this tab in the navigation bar.

**children**

[*list*] list of child `:class:~Tabs <~gwsumm.tabs.Tab>`` of this one. This is used to position this tab in the navigation bar.

**group**

[*str*] name of containing group for this tab in the navigation bar dropdown menu. This is only relevant if this tab has a parent.

**path**

[*str*] base output directory for this tab (should be the same directory for all tabs in this run).

**classmethod** `from_ini(cp, section, *args, **kwargs)`

Configure a new *ExternalTab* from a *ConfigParser* section

**Parameters****cp**

[`:class:~gwsumm.config.ConfigParser``] configuration to parse.

**section**

[*str*] name of section to read

**See also:**

`:obj:~Tab.from_ini``

for documentation of the standard configuration options

**Notes**

On top of the standard configuration options, the *ExternalTab* can be configured with the `url` option, specifying the URL of the external content to be included:

```
[tab-external]
name = External data
type = external
url = https://www.example.org/index.html
```

**html\_content**(*content*)

Build the `#main` div for this tab.

**Parameters**

#### content

[*str*, ~MarkupPy.markup.page] HTML content to be wrapped

#### Returns

##### #main

[~MarkupPy.markup.page] A new *page* with the input content wrapped as

**type = 'external'**

**property url**

**write\_html(\*\*kwargs)**

Write the HTML page for this tab.

**See also:**

**:obj:`gwsomm.tabs.Tab.write\_html`**  
for details of all valid keyword

**:obj:`arguments`**

**class gwsomm.tabs.builtin.PlotTab(\*args, \*\*kwargs)**

Bases: [Tab](#)

A simple tab to layout some figures in the #main div.

#### Parameters

##### name

[*str*] name of this tab (required)

##### plots

[*list*, optional] list of plots to display on this tab. More plots can be added at any time via **:meth:`PlotTab.add\_plot`**

##### layout

[*int*, *list*, optional] the number of plots to display in each row, or a list of numbers to define each row individually. If the number of plots defined by the layout is less than the total number of plots, the layout for the final row will be repeated as necessary.

For example `layout=[1, 2, 3]` will display a single plot on the top row, two plots on the second, and 3 plots on each row thereafter.

##### foreword

[~MarkupPy.markup.page, *str*, optional] content to include in the #main HTML before the plots

##### afterword

[~MarkupPy.markup.page, *str*, optional] content to include in the #main HTML after the plots

##### index

[*str*, optional] HTML file in which to write. By default each tab is written to an index.html file in its own directory. Use **:attr:`~Tab.index`** to find out the default index, if not given.

##### shortname

[*str*, optional] shorter name for this tab to use in the navigation bar. By default the regular name is used

**parent**

[**:class:~gwsomm.tabs.Tab**`, optional] parent of this tab. This is used to position this tab in the navigation bar.

**children**

[*list*, optional] list of child **:class:~Tabs <~gwsomm.tabs.Tab>** of this one. This is used to position this tab in the navigation bar.

**group**

[*str*, optional] name of containing group for this tab in the navigation bar dropdown menu. This is only relevant if this tab has a parent.

**path**

[*str*, optional,] base output directory for this tab (should be the same directory for all tabs in this run).

**add\_plot**(*plot*)

Add a plot to this tab.

**Parameters**
**plot**

[*str*, **:class:~gwsomm.plot.SummaryPlot**] either the URL of a plot to embed, or a formatted *SummaryPlot* object.

**property afterword**

HTML content to be included after the plots

**property foreword**

HTML content to be included before the plots

**classmethod from\_ini**(*cp*, *section*, *\*args*, *\*\*kwargs*)

Define a new tab from a **:class:~gwsomm.config.GWConfigParser**

**Parameters**
**cp**

[**:class:~ConfigParser.GWConfigParser**] customised configuration parser containing given section

**section**

[*str*] name of section to parse

**Returns**
**tab**

[*PlotTab*] a new tab defined from the configuration

**html\_content**(*content*)

Build the #main div for this tab.

**Parameters**
**content**

[*str*, *~MarkupPy.markup.page*] HTML content to be wrapped

**Returns**
**#main**

[*~MarkupPy.markup.page*] A new *page* with the input content wrapped as



**property layout**

List of how many plots to display on each row in the output.

By default this is 1 if the tab contains only 1 or 3 plots, or 2 if otherwise. The final number given in the list will be repeated as necessary.

**Type**

*list of ints <int>*

**scaffold\_plots**(*plots=None, state=None, layout=None, aclass='fancybox', \*\*fbkw*)

Build a grid of plots using bootstrap's scaffolding.

**Returns****page**

`[:class:~MarkupPy.markup.page] formatted markup with grid of plots`

**set\_layout**(*layout*)

Set the plot scaffolding layout for this tab

**Parameters**

**l**

[*int, list of int*] the desired scaffold layout, one of

- an *int*, indicating the number of plots on every row, or
- a *list*, indicating the number of plots on each row, with the final *int* repeated for any remaining rows; each entry should be an *int* or a pair of *int* indicating the number of plots on this row AND the desired the scaling of this row, see the examples...

**Examples**

To layout 2 plots on each row

```
>>> tab.set_layout(2)
```

or

```
>>> tab.set_layout([2])
```

To layout 2 plots on the first row, and 3 on all other rows

```
>>> tab.set_layout((2, 3))
```

To layout 2 plots on the first row, and 1 on the second row BUT have it the same size as plots on a 2-plot row

```
>>> tab.set_layout((2, (1, 2)))
```

**type = 'plots'**

**write\_html**(*foreword=None, afterword=None, \*\*kwargs*)

Write the HTML page for this tab.

**Parameters****foreword**

[*str, :class:~MarkupPy.markup.page*, optional] content to place above the plot grid, defaults to **:attr:~PlotTab.foreword**

**afterword**

[*str*, :class:`~MarkupPy.markup.page`, optional] content to place below the plot grid, defaults to :attr:`~PlotTab.afterword`

**\*\*kwargs**

other keyword arguments to be passed through :meth:`~Tab.write\_html`

**See also:**

:obj:`~gwsomm.tabs.Tab.write\_html`

for details of all valid unnamed keyword arguments

**class** gwsomm.tabs.builtin.StateTab(\*args, \*\*kwargs)

Bases: [PlotTab](#)

Tab with multiple content pages defined via 'states'

Each state is printed to its own HTML file which is loaded via javascript upon request into the #main div of the index for the tab.

**Parameters****name**

[*str*] name of this tab (required)

**states**

[*list*] a list of states for this tab. Each state can take any form, but must be castable to a *str* in order to be printed.

**plots**

[*list*] list of plots to display on this tab. More plots can be added at any time via :meth:`~PlotTab.add\_plot`

**layout**

[*int*, *list*] the number of plots to display in each row, or a list of numbers to define each row individually. If the number of plots defined by the layout is less than the total number of plots, the layout for the final row will be repeated as necessary.

For example layout=[1, 2, 3] will display a single plot on the top row, two plots on the second, and 3 plots on each row thereafter.

**index**

[*str*] HTML file in which to write. By default each tab is written to an index.html file in its own directory. Use :attr:`~Tab.index` to find out the default index, if not given.

**shortname**

[*str*] shorter name for this tab to use in the navigation bar. By default the regular name is used

**parent**

[*:class:`~gwsomm.tabs.Tab`*] parent of this tab. This is used to position this tab in the navigation bar.

**children**

[*list*] list of child :class:`~Tabs <~gwsomm.tabs.Tab>` of this one. This is used to position this tab in the navigation bar.

**group**

[*str*] name of containing group for this tab in the navigation bar dropdown menu. This is only relevant if this tab has a parent.

### path

[*str*] base output directory for this tab (should be the same directory for all tabs in this run).

**add\_state**(*state*, *default=False*)

Add a *SummaryState* to this tab

### Parameters

#### state

[*str*, :class:`~gwsumm.state.SummaryState`] either the name of a state, or a *SummaryState*.

#### register

[*bool*, default: *False*] automatically register all new states

**property defaultstate**

**property frames**

**classmethod from\_ini**(*cp*, *section*, \**args*, \*\**kwargs*)

Define a new tab from a :class:`~gwsumm.config.GWConfigParser`

### Parameters

#### cp

[*class*:~ConfigParser.GWConfigParser] customised configuration parser containing given section

#### section

[*str*] name of section to parse

### Returns

#### tab

[*PlotTab*] a new tab defined from the configuration

**static html\_content**(*frame*)

Build the #main div for this tab.

In this construction, the <div id="id\_"> is empty, with a javascript hook to load the given frame into the div when ready.

**html\_navbar**(*help\_=None*, \*\**kwargs*)

Build the navigation bar for this *Tab*.

The navigation bar will consist of a switch for this page linked to other interferometer servers, followed by the navbar brand, then the full dropdown-based navigation menus configured for the given tabs and their descendents.

### Parameters

#### help\_

[*str*, :class:`~MarkupPy.markup.page`] content for upper-right of navbar

#### ifo

[*str*, optional] prefix for this IFO.

#### ifomap

[*dict*, optional] *dict* of (ifo, {base url}) pairs to map to summary pages for other IFOs.

#### tabs

[*list*, optional] list of parent tabs (each with a list of children) to include in the navigation bar.

## Returns

### page

[~MarkupPy.markup.page] a markup page containing the navigation bar.

## property states

The set of `:class:`states <gwsumm.state.SummaryState>`` over whos times this tab's data will be processed.

The set of states will be linked in the given order with a switch on the far-right of the HTML navigation bar.

`type = 'state'`

```
write_html(title=None, subtitle=None, tabs=[], ifo=None, ifomap={}, help_=None, css={'font-awesome':
'https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.1/css/fontawesome.min.css',
'font-awesome-solid':
'https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.1/css/solid.min.css', 'gwbootstrap':
'https://cdn.jsdelivr.net/npm/gwbootstrap@1.3.6/lib/gwbootstrap.min.css'}, js={'bootstrap':
'https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js', 'datepicker':
'https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datepicker/1.9.0/js/bootstrap-datepicker.min.js',
'fancybox': 'https://cdn.jsdelivr.net/npm/@fancyapps/ui@5.0/dist/fancybox/fancybox.umd.js',
'gwbootstrap': 'https://cdn.jsdelivr.net/npm/gwbootstrap@1.3.6/lib/gwbootstrap-extra.min.js',
'jquery': 'https://code.jquery.com/jquery-3.7.1.min.js', 'jquery-ui':
'https://code.jquery.com/ui/1.13.2/jquery-ui.min.js', 'moment':
'https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.30.1/moment.min.js'}, about=None,
footer=None, **inargs)
```

Write the HTML page for this state Tab.

## Parameters

### maincontent

[*str*, `:class:`~MarkupPy.markup.page``] simple string content, or a structured *page* of markup to embed as the content of the #main div.

### title

[*str*, optional, default: {parent.name}] level 1 heading for this *Tab*.

### subtitle

[*str*, optional, default: {self.name}] level 2 heading for this *Tab*.

### tabs: `list`, optional

list of top-level tabs (with children) to populate navbar

### ifo

[*str*, optional] prefix for this IFO.

### ifomap

[*dict*, optional] *dict* of (ifo, {base url}) pairs to map to summary pages for other IFOs.

### help\_

[*str*, `:class:`~MarkupPy.markup.page``, optional] non-menu content for navigation bar, defaults to calendar

### css

[*list*, optional] list of resolvable URLs for CSS files. See *gwsumm.html.static.CSS* for the default list.

### js

[*list*, optional] list of resolvable URLs for javascript files. See *gwsumm.html.JS* for the default list.

**about**

[*str*, optional] href for the ‘About’ page

**footer**

[*str*, ~*MarkupPy.markup.page*] user-defined content for the footer (placed below everything else)

**\*\*inargs**

other keyword arguments to pass to the :meth:`~Tab.build\_inner\_html` method

**write\_state\_html**(*state*, *pre=None*, *post=None*, *plots=True*)

Write the frame HTML for the specific state of this tab

**Parameters**
**state**

[~*gwsumm.state.SummaryState*] *SummaryState* over which to generate inner HTML

## gwsumm.tabs.core module

This module defines the core *Tab* object.

The basic *Tab* allows for simple embedding of arbitrary text inside a standardised HTML interface. Most real-world applications will use a sub-class of *Tab* to create more complex HTML output.

The *Tab* class comes in three flavours:

- ‘static’, no specific GPS time reference
- ‘interval’, displaying data in a given GPS [start, stop) interval
- ‘event’, displaying data around a specific central GPS time

The flavour is dynamically set when each instance is created based on the *mode* keyword, or the presence of *span*, *start* and *end*, or *gpstime* keyword arguments.

**class** *gwsumm.tabs.core.BaseTab*(*name*, *index=None*, *shortname=None*, *parent=None*, *children=[]*,  
*group=None*, *notes=None*, *overlay=None*, *path='.'*, *mode=None*,  
*hidden=False*)

Bases: *object*

The core *Tab* object, defining basic functionality

**add\_child**(*tab*)

Add a child to this *SummaryTab*

**Parameters**
**tab**

[*SummaryTab*] child tab to record

**property children**

List of child tabs for this *Tab*

If this tab is given children, it cannot also have a parent, as it will define its own dropdown menu in the HTML navigation bar, linking to itself and its children.

**Type**

*list of tabs* <*Tab*>

**classmethod** `from_ini(cp, section, *args, **kwargs)`

Define a new tab from a `~gwsomm.config.GWConfigParser`

**Parameters**

**cp**

[`~gwsomm.config.GWConfigParser`] customised configuration parser containing given section

**section**

[*str*] name of section to parse

**\*args, \*\*kwargs**

other positional and keyword arguments to pass to the class constructor (`__init__`)

**Returns**

**tab**

[*Tab*] a new tab defined from the configuration

**Notes**

This method parses the following configuration options

name	Full name for this <i>Tab</i>
shortname	Short name for this tab
parent	Short name of the parent page for this <i>Tab</i>
group	Dropdown group for this <i>Tab</i> in the navigation bar
notes	Release notes for this <i>Tab</i>
overlay	Boolean switch to enable plot overlay for this <i>Tab</i>
index	The HTML path (relative to the <i>~Tab.path</i> ) for this tab

Sub-classes should parse their own configuration values and then pass these as `*args` and `**kwargs` to this method via *super*:

```
class MyTab(Tab):
    [...]
    def from_ini(cls, cp, section)
        """Define a new `MyTab`.
        """
        foo = cp.get(section, 'foo')
        bar = cp.get(section, 'bar')
        return super(MyTab, cls).from_ini(cp, section, foo, bar=bar)
```

**get\_child(name)**

Find a child tab of this *SummaryTab* by name

**Parameters**

**name**

[*str*] string identifier of child tab to use in search

**Returns**

**child**

[*SummaryTab*] the child tab found by name

**Raises****RuntimeError**

if no child tab can be found matching the given name

**property group**

Dropdown group for this *Tab* in the navigation bar

**Type**

*str*

**property href**

HTML href (relative to the *~Tab.path*) for this tab

This attribute is just a convenience to clean-up the *~Tab.index* for a given tab, by removing index.htmls. hierarchy.

**Type**

*str*

**html\_banner**(*title=None, subtitle=None*)

Build the HTML headline banner for this tab.

**Parameters****title**

[*str*] title for this page

**subtitle**

[*str*] sub-title for this page

**Returns****banner**

[*~MarkupPy.markup.page*] formatter markup page for the banner

**static html\_content**(*content*)

Build the #main div for this tab.

**Parameters****content**

[*str, ~MarkupPy.markup.page*] HTML content to be wrapped

**Returns****#main**

[*~MarkupPy.markup.page*] A new *page* with the input content wrapped as

**html\_navbar**(*help\_=None, calendar=[], tabs=[], ifo=None, ifomap={}, \*\*kwargs*)

Build the navigation bar for this tab.

**Parameters****help\_**

[*str, ~MarkupPy.markup.page*] content to place on the upper-right side of the navbar

**calendar**

[*list, optional*] datepicker calendar objects for navigation

**tabs**

[*list, optional*] list of parent tabs (each with a list of children) to include in the navigation bar.

**ifo**

[*str*, optional] prefix for this IFO.

**ifomap**

[*dict*, optional] *dict* of (ifo, {base url}) pairs to map to summary pages for other IFOs.

**\*\*kwargs**

other keyword arguments to pass to `:meth:`gwsumm.html.navbar``

**Returns**

**page**

[*~MarkupPy.markup.page*] a markup page containing the navigation bar.

**property index**

The HTML path (relative to the *~Tab.path*) for this tab

**property mode**

The date-time mode of this tab.

**Type**

*int*

**See also:**

`:obj:`gwsumm.mode``

for details on the modes

**property name**

Full name for this *Tab*

**Type**

*str*

**property notes**

Release notes for this *Tab*

**property overlay**

Boolean switch to enable plot overlay for this *Tab*

**property parent**

Short name of the parent page for this *Tab*

A given tab can either be a parent for a set of child tabs, or can have a parent, it cannot be both. In this system, the *parent* attribute defines the heading under which this tab will be linked in the HTML navigation bar.

**Type**

*str*

**set\_parent(*p*)**

Set the parent *Tab* for this tab

**Parameters**

**p**

[*Tab*] the parent tab for this one

**property shortname**

Short name for this tab

This will be displayed in the navigation bar.



## Type

*str*

### property shorttitle

Page title for this tab

### property title

Page title for this tab

**write\_html**(*maincontent*, *title=None*, *subtitle=None*, *tabs=[]*, *ifo=None*, *ifomap={}*, *help\_=None*, *base=None*, *css=None*, *js=None*, *about=None*, *footer=None*, *issues=True*, *\*\*inargs*)

Write the HTML page for this *Tab*.

## Parameters

### maincontent

[*str*, *~MarkupPy.markup.page*] simple string content, or a structured *page* of markup to embed as the content of the #main div.

### title

[*str*, optional, default: {parent.name}] level 1 heading for this *Tab*.

### subtitle

[*str*, optional, default: {self.name}] level 2 heading for this *Tab*.

### tabs: `list`, optional

list of top-level tabs (with children) to populate navbar

### ifo

[*str*, optional] prefix for this IFO.

### ifomap

[*dict*, optional] *dict* of (ifo, {base url}) pairs to map to summary pages for other IFOs.

### help\_

[*str*, *~MarkupPy.markup.page*, optional] non-menu content for navigation bar

### css

[*list*, optional] list of resolvable URLs for CSS files. See *gwsumm.html.CSS* for the default list.

### js

[*list*, optional] list of resolvable URLs for javascript files. See *gwsumm.html.JS* for the default list.

### about

[*str*, optional] href for the 'About' page

### footer

[*str*, *~MarkupPy.markup.page*] external link, if applicable (linked from an icon in the footer)

### issues

[*bool* or *str*, default: *True*] print link to github.com issue tracker for this package

### \*\*inargs

other keyword arguments to pass to the **:meth:`~Tab.build\_inner\_html`** method

**class** gwsumm.tabs.core.Tab(\*args, \*\*kwargs)

Bases: [BaseTab](#)

A Simple HTML tab.

This *class* provides a mechanism to generate a full-formatted HTML page including banner, navigation-bar, content, and a footer, without the user worrying too much about the details.

For example:

```
>>> # import Tab and make a new one with a given title and HTML file
>>> from gwsumm.tabs import Tab
>>> tab = Tab('My new tab', 'mytab.html')
>>> # write the Tab to disk with some simple content
>>> tab.write_html('This is my content', brand='Brand name')
```

### Parameters

#### **name**

[*str*] name of this tab (required)

#### **index**

[*str*] HTML file in which to write. By default each tab is written to an index.html file in its own directory. Use `~Tab.index` to find out the default index, if not given.

#### **shortname**

[*str*] shorter name for this tab to use in the navigation bar. By default the regular name is used

#### **parent**

[`~gwsumm.tabs.Tab`] parent of this tab. This is used to position this tab in the navigation bar.

#### **children**

[*list*] list of child *Tabs* `<~gwsumm.tabs.Tab>` of this one. This is used to position this tab in the navigation bar.

#### **group**

[*str*] name of containing group for this tab in the navigation bar dropdown menu. This is only relevant if this tab has a parent.

#### **path**

[*str*] base output directory for this tab (should be the same directory for all tabs in this run)

### Notes

A *Tab* cannot have both a `~Tab.parent` and `~tab.Children`. This is a limitation imposed by the twitter bootstrap navigation bar implementation, which does not allow nested dropdown menus. In order to collect child tabs in a given place, assign them all the same `~Tab.group`.

**type = 'basic'**

**class** gwsumm.tabs.core.TabList(entries=[])

Bases: `list`

Custom *list* of *Tab* objects with sorting and parsing

**classmethod** from\_ini(*config*, *tag*='tab[\_]', *match*=[], *path*='.', *plotdir*='plots')

**get\_hierarchy()**

**sort**(*key*=None, *reverse*=False)

Sort this *TabList* in place

## gwsumm.tabs.data module

This module defines tabs for generating plots from data on-the-fly.

This module also provides the *ProcessedTab* mixin, which should be used to declare that a tab has a *process()* method that should be executed as part of a workflow, see the *gw\_summary* executable as an example.

**class** `gwsumm.tabs.data.DataTab(*args, **kwargs)`

Bases: *ProcessedTab*, *StateTab*

A tab where plots and data summaries are built upon request

This is the ‘default’ tab for the command-line *gw\_summary* executable.

All *\*args* and *\*\*kwargs* are passed up-stream to the base class constructor, excepting the following:

### Parameters

#### **name**

[*str*] name of this tab (required)

#### **start**

[*LIGOTimeGPS*, *str*] start time of this *DataTab*, anything that can be parsed by *~gwpy.time.to\_gps* is fine

#### **end**

[*LIGOTimeGPS*, *str*] end time of this *DataTab*, format as for *start*

#### **states**

[*list* of *states* <*gwsumm.state.SummaryState*>] the *list* of *states* (*~gwsumm.state.SummaryState*) over which this *DataTab* should be processed. More *states* can be added later (but before running ***:meth:~DataTab.process***) via ***:meth:~DataTab.add\_state***.

#### **ismeta**

[*bool*, optional, default: *False*] indicates that this tab only contains data already by others and so doesn’t need to be processed.

#### **noplots**

[*bool*, optional, default: *False*] indicates that this tab only exists to trigger data access, and shouldn’t actually generate any figures

#### **\*\*kwargs**

other keyword arguments

See also:

***:obj:~gwsumm.tabs.StateTab***

for details on the other keyword arguments (*\*\*kwargs*) accepted by the constructor for the *DataTab*.

**finalize\_states**(*config*=<*configparser.ConfigParser* object>, *segdb\_error*=‘raise’, *\*\*kwargs*)

Fetch the segments for each state for this *SummaryTab*

**classmethod from\_ini**(*cp*, *section*, *plotdir*=‘plots’, *\*\*kwargs*)

Define a new *SummaryTab* from the given section of the *ConfigParser*.

### Parameters

#### **cp**

[***:class:~gwsumm.config.GWConfigParser***] customised configuration parser containing given section

**section**

[*str*] name of section to parse

**plotdir**

[*str*, optional, default: 'plots'] output path for plots, relative to current directory

**Returns**

**tab**

[*DataTab*] a new *DataTab* defined from the configuration

**get\_channels(\*types, \*\*kwargs)**

Return the *set* of data channels required for plots of the given *types*.

**Parameters**

**\*types**

[*list* of *str*] *list* of plot data type strings whose channel sets to return

**new**

[*bool*, default: *True*] only include plots whose 'new' attribute is *True*

**Returns**

**channels**

[*list*] an alphabetically-sorted *list* of channels

**get\_flags(\*types, \*\*kwargs)**

Return the *set* of data-quality flags required for plots of the given *types*.

**Parameters**

**\*types**

[*list* of *str*] *list* of plot type strings whose flag sets to return

**Returns**

**flags**

[*list*] an alphabetically-sorted *list* of flags

**get\_triggers(\*types, \*\*kwargs)**

Return the *set* of data-quality flags required for plots of the given *types*.

**Parameters**

**\*types**

[*list* of *str*] *list* of plot type strings whose flag sets to return

**Returns**

**flags**

[*list*] an alphabetically-sorted *list* of flags

**html\_content(frame)**

Build the #main div for this tab.

In this construction, the <div id="id\_"> is empty, with a javascript hook to load the given frame into the div when ready.

**static print\_segments(flag, table=False, caption=None)**

Print the contents of a *SegmentList* in HTML

**process**(*config*=<*configparser.ConfigParser* object>, *nproc*=1, *\*\*stateargs*)

Process data for this tab

#### Parameters

##### **config**

[*ConfigParser.ConfigParser*, optional] job configuration to pass to *DataTab.finalize\_states*

##### **\*\*stateargs**

all other keyword arguments are passed directly onto the :meth:`~DataTab.process\_state` method.

**process\_state**(*state*, *nds*=None, *nproc*=1, *config*=<*GWSummConfigParser*()>, *datacache*=None, *trigcache*=None, *segmentcache*=None, *segdb\_error*='raise', *datafind\_error*='raise')

Process data for this tab in a given state

#### Parameters

##### **state**

[*~gwsumm.state.SummaryState*] the state to process. Can give *None* to process ALLSTATE with no plots, useful to load all data for other states

##### **nds**

[*bool*, optional] *True* to use NDS to read data, otherwise read from frames. Use *None* to read from frames if possible, otherwise using NDS.

##### **nproc**

[*int*, optional] number of parallel cores to use when reading data and making plots, default: 1

##### **config**

[*ConfigParser*, optional] configuration for this analysis

##### **datacache**

[*~glue.lal.Cache*, optional] *Cache* of files from which to read time-series data

##### **trigcache**

[*~glue.lal.Cache*, optional] *Cache* of files from which to read event triggers

##### **segmentcache**

[*~glue.lal.Cache*, optional] *Cache* of files from which to read segments

##### **segdb\_error**

[*str*, optional] if 'raise': raise exceptions when the segment database reports exceptions, if 'warn'``, print warnings but continue, otherwise ``'ignore' them completely and carry on.

**type** = 'data'

**write\_html**(\*args, \*\*kwargs)

Write the HTML page for this state Tab.

#### Parameters

##### **maincontent**

[*str*, :class:`~MarkupPy.markup.page`] simple string content, or a structured *page* of markup to embed as the content of the #main div.

##### **title**

[*str*, optional, default: {parent.name}] level 1 heading for this *Tab*.

**subtitle**

[*str*, optional, default: {self.name}] level 2 heading for this *Tab*.

**tabs: `list`, optional**

list of top-level tabs (with children) to populate navbar

**ifo**

[*str*, optional] prefix for this IFO.

**ifomap**

[*dict*, optional] *dict* of (ifo, {base url}) pairs to map to summary pages for other IFOs.

**help\_**

[*str*, :class:`~MarkupPy.markup.page`, optional] non-menu content for navigation bar, defaults to calendar

**css**

[*list*, optional] list of resolvable URLs for CSS files. See *gwsumm.html.static.CSS* for the default list.

**js**

[*list*, optional] list of resolvable URLs for javascript files. See *gwsumm.html.JS* for the default list.

**about**

[*str*, optional] href for the 'About' page

**footer**

[*str*, ~*MarkupPy.markup.page*] user-defined content for the footer (placed below everything else)

**\*\*inargs**

other keyword arguments to pass to the :meth:`~Tab.build\_inner\_html` method

**write\_state\_html(*state*)**

Write the '#main' HTML content for this tab.

For now, this function just links all the plots in a 2-column format.

**write\_state\_information(*state*)**
**write\_state\_placeholder(*state*)**

Write a placeholder '#main' content for this tab

**class gwsumm.tabs.data.ProcessedTab**

Bases: `object`

Abstract base class to detect necessity to run `Tab.process()`

**process()**

This method must be overridden by all subclasses

**type = '\_processed'**

## gwsumm.tabs.etg module

Custom *SummaryTab* for the output of an ETG.

```
class gwsumm.tabs.etg.EventTriggerTab(*args, **kwargs)
```

Bases: *DataTab*

Custom *DataTab* displaying a summary of event trigger generation

### Parameters

#### name

[*str*] name of this tab (required)

#### start

[*LIGOTimeGPS*, *str*] start time of this *DataTab*, anything that can be parsed by *~gwpy.time.to\_gps* is fine

#### end

[*LIGOTimeGPS*, *str*] end time of this *DataTab*, format as for *start*

#### channel

[*str*] name of the channel of interest

#### etg

[*str*, optional] name of this event trigger generator (ETG), defaults to the name of this *EventTriggerTab*

#### states

[*list* of *states* <*gwsumm.state.SummaryState*>] the *list* of *states* (*~gwsumm.state.SummaryState*) over which this *DataTab* should be processed. More states can be added later (but before running ***:meth:`~DataTab.process`***) via ***:meth:`~DataTab.add\_state`***.

#### table

[*type*, *str*, optional] *LIGO\_LW* *~glue.ligolw.table.Table* class to use for this ETG, e.g. use *~glue.ligolw.lscables.SnglBurstTable* for Omicron, or *~glue.ligolw.lscables.SnglInspiralTable* for CBC

#### cache

[*~glue.lal.Cache*, *str*, optional] *Cache* object, or path to a LAL-format cache file on disk, from which to read the event triggers. If no cache is given, the *gwtrigfind* module will be used to automatically locate the trigger files.

#### url

[*str*, optional] URL for linking to more details results for this tab.

#### \*\*kwargs

all other keyword arguments accepted by the *DataTab*

See also:

***:obj:`~gwsumm.tabs.DataTab`***

for details on the other keyword arguments (***\*\*kwargs***) accepted by the constructor for this *EventTriggerTab*.

```
finalize_states(config=None, **kwargs)
```

Fetch the segments for each state for this *SummaryTab*

**classmethod** `from_ini(config, section, **kwargs)`

Define a new *EventTriggerTab* from a *ConfigParser*.

#### Parameters

**cp**

[**:class:~gwsomm.config.GWConfigParser**] customised configuration parser containing given section

**section**

[*str*] name of section to parse

**\*args, \*\*kwargs**

other positional and keyword arguments to pass to the class constructor (`__init__`)

**See also:**

**:obj:~DataTab.from\_ini**

the parent parsing method that handles parsing plot definitions, amongst other things

#### Notes

In addition to those attributes parsed by the parent *DataTab.from\_ini* method, this method will parse the following attributes from a *ConfigParser*:

```
.. autosummary::
```

```
~EventTriggerTab.channel ~EventTriggerTab.etg ~EventTriggerTab.url ~EventTriggerTab.cache
~EventTriggerTab.table
```

Additionally, the loudest events table is configured by giving the following options

\_\_\_\_\_ *loudest* the number of events to include in the table  
*loudest-rank* the statistic to use in sorting the table *loudest-dt* the minimum time separation for unique events *loudest-columns* the columns to print in the table *loudest-labels* the labels for each of the given columns \_\_\_\_\_

**process(\*args, \*\*kwargs)**

Process data for this tab

#### Parameters

**config**

[*ConfigParser.ConfigParser*, optional] job configuration to pass to *DataTab.finalize\_states*

**\*\*stateargs**

all other keyword arguments are passed directly onto the **:meth:~DataTab.process\_state** method.

**process\_state(state, \*args, \*\*kwargs)**

Process data for this tab in a given state

#### Parameters

**state**

[*~gwsomm.state.SummaryState*] the state to process. Can give *None* to process ALLSTATE with no plots, useful to load all data for other states



**nds**

[*bool*, optional] *True* to use NDS to read data, otherwise read from frames. Use *None* to read from frames if possible, otherwise using NDS.

**nproc**

[*int*, optional] number of parallel cores to use when reading data and making plots, default: 1

**config**

[*ConfigParser*, optional] configuration for this analysis

**datacache**

[*~glue.lal.Cache*, optional] *Cache* of files from which to read time-series data

**trigcache**

[*~glue.lal.Cache*, optional] *Cache* of files from which to read event triggers

**segmentcache**

[*~glue.lal.Cache*, optional] *Cache* of files from which to read segments

**segdb\_error**

[*str*, optional] if 'raise': raise exceptions when the segment database reports exceptions, if 'warn'``, print warnings but continue, otherwise ``'ignore' them completely and carry on.

```
type = 'triggers'
```

```
write_state_html(state, pre=None)
```

Write the '#main' HTML content for this *EventTriggerTab*.

## gwsumm.tabs.fscan module

Custom *SummaryTab* for the output of the FScan algorithm.

```
class gwsumm.tabs.fscan.FscanTab(*args, **kwargs)
```

Bases: *DataTab*

Custom tab displaying a summary of Fscan results.

```
classmethod from_ini(config, section, **kwargs)
```

Define a new *FscanTab* from a *ConfigParser*.

```
process(config=<GWSummConfigParser()>, **kwargs)
```

Process data for this tab

### Parameters

**config**

[*ConfigParser.ConfigParser*, optional] job configuration to pass to *DataTab.finalize\_states*

**\*\*stateargs**

all other keyword arguments are passed directly onto the :meth:`~DataTab.process\_state` method.

```
type = 'fscan'
```

```
write_state_html(state)
```

Write the '#main' HTML content for this *FscanTab*.

## gwsumm.tabs.gracedb module

Custom *SummaryTab* to display events queried from the Gravitational-wave Candidate Event Database (GraceDb)

**class** gwsumm.tabs.gracedb.GraceDbTab(\*args, \*\*kwargs)

Bases: *DataTab*

Custom tab displaying a summary of GraceDb results.

**classmethod** from\_ini(config, section, \*\*kwargs)

Define a new *GraceDbTab* from a *ConfigParser*.

**process**(config=<GWSummConfigParser()>, \*\*kwargs)

Process data for this tab

### Parameters

**config**

[*ConfigParser.ConfigParser*, optional] job configuration to pass to *DataTab.finalize\_states*

**\*\*kwargs**

all other keyword arguments are passed directly onto the :meth:`~DataTab.process\_state` method.

**process\_state**(state, \*\*kwargs)

Process data for this tab in a given state

### Parameters

**state**

[*~gwsumm.state.SummaryState*] the state to process. Can give *None* to process ALLSTATE with no plots, useful to load all data for other states

**nds**

[*bool*, optional] *True* to use NDS to read data, otherwise read from frames. Use *None* to read from frames if possible, otherwise using NDS.

**nproc**

[*int*, optional] number of parallel cores to use when reading data and making plots, default: 1

**config**

[*ConfigParser*, optional] configuration for this analysis

**datacache**

[*~glue.lal.Cache*, optional] *Cache* of files from which to read time-series data

**trigcache**

[*~glue.lal.Cache*, optional] *Cache* of files from which to read event triggers

**segmentcache**

[*~glue.lal.Cache*, optional] *Cache* of files from which to read segments

**segdb\_error**

[*str*, optional] if 'raise': raise exceptions when the segment database reports exceptions, if 'warn'``, print warnings but continue, otherwise ``'ignore' them completely and carry on.

**type** = 'gracedb'

**write\_state\_html**(*state*)

Write the '#main' HTML content for this *GraceDbTab*.

## gwsumm.tabs.guardian module

Definition of the *GuardianTab*

**class** gwsumm.tabs.guardian.**GuardianTab**(\*args, \*\*kwargs)

Bases: *DataTab*

Summarises the data recorded by an Advanced LIGO Guardian node.

Each guardian node controls and monitors state transitions for a specific subsystem of the Advanced LIGO interferometer. The *GuardianTab* summarises those data with a state segments plot, a transitions summary table, and a detailed list of transitions and segments for each listed state.

**classmethod** **from\_ini**(*config, section, plotdir='plots', \*\*kwargs*)

Define a new *GuardianTab*.

**process**(*nds=None, nproc=1, config=<GWSummConfigParser()>, datacache=None, segmentcache=[], datafind\_error='raise', \*\*kwargs*)

Process data for the given state.

**type** = 'guardian'

**write\_state\_html**(*state*)

Write the HTML for the given state of this *GuardianTab*

## gwsumm.tabs.management module

Definition of the *AccountingTab*

**class** gwsumm.tabs.management.**AccountingTab**(\*args, \*\*kwargs)

Bases: *DataTab*

Summarise the data recorded by the operating mode channels

**classmethod** **from\_ini**(*config, section, plotdir='plots', \*\*kwargs*)

Define a new *SummaryTab* from the given section of the *ConfigParser*.

### Parameters

**cp**

[**:class:~gwsumm.config.GWConfigParser**] customised configuration parser containing given section

**section**

[*str*] name of section to parse

**plotdir**

[*str*, optional, default: 'plots'] output path for plots, relative to current directory

### Returns

**tab**

[*DataTab*] a new *DataTab* defined from the configuration

```
process(nds=None, nproc=1, config=<GWSummConfigParser()>, datacache=None, datafind_error='raise',  
        **kwargs)
```

Process time accounting data

```
type = 'accounting'
```

```
write_state_html(state)
```

Write the HTML for the given state of this *GuardianTab*

## gwsumm.tabs.misc module

This module defines some utility *Tab* subclasses, including HTTP error handlers.

```
class gwsumm.tabs.misc.AboutTab(*args, **kwargs)
```

Bases: *Tab*

Page describing how the containing HTML pages were generated

```
type = 'about'
```

```
write_html(config=[], prog=None, **kwargs)
```

Write the HTML page for this *Tab*.

### Parameters

#### **maincontent**

[*str*, ~*MarkupPy.markup.page*] simple string content, or a structured *page* of markup to embed as the content of the #main div.

#### **title**

[*str*, optional, default: {parent.name}] level 1 heading for this *Tab*.

#### **subtitle**

[*str*, optional, default: {self.name}] level 2 heading for this *Tab*.

#### **tabs: `list`, optional**

list of top-level tabs (with children) to populate navbar

#### **ifo**

[*str*, optional] prefix for this IFO.

#### **ifomap**

[*dict*, optional] *dict* of (ifo, {base url}) pairs to map to summary pages for other IFOs.

#### **help\_**

[*str*, ~*MarkupPy.markup.page*, optional] non-menu content for navigation bar

#### **css**

[*list*, optional] list of resolvable URLs for CSS files. See *gwsumm.html.CSS* for the default list.

#### **js**

[*list*, optional] list of resolvable URLs for javascript files. See *gwumm.html.JS* for the default list.

#### **about**

[*str*, optional] href for the 'About' page

#### **footer**

[*str*, ~*MarkupPy.markup.page*] external link, if applicable (linked from an icon in the footer)

**issues**

[*bool* or *str*, default: *True*] print link to github.com issue tracker for this package

**\*\*inargs**

other keyword arguments to pass to the :meth:`~Tab.build\_inner\_html` method

**class** gwsumm.tabs.misc.**Error404Tab**(\*args, \*\*kwargs)

Bases: *Tab*

Custom HTTP 404 error page

**type** = '404'

**write\_html**(*config*=[], *top*=None, \*\*kwargs)

Write the HTML page for this *Tab*.

**Parameters**
**maincontent**

[*str*, ~MarkupPy.markup.page] simple string content, or a structured *page* of markup to embed as the content of the #main div.

**title**

[*str*, optional, default: {parent.name}] level 1 heading for this *Tab*.

**subtitle**

[*str*, optional, default: {self.name}] level 2 heading for this *Tab*.

**tabs: `list`, optional**

list of top-level tabs (with children) to populate navbar

**ifo**

[*str*, optional] prefix for this IFO.

**ifomap**

[*dict*, optional] *dict* of (ifo, {base url}) pairs to map to summary pages for other IFOs.

**help\_**

[*str*, ~MarkupPy.markup.page, optional] non-menu content for navigation bar

**css**

[*list*, optional] list of resolvable URLs for CSS files. See gwsumm.html.CSS for the default list.

**js**

[*list*, optional] list of resolvable URLs for javascript files. See gwumm.html.JS for the default list.

**about**

[*str*, optional] href for the 'About' page

**footer**

[*str*, ~MarkupPy.markup.page] external link, if applicable (linked from an icon in the footer)

**issues**

[*bool* or *str*, default: *True*] print link to github.com issue tracker for this package

**\*\*inargs**

other keyword arguments to pass to the :meth:`~Tab.build\_inner\_html` method

## gwsumm.tabs.registry module

Registry for GWSumm data tabs.

All Tabs should be registered for easy identification from the configuration INI files

`gwsumm.tabs.registry.get_tab(name)`

Query the registry for the tab class registered to the given name

`gwsumm.tabs.registry.register_tab(tab, name=None, force=False)`

Register a new summary *Tab* to the given name

### Parameters

#### **tab**

[*type*] defining Class for this tab type.

#### **name**

[*str*, optional] unique descriptive name for this type of tab, must not contain any spaces, e.g. 'hveto'. If *name=None*, the *Tab.type* class attribute of the given tab will be used.

#### **force**

[*bool*] overwrite existing registration for this type

### Raises

#### **ValueError**

if name is already registered and *force* not given as *True*

## gwsumm.tabs.sei module

*SummaryTab* for seismic watchdog monitoring

**class** `gwsumm.tabs.sei.SEIWatchDogTab(*args, **kwargs)`

Bases: [\*DataTab\*](#)

Summarise the WatchDog trips recorded from the SEI system.

**classmethod** `from_ini(config, section, plotdir='plots', **kwargs)`

Define a new *SEIWatchDogTab*.

**process** (*nds=None, nproc=1, config=<GWSummConfigParser()>, datacache=None, trigcache=None, datafind\_error='raise', \*\*kwargs*)

Process data for the given state.

**type** = 'seismic-watchdog'

**window** = 5

**write\_state\_html** (*state*)

Build HTML summary of watchdog trips

## gwsumm.tabs.stamp module

Custom *SummaryTab* for the output of the FScan algorithm.

**class** gwsumm.tabs.stamp.**StampPEMTab**(\*args, \*\*kwargs)

Bases: *DataTab*

Custom tab displaying a summary of StampPEM results.

**classmethod** **from\_ini**(config, section, \*\*kwargs)

Define a new *StampPEMTab* from a *ConfigParser*.

**process**(config=<GWSummConfigParser()>, \*\*kwargs)

Process data for this tab

### Parameters

**config**

[*ConfigParser.ConfigParser*, optional] job configuration to pass to *DataTab.finalize\_state*

**\*\*stateargs**

all other keyword arguments are passed directly onto the :meth:`~DataTab.process\_state` method.

**type** = 'stamp'

**write\_state\_html**(state)

Write the '#main' HTML content for this *StampPEMTab*.

## Module contents

This module defines the *Tab* API, and all of the built-in tab objects

## gwsumm.tests package

### Submodules

## gwsumm.tests.common module

Compatibility module

gwsumm.tests.common.**empty\_globalv\_CHANNELS**(f)

## gwsumm.tests.test\_archive module

Tests for *gwsumm.archive*

gwsumm.tests.test\_archive.**create**(data, \*\*metadata)

gwsumm.tests.test\_archive.**empty\_globalv**()

gwsumm.tests.test\_archive.**test\_archive\_load\_table**()

```
gwsumm.tests.test_archive.test_read_archive()
```

```
gwsumm.tests.test_archive.test_write_archive(delete=True)
```

## **gwsumm.tests.test\_batch module**

Tests for the *gwsumm.batch* command-line interface

```
gwsumm.tests.test_batch.test_main(krb, x509, tmpdir, caplog)
```

```
gwsumm.tests.test_batch.test_main_invalid_modes(capsys)
```

```
gwsumm.tests.test_batch.test_main_loop_over_modes(tmpdir, caplog, mode)
```

## **gwsumm.tests.test\_channels module**

Test suite

```
gwsumm.tests.test_channels.teardown_module()
```

Undo any `set_mode()` operations from this module

```
gwsumm.tests.test_channels.test_get_channel()
```

Test **:func:~gwsumm.channels.get\_channel~**

```
gwsumm.tests.test_channels.test_get_channel_trend()
```

Test `get_channel` for trends

*get\_channel* should query for the trend and the underlying raw channel

```
gwsumm.tests.test_channels.test_get_channels()
```

```
gwsumm.tests.test_channels.test_split(cstr, clist)
```

```
gwsumm.tests.test_channels.test_split_combination(cstr, clist)
```

```
gwsumm.tests.test_channels.test_update_missing_channel_params()
```

## **gwsumm.tests.test\_config module**

Tests for **:mod:~gwsumm.config~**

```
class gwsumm.tests.test_config.TestGWSummConfigParser
```

Bases: `object`

**PARSER**

alias of *GWSummConfigParser*

**classmethod** `cnfg()`

**classmethod** `new()`

**test\_configdir()**

**test\_finalize()**



```
test_from_configparser(cnfg)
test_get_css()
test_get_javascript()
test_init()
test_interpolate_section_names(cnfg)
test_load_channels()
test_load_plugins(cnfg)
test_load_rcParams()
test_load_states()
test_load_units(cnfg)
test_nditems(cnfg)
test_ndoptions(cnfg)
test_read()
test_set_date_options()
test_set_ifo_options(ifo, obs, exp)
gwsumm.tests.test_config.assert_configparser_equal(a, b)
```

### gwsumm.tests.test\_data module

Tests for *gwsumm.data*

```
class gwsumm.tests.test_data.TestData
    Bases: object
    Tests for :mod:`gwsumm.data`:
    classmethod setup_class()
    classmethod teardown_class()
    test_add_timeseries()
    test_find_frame_type()
    test_get_channel_type()
    test_get_coherence_spectrogram()
    test_get_coherence_spectrum()
    test_get_fftparams()
    test_get_spectrogram()
```

```
test_get_spectrum()
test_get_timeseries()
test_make_globalv_key()
test_parse_math_definition(definition, math)
```

```
gwsumm.tests.test_data.download(remote, target=None)
```

Download a file

### **gwsumm.tests.test\_mode module**

Test suite for the mode module

```
gwsumm.tests.test_mode.teardown_module()
    Undo any set_mode() operations from this module
gwsumm.tests.test_mode.test_get_base(m, basestr)
gwsumm.tests.test_mode.test_get_mode()
gwsumm.tests.test_mode.test_set_mode()
```

### **gwsumm.tests.test\_plot module**

Tests for *gwsumm.plot*

```
class gwsumm.tests.test_plot.TestDataPlot
    Bases: TestSummaryPlot
    DEFAULT_ARGS = [['X1:TEST-CHANNEL', 'Y1:TEST-CHANNEL2'], 0, 100]
    TYPE = 'data'
    test_add_channel()
    test_allchannels(plot)
    test_apply_parameters(plot)
    test_channels(plot)
    test_end(plot)
    test_finalize(plot)
    test_from_ini()
    test_get_channel_groups()
    test_href(plot)
    test_ifos(plot)
    test_init()
```

```
test_outputfile(plot)
test_parse_legend_kwargs(plot)
test_parse_plot_kwargs(plot)
test_parse_plot_kwargs_labels(plot, usetex, result)
test_parse_rcParams(plot)
test_span(plot)
test_start(plot)
test_state(plot)
test_tag(plot)
class gwsumm.tests.test_plot.TestSummaryPlot
    Bases: object
    DEFAULT_ARGS = []
    DEFAULT_KWARGS = {}
    TYPE = None
    classmethod create(*args, **kwargs)
    classmethod plot()
    classmethod setup_class()
    test_eq(plot)
    test_href(plot, url, href)
    test_init()
    test_new(plot, isnew, new)
    test_repr(plot)
    test_src(plot)
    test_str(plot)
gwsumm.tests.test_plot.test_get_column_label(column, label)
gwsumm.tests.test_plot.test_get_plot(name, plot)
gwsumm.tests.test_plot.test_registry_plot()
```

### gwsumm.tests.test\_tabs module

Tests for *gwsumm.tabs*

```
class gwsumm.tests.test_tabs.TestExternalTab
```

Bases: *TestTab*

```
DEFAULT_ARGS = ['Test', '//test.com']
```

```
TYPE = 'external'
```

```
test_init()
```

```
class gwsumm.tests.test_tabs.TestPlotTab
```

Bases: *TestTab*

```
TYPE = 'plots'
```

```
test_add_plot()
```

```
test_init()
```

```
test_layout()
```

```
class gwsumm.tests.test_tabs.TestTab
```

Bases: *object*

```
DEFAULT_ARGS = ['Test']
```

```
TYPE = 'basic'
```

```
create(*args, **kwargs)
```

```
classmethod setup_class()
```

```
test_index()
```

```
test_init()
```

```
test_shortcode()
```

```
gwsumm.tests.test_tabs.test_get_tab(name, tab)
```

```
gwsumm.tests.test_tabs.test_register_tab()
```

### gwsumm.tests.test\_utils module

Tests for *gwsumm.utils*

```
gwsumm.tests.test_utils.test_elapsed_time()
```

```
gwsumm.tests.test_utils.test_get_default_ifo(ifo, host)
```

```
gwsumm.tests.test_utils.test_get_odc_bitmask(chan, mask)
```

```
gwsumm.tests.test_utils.test_mkdir()
```

```
gwsumm.tests.test_utils.test_nat_sorted()
```

```
gwsumm.tests.test_utils.test_safe_eval(value, out)
```

```
gwsumm.tests.test_utils.test_safe_eval_2()
```

```
gwsumm.tests.test_utils.test_vprint(capsys)
```

## Module contents

Test suite

## Submodules

### gwsumm.archive module

This module handles HDF archiving of data.

In production for LIGO, the LIGO Summary Pages (LSP) Service runs at regular intervals (about every 10 minutes), so an HDF5 file is used to archive the data read and produced from one instance so that the next instance doesn't have to re-read and re-produce the same data.

All data products are stored just using the 'standard' gwpy `.write()` method for that object.

```
gwsumm.archive.archive_table(table, key, parent)
```

Add a table to the given HDF5 group

**Warning:** If the input `table` is empty, it will not be archived

#### Parameters

**table**

[~*astropy.table.Table*] the data to archive

**key**

[*str*] the path (relative to `parent`) at which to store the table

**parent**

[*h5py.Group*] the h5py group in which to add this dataset

```
gwsumm.archive.find_daily_archives(start, end, ifo, tag, basedir='.')
```

Find the daily archives spanning the given GPS [start, end) interval

#### Parameters

**start**

[*float*, ~*datetime.datetime*, ~*astropy.time.Time*, *str*] start time of the archive file to find, any object that can be converted into a *LIGOTimeGPS*, ~*astropy.time.Time*, or ~*datetime.datetime* is acceptable

**end**

[*float*, ~*datetime.datetime*, ~*astropy.time.Time*, *str*] end time of the archive file to find, any object that can be converted into a *LIGOTimeGPS*, ~*astropy.time.Time*, or ~*datetime.datetime* is acceptable

**ifo**

[*str*] interferometer string, ex. 'H1'

**tag**

[*str*] tag string for the archive file

**basedir**

[*path-like*, optional] base path to archive files, default: '.'

**Returns**

**archives**

[*list*] list of matching archive files

## Notes

This will only search the day directories with the format *YYYYMMDD*

`gwsumm.archive.load_table(dataset)`

Read table from the given HDF5 group

The *EventTable* is read, stored in the memory archive, then returned

**Parameters**

**dataset**

[*h5py.Dataset*] n-dimensional table to load from hdf5

**Returns**

**table**

[*~gwpy.table.EventTable*] the table of events loaded from hdf5

`gwsumm.archive.read_data_archive(sourcefile, rm_source_on_fail=True)`

Read archived data from an HDF5 archive source

This method reads all found data into the data containers defined by the *gwsumm.globalv* module, then returns nothing.

**Parameters**

**sourcefile**

[*str*] path to source HDF5 file

**rm\_source\_on\_fail**

[*bool*, optional] remove the source HDF5 file if there was an *OSError* when opening the file

`gwsumm.archive.segments_from_array(array)`

Convert a 2-dimensional *numpy.ndarray* to a *SegmentList*

**Parameters**

**array**

[*float numpy.ndarray*] input numpy array to convert into a segment list

**Returns**

**out**

[*~gwpy.segments.SegmentList*] output segment list

`gwsumm.archive.segments_to_array(segmentlist)`

Convert a *SegmentList* to a 2-dimensional *numpy.ndarray*

**Parameters**

**segmentlist**

[~gwpy.segments.SegmentList] input segment list to convert

## Returns

**out**

[float numpy.ndarray] output segment list as a numpy array

`gwsumm.archive.write_data_archive(outfile, channels=True, timeseries=True, spectrogram=True, segments=True, triggers=True)`

Build and save an HDF archive of data processed in this job.

## Parameters

**outfile**

[str] path to target HDF5 file

**timeseries**

[bool, optional] include *TimeSeries* data in archive

**spectrogram**

[bool, optional] include *Spectrogram* data in archive

**segments**

[bool, optional] include *DataQualityFlag* data in archive

**triggers**

[bool, optional] include *EventTable* data in archive

## gwsumm.batch module

Pipeline generator for the Gravitational-wave interferometer summary information system (*gwsumm*)

This module constructs a directed acyclic graph (DAG) that defines a workflow to be submitted via the HTCondor scheduler.

**class** `gwsumm.batch.GWHelpFormatter(*args, **kwargs)`

Bases: `HelpFormatter`

**class** `gwsumm.batch.GWSummaryDAGNode(job)`

Bases: `CondorDAGNode`

**get\_cmd\_line()**

Return the full command line that will be used when this node is run by DAGman.

**class** `gwsumm.batch.GWSummaryJob(universe, tag='gw_summary', subdir=None, logdir=None, **cmds)`

Bases: `CondorDAGJob`

Job representing a configurable instance of `gw_summary`.

**add\_opt(opt, value="")**

Add a command line option to the executable. The order that the arguments will be appended to the command line is not guaranteed, but they will always be added before any command line arguments. The name of the option is prefixed with double hyphen and the program is expected to parse it with `getopt_long()`.  
 @param opt: command line option to add. @param value: value to pass to the option (None for no argument).

**get\_command()**

```
logtag = '$(cluster)-$(process)'
```

```
set_command(command)
```

```
write_sub_file()
```

Write a submit file for this Condor job.

```
gwsumm.batch.create_parser()
```

Create a command-line parser for this entry point

```
gwsumm.batch.main(args=None)
```

Run the command-line Omega scan tool in batch mode

## gwsumm.channels module

Utilities for channel access

```
gwsumm.channels.get_channel(channel, find_parent=True, timeout=5)
```

Find (or create) a **`:class:`~gwpv.detector.Channel``**.

If `channel` has already been created, the cached copy will be returned, otherwise a new `~gwpv.detector.Channel` object will be created.

### Parameters

**channel**

[*str*] name of new channel

**find\_parent**

[*bool*, optional, default: *True*] query for raw version of trend channel (trends not in CIS)

**timeout**

[*float*, optional, default: 5] number of seconds to wait before connection times out

### Returns

**Channel**

**`:class:`~gwpv.detector.Channel``** new channel.

```
gwsumm.channels.get_channels(channels, **kwargs)
```

Find (or create) multiple channels calling `get_channel()`

### Parameters

**channels**

[*list*] list of channels as *str* or `~gwpv.detector.Channel` objects

**\*\*kwargs**

keyword arguments applied to each channel in the list

### Returns

**ChannelList**

[`~gwpv.detector.ChannelList`] a list of channels

See also:

**`:obj:`~get_channel``**



`gwsumm.channels.split(channelstring)`

Split a comma-separated list of channels that may, or may not contain NDS2 channel types as well

**Parameters**

**channelstring**

[*str*] comma-separated string of channels

**Returns**

**out**

[*list*] list of strings for each channel

`gwsumm.channels.split_combination(channelstring)`

Split a math-combination of channels

**Parameters**

**channelstring**

[*str*]

**Returns**

**ChannelList**

[~*gwpy.detector.ChannelList*]

`gwsumm.channels.update_channel_params()`

Update the *globalv.CHANNELS* list based on internal parameter changes

This is required to update *Channel.type* based on *Channel.frame\_type*, and similar.

`gwsumm.channels.update_missing_channel_params(channel, **kwargs)`

Update empty channel parameters using the given input

This method will only set parameters in the channel if the target parameter is *None*.

**Parameters**

**channel**

[~*gwpy.detector.Channel*] channel to update

**\*\*kwargs**

(*key*, *value*) pairs to set

**Returns**

**target**

[~*gwpy.detector.Channel*] the channel after updating parameters

## **gwsumm.globalv module**

Set of global memory variables for GWSumm package

## gwsumm.io module

Input/output utilities

`gwsumm.io.read_frequencyseries(filename)`

Read a `~gwp.py.frequencyseries.FrequencySeries` from a file

IF using HDF5, the filename can be given as a combined filename/path, i.e. `test.hdf5/path/to/dataset`.

### Parameters

**filename**

[*str*] path of file to read

### Returns

**series**

[`~gwp.py.frequencyseries.FrequencySeries`] the data as read

### Raises

**astropy.io.registry.IORegistryError**

if the input format cannot be identified or is not registered

## gwsumm.mode module

Job modes

`class gwsumm.mode.Mode(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)`

Bases: `OrderedEnum`

Enumeration of valid processing ‘modes’

Each mode provides an association with a particular GPS interval

**day** = 10

**dir\_format()**

**event** = 1

**gps** = 2

**is\_calendar()**

**month** = 12

**static** = 0

**week** = 11

**year** = 13

`class gwsumm.mode.OrderedEnum(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)`

Bases: `Enum`

`gwsumm.mode.get_base(date, mode=None)`

Determine the correct base attribute for the given date and mode.

#### Parameters

**date**

`[:class:datetime.datetime]` formatted date

**mode**

`[int, str]` enumerated interger code (or name) for the required mode

#### Returns

**base**

`[str]` the recommended base URL to have a correctly linked calendar

`gwsumm.mode.get_mode(m=None)`

Return the enum for the given mode, defaults to the current mode.

`gwsumm.mode.set_mode(m)`

Set the current mode.

## gwsumm.segments module

Utilities for segment handling and display

`gwsumm.segments.format_padding(flags, padding)`

Format an arbitrary collection of paddings into a *dict*

`gwsumm.segments.get_segments(flag, validity=None, config=<configparser.ConfigParser object>, cache=None, query=True, return_=True, coalesce=True, padding=None, ignore_undefined=False, segdb_error='raise', url=None, **read_kw)`

Retrieve the segments for a given flag

Segments will be loaded from global memory if already defined, otherwise they will be loaded from the given `:class:`~glue.lal.Cache``, or finally from the segment database

#### Parameters

**flag**

`[str, list]` either the name of one flag, or a list of names

**validity**

`[~gwpy.segments.SegmentList]` the segments over which to search for other segments

**query**

`[bool]`, optional, default: *True* actually execute a read/query operation (if needed), otherwise just retrieve segments that have already been cached

**config**

`[~configparser.ConfigParser]`, optional the configuration for your analysis, if you have one. If present the `[segment-database]` section will be queried for the following options

- `gps-start-time`, and `gps-end-time`, if `validity` is not given
- `url` (the remote hostname for the segment database) if the `url` keyword is not given

**cache**

`:class:`~glue.lal.Cache``, optional a cache of files from which to read segments, otherwise segments will be downloaded from the segment database

**coalesce**

[*bool*, optional, default: *True*] coalesce all segmentlists before returning, otherwise just return segments as they were downloaded/read

**padding**

[*tuple*, or *dict* of *tuples*, optional] (*start*, *end*) padding with which to pad segments that are downloaded/read

**ignore\_undefined**

[*bool*, optional, default: *False*] Special case needed for network calculation compound flags so that when this is *True*, *DataQualityFlag.known* values are set to the same value as *validity*

**segdb\_error**

[*str*, optional, default: *'raise'*] how to handle errors returned from the segment database, one of

- *'raise'* (default) : raise the exception as normal
- *'warn'* : print the exception as a warning, but return no segments
- *'ignore'* : silently ignore the error and return no segments

**url**

[*str*, optional] the remote hostname for the target segment database

**return\_**

[*bool*, optional, default: *True*] internal flag to enable (*True*) or disable (*False*) actually returning anything. This is useful if you want to download/read segments now but not use them until later (e.g. plotting)

**\*\*read\_kw**

[*dict*, optional] additional keyword arguments to *~gwpy.segments.DataQualityDict.read* or *~gwpy.segments.DataQualityFlag.read*

**Returns****flag**

[*~gwpy.segments.DataQualityFlag*] the flag object representing segments for the given single flag, OR

**flagdict**

[*~gwpy.segments.DataQualityDict*] the dict of *~gwpy.segments.DataQualityFlag* objects for multiple flags, if *flag* is given as a *list*, OR

**None**

if *return\_=False*

`gwsumm.segments.not_equal(a, b, f)`

`gwsumm.segments.split_compound_flag(compound)`

Parse the configuration for this state.

**Returns****flags**

[*tuple*] a 2-tuple containing lists of flags defining required ON and OFF segments respectively for this state

## gwsumm.triggers module

Read and store transient event triggers

`gwsumm.triggers.add_triggers(table, key, segments=None)`

Add a *EventTable* to the global memory cache

`gwsumm.triggers.get_etg_read_kwargs(etg, config=None, exclude=['columns'])`

Read keyword arguments to pass to the trigger reader for a given etg

`gwsumm.triggers.get_etg_table(etg)`

Find which table should be used for the given etg

### Parameters

**etg**

[*str*] name of Event Trigger Generator for which to query

### Returns

**table**

[*type*, subclass of `~ligo.lw.table.Table`] LIGO\_LW table registered to the given ETG

### Raises

**KeyError**

if the ETG is not registered

`gwsumm.triggers.get_time_column(table, etg)`

Get the time column name for this table

`gwsumm.triggers.get_times(table, etg)`

Get the time data for this table

See also:

**`:obj:~get_time_column``**

`gwsumm.triggers.get_triggers(channel, etg, segments, config=<GWSummConfigParser()>, cache=None, columns=None, format=None, query=True, nproc=1, ligolwtable=None, filter=None, timecolumn=None, verbose=False, return_=True)`

Read a table of transient event triggers for a given channel.

`gwsumm.triggers.keep_in_segments(table, segmentlist, etg=None)`

Return a view of the table containing only those rows in the segmentlist

`gwsumm.triggers.read_cache(cache, segments, etg, nproc=1, timecolumn=None, **kwargs)`

Read a table of events from a cache

This function is mainly meant for use from the `get_triggers` method

### Parameters

**cache**

**`:class:~glue.lal.Cache``** the formatted list of files to read

**segments**

[`~gwpy.segments.SegmentList`] the list of segments to read

**etg**

[*str*] the name of the trigger generator that created the files

**nproc**

[*int*, optional] the number of parallel processes to use when reading

**\*\*kwargs**

other keyword arguments are passed to the *EventTable.read* or *{tableclass}.read* methods

**Returns**

**table**

[~*gwpv.table.EventTable*, *None*] a table of events, or *None* if the cache has no overlap with the segments

## gwsumm.units module

Extra units for GW data processing

## gwsumm.utils module

Utilities for GWSumm

`gwsumm.utils.elapsed_time()`

Return the time (seconds) since this job started

`gwsumm.utils.get_default_ifo(fqdn='build-24123547-project-416166-gwsumm')`

Find the default interferometer prefix (IFO) for the given host

**Parameters**

**fqdn**

[*str*] the fully-qualified domain name (FQDN) of the host on which you wish to find the default IFO

**Returns**

**IFO**

[*str*] the upper-case X1-style prefix for the default IFO, if found, e.g. *L1*

**Raises**

**ValueError**

if not default interferometer prefix can be parsed

`gwsumm.utils.get_odc_bitmask(odcchannel)`

`gwsumm.utils.mkdir(*paths)`

Conditional mkdir operation, for convenience

`gwsumm.utils.nat_sorted(iterable, key=None)`

Sorted a list in the way that humans expect.

**Parameters**

**iterable**

[*iterable*] iterable to sort

**key**

[*callable*] sorting key

**Returns**

**lsorted**

[*list*] sorted() version of input 1

`gwsumm.utils.safe_eval(val, strict=False, globals_=None, locals_=None)`

Evaluate the given string as a line of python, if possible

If the **:meth:`eval`** fails, a *str* is returned instead, unless *strict=True* is given.

**Parameters****val**

[*str*] input text to evaluate

**strict**

[*bool*, optional, default: *False*] raise an exception when the *eval* call fails (*True*) otherwise return the input as a *str* (*False*, default)

**globals\_**

[*dict*, optional] dict of global variables to pass to *eval*, defaults to current *globals*

**locals\_**

[*dict*, optional] dict of local variables to pass to *eval*, defaults to current *locals*

---

**Note:** Note the trailing underscore on the *globals\_* and *locals\_* kwargs, this is required to not clash with the builtin *globals* and *locals* methods`.

---

**Raises****ValueError**

if the input string is considered unsafe to evaluate, normally meaning it contains something that might interact with the filesystem (e.g. *os.path* calls)

**NameError****SyntaxError**

if the input cannot be evaluated, and *strict=True* is given

See also:

**:obj:`eval`**

for more documentation on the underlying evaluation method

`gwsumm.utils.vprint(message, verbose=True, stream=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>, profile=True)`

Prints the given message to the stream.

**Parameters****message**

[*str*] string to print

**verbose**

[*bool*, optional, default: *True*] flag to print or not, default: print

**stream**

[*file*, optional, default: *stdout*] file object stream in which to print, default: *stdout*

**profile**

[*bool*, optional, default: *True*] flag to print timestamp for debugging and profiling purposes

## Module contents

Gravitational-wave interferometer summary information system

**EventTab**

**IntervalTab**

**StaticTab**

**gwsumm**



## PYTHON MODULE INDEX

### g

- gwsumm, 108
- gwsumm.archive, 97
- gwsumm.batch, 99
- gwsumm.channels, 100
- gwsumm.config, 17
- gwsumm.data, 26
- gwsumm.data.coherence, 19
- gwsumm.data.mathutils, 20
- gwsumm.data.range, 21
- gwsumm.data.spectral, 21
- gwsumm.data.timeseries, 22
- gwsumm.data.utils, 26
- gwsumm.globalv, 101
- gwsumm.html, 30
- gwsumm.html.bootstrap, 28
- gwsumm.html.html5, 29
- gwsumm.html.static, 30
- gwsumm.html.tests, 28
- gwsumm.html.tests.test\_bootstrap, 27
- gwsumm.html.tests.test\_html5, 27
- gwsumm.html.tests.test\_static, 28
- gwsumm.io, 102
- gwsumm.mode, 16
- gwsumm.plot, 15
- gwsumm.plot.builtin, 35
- gwsumm.plot.core, 41
- gwsumm.plot.guardian, 32
- gwsumm.plot.guardian.core, 31
- gwsumm.plot.guardian.tests, 31
- gwsumm.plot.guardian.tests.test\_main, 31
- gwsumm.plot.mixins, 44
- gwsumm.plot.noisebudget, 45
- gwsumm.plot.range, 46
- gwsumm.plot.registry, 50
- gwsumm.plot.segments, 50
- gwsumm.plot.sei, 57
- gwsumm.plot.triggers, 34
- gwsumm.plot.triggers.core, 32
- gwsumm.plot.triggers.tests, 32
- gwsumm.plot.triggers.tests.test\_main, 32
- gwsumm.plot.utils, 57
- gwsumm.segments, 103
- gwsumm.state, 15
- gwsumm.state.all, 58
- gwsumm.state.core, 59
- gwsumm.state.registry, 61
- gwsumm.tabs, 13
- gwsumm.tabs.builtin, 65
- gwsumm.tabs.core, 73
- gwsumm.tabs.data, 79
- gwsumm.tabs.etg, 83
- gwsumm.tabs.fscan, 85
- gwsumm.tabs.gracedb, 86
- gwsumm.tabs.guardian, 87
- gwsumm.tabs.management, 87
- gwsumm.tabs.misc, 88
- gwsumm.tabs.registry, 90
- gwsumm.tabs.sei, 90
- gwsumm.tabs.stamp, 91
- gwsumm.tests, 97
- gwsumm.tests.common, 91
- gwsumm.tests.test\_archive, 91
- gwsumm.tests.test\_batch, 92
- gwsumm.tests.test\_channels, 92
- gwsumm.tests.test\_config, 92
- gwsumm.tests.test\_data, 93
- gwsumm.tests.test\_mode, 94
- gwsumm.tests.test\_plot, 94
- gwsumm.tests.test\_tabs, 96
- gwsumm.tests.test\_utils, 96
- gwsumm.triggers, 105
- gwsumm.units, 106
- gwsumm.utils, 106



## A

AboutTab (class in `gwsomm.tabs.misc`), 88  
 AccountingTab (class in `gwsomm.tabs.management`), 87  
 add\_channel() (`gwsomm.plot.core.DataPlot` method), 41  
 add\_child() (`gwsomm.tabs.core.BaseTab` method), 73  
 add\_coherence\_component\_spectrogram() (in module `gwsomm.data.coherence`), 19  
 add\_flag() (`gwsomm.plot.segments.SegmentDataPlot` method), 53  
 add\_future\_shade() (`gwsomm.plot.builtin.TimeSeriesDataPlot` method), 38  
 add\_hvlines() (`gwsomm.plot.core.DataPlot` method), 41  
 add\_legend() (`gwsomm.plot.segments.SegmentDataPlot` method), 53  
 add\_loudest\_event() (`gwsomm.plot.triggers.core.TriggerDataPlot` method), 32  
 add\_opt() (`gwsomm.batch.GWSummaryJob` method), 99  
 add\_plot() (`gwsomm.tabs.builtin.PlotTab` method), 68  
 add\_spectrogram() (in module `gwsomm.data.spectral`), 21  
 add\_state() (`gwsomm.tabs.builtin.StateTab` method), 71  
 add\_state\_segments() (`gwsomm.plot.builtin.TimeSeriesDataPlot` method), 38  
 add\_timeseries() (in module `gwsomm.data.timeseries`), 22  
 add\_triggers() (in module `gwsomm.triggers`), 105  
 afterword (`gwsomm.tabs.builtin.PlotTab` property), 68  
 all\_adc() (in module `gwsomm.data.timeseries`), 22  
 allchannels (`gwsomm.plot.core.DataPlot` property), 41  
 allchannels (`gwsomm.plot.triggers.core.TriggerPlotMixin` property), 33  
 allflags (`gwsomm.plot.segments.SegmentDataPlot` property), 53  
 apply\_parameters() (`gwsomm.plot.core.DataPlot` method), 41

apply\_transfer\_function\_series() (in module `gwsomm.data.spectral`), 21  
 archive\_table() (in module `gwsomm.archive`), 97  
 assert\_configparser\_equal() (in module `gwsomm.tests.test_config`), 93

## B

banner() (in module `gwsomm.html.bootstrap`), 28  
 base\_map\_dropdown() (in module `gwsomm.html.bootstrap`), 28  
 BaseTab (class in `gwsomm.tabs.core`), 73

## C

calculate\_duty\_factor() (`gwsomm.plot.segments.DutyDataPlot` method), 50  
 calculate\_time\_volume() (`gwsomm.plot.range.SimpleTimeVolumeDataPlot` static method), 49  
 calendar() (in module `gwsomm.html.bootstrap`), 28  
 caption (`gwsomm.plot.core.SummaryPlot` property), 44  
 channels (`gwsomm.plot.core.DataPlot` property), 42  
 children (`gwsomm.tabs.core.BaseTab` property), 73  
 cnfg() (`gwsomm.tests.test_config.TestGWSummConfigParser` class method), 92  
 CoherenceSpectrogramDataPlot (class in `gwsomm.plot.builtin`), 35  
 CoherenceSpectrumDataPlot (class in `gwsomm.plot.builtin`), 35  
 combined\_time\_volume() (`gwsomm.plot.range.SimpleTimeVolumeDataPlot` method), 49  
 comments\_box() (in module `gwsomm.html.html5`), 29  
 common\_limits() (in module `gwsomm.plot.segments`), 56  
 complex\_percentile() (in module `gwsomm.data.coherence`), 19  
 copy() (`gwsomm.state.core.SummaryState` method), 59  
 copy() (`gwsomm.state.SummaryState` method), 63  
 create() (`gwsomm.tests.test_plot.TestSummaryPlot` class method), 95  
 create() (`gwsomm.tests.test_tabs.TestTab` method), 96

`create()` (in module `gwsomm.tests.test_archive`), 91  
`create_parser()` (in module `gwsomm.batch`), 100

## D

`data` (`gwsomm.plot.builtin.CoherenceSpectrogramDataPlot` attribute), 35  
`data` (`gwsomm.plot.builtin.CoherenceSpectrumDataPlot` attribute), 35  
`data` (`gwsomm.plot.builtin.RayleighSpectrogramDataPlot` attribute), 36  
`data` (`gwsomm.plot.builtin.RayleighSpectrumDataPlot` attribute), 36  
`data` (`gwsomm.plot.builtin.SpectralVarianceDataPlot` attribute), 37  
`data` (`gwsomm.plot.builtin.SpectrogramDataPlot` attribute), 37  
`data` (`gwsomm.plot.builtin.SpectrumDataPlot` attribute), 38  
`data` (`gwsomm.plot.builtin.TimeSeriesDataPlot` attribute), 39  
`data` (`gwsomm.plot.builtin.TimeSeriesHistogram2dDataPlot` attribute), 39  
`data` (`gwsomm.plot.builtin.TimeSeriesHistogramPlot` attribute), 40  
`data` (`gwsomm.plot.noisebudget.NoiseBudgetPlot` attribute), 45  
`data` (`gwsomm.plot.noisebudget.RelativeNoiseBudgetPlot` attribute), 45  
`data` (`gwsomm.plot.range.RangeCumulativeSpectrumDataPlot` attribute), 47  
`data` (`gwsomm.plot.range.RangePlotMixin` attribute), 48  
`data` (`gwsomm.plot.range.RangeSpectrumDataPlot` attribute), 48  
`data` (`gwsomm.plot.range.SimpleTimeVolumeDataPlot` attribute), 49  
`data` (`gwsomm.plot.segments.DutyDataPlot` attribute), 50  
`data` (`gwsomm.plot.segments.ODCDataPlot` attribute), 52  
`data` (`gwsomm.plot.segments.SegmentDataPlot` attribute), 53  
`data` (`gwsomm.plot.segments.SegmentHistogramPlot` attribute), 55  
`data` (`gwsomm.plot.segments.StateVectorDataPlot` attribute), 55  
`data` (`gwsomm.plot.sei.SeiWatchDogPlot` attribute), 57  
`data` (`gwsomm.plot.triggers.core.TriggerDataPlot` attribute), 32  
`data` (`gwsomm.plot.triggers.core.TriggerHistogramPlot` attribute), 33  
`data` (`gwsomm.plot.triggers.core.TriggerRateDataPlot` attribute), 34  
`data` (`gwsomm.plot.triggers.core.TriggerTimeSeriesDataPlot` attribute), 34  
`DataLabelSvgMixin` (class in `gwsomm.plot.mixins`), 44  
`DataPlot` (class in `gwsomm.plot.core`), 41  
`DataTab` (class in `gwsomm.tabs.data`), 79  
`day` (`gwsomm.mode.Mode` attribute), 102  
`DEFAULT_ARGS` (`gwsomm.tests.test_plot.TestDataPlot` attribute), 94  
`DEFAULT_ARGS` (`gwsomm.tests.test_plot.TestSummaryPlot` attribute), 95  
`DEFAULT_ARGS` (`gwsomm.tests.test_tabs.TestExternalTab` attribute), 96  
`DEFAULT_ARGS` (`gwsomm.tests.test_tabs.TestTab` attribute), 96  
`DEFAULT_KWARGS` (`gwsomm.tests.test_plot.TestSummaryPlot` attribute), 95  
`defaults` (`gwsomm.plot.builtin.CoherenceSpectrogramDataPlot` attribute), 35  
`defaults` (`gwsomm.plot.builtin.CoherenceSpectrumDataPlot` attribute), 35  
`defaults` (`gwsomm.plot.builtin.RayleighSpectrogramDataPlot` attribute), 36  
`defaults` (`gwsomm.plot.builtin.RayleighSpectrumDataPlot` attribute), 36  
`defaults` (`gwsomm.plot.builtin.SpectralVarianceDataPlot` attribute), 37  
`defaults` (`gwsomm.plot.builtin.SpectrogramDataPlot` attribute), 37  
`defaults` (`gwsomm.plot.builtin.SpectrumDataPlot` attribute), 38  
`defaults` (`gwsomm.plot.builtin.TimeSeriesDataPlot` attribute), 39  
`defaults` (`gwsomm.plot.builtin.TimeSeriesHistogram2dDataPlot` attribute), 39  
`defaults` (`gwsomm.plot.builtin.TimeSeriesHistogramPlot` attribute), 40  
`defaults` (`gwsomm.plot.core.DataPlot` attribute), 42  
`defaults` (`gwsomm.plot.guardian.core.GuardianStatePlot` attribute), 31  
`defaults` (`gwsomm.plot.noisebudget.NoiseBudgetPlot` attribute), 45  
`defaults` (`gwsomm.plot.noisebudget.RelativeNoiseBudgetPlot` attribute), 45  
`defaults` (`gwsomm.plot.range.RangeCumulativeHistogramPlot` attribute), 46  
`defaults` (`gwsomm.plot.range.RangeCumulativeSpectrumDataPlot` attribute), 47  
`defaults` (`gwsomm.plot.range.RangeDataHistogramPlot` attribute), 47  
`defaults` (`gwsomm.plot.range.RangeDataPlot` attribute), 47  
`defaults` (`gwsomm.plot.range.RangePlotMixin` attribute), 48  
`defaults` (`gwsomm.plot.range.RangeSpectrogramDataPlot` attribute), 48  
`defaults` (`gwsomm.plot.range.RangeSpectrumDataPlot` attribute), 48

[attribute](#), 49  
[defaults \(gwsomm.plot.range.SimpleTimeVolumeDataPlotdraw\(\) attribute\)](#), 49  
[defaults \(gwsomm.plot.segments.DutyDataPlot attribute\)](#), 50  
[defaults \(gwsomm.plot.segments.NetworkDutyBarPlot attribute\)](#), 51  
[defaults \(gwsomm.plot.segments.NetworkDutyPiePlot attribute\)](#), 52  
[defaults \(gwsomm.plot.segments.ODCDataPlot attribute\)](#), 52  
[defaults \(gwsomm.plot.segments.SegmentBarPlot attribute\)](#), 53  
[defaults \(gwsomm.plot.segments.SegmentDataPlot attribute\)](#), 53  
[defaults \(gwsomm.plot.segments.SegmentHistogramPlot attribute\)](#), 55  
[defaults \(gwsomm.plot.segments.SegmentPiePlot attribute\)](#), 55  
[defaults \(gwsomm.plot.segments.StateVectorDataPlot attribute\)](#), 56  
[defaults \(gwsomm.plot.triggers.core.TriggerDataPlot attribute\)](#), 32  
[defaults \(gwsomm.plot.triggers.core.TriggerRateDataPlot attribute\)](#), 34  
[defaultstate \(gwsomm.tabs.builtin.StateTab property\)](#), 71  
[definition \(gwsomm.state.core.SummaryState property\)](#), 59  
[definition \(gwsomm.state.SummaryState property\)](#), 63  
[dialog\\_box\(\)](#) (in module [gwsomm.html.html5](#)), 29  
[dict\(\)](#) ([gwsomm.data.utils.FftParams](#) method), 26  
[dir\\_format\(\)](#) ([gwsomm.mode.Mode](#) method), 102  
[download\(\)](#) (in module [gwsomm.tests.test\\_data](#)), 94  
[draw\(\)](#) ([gwsomm.plot.builtin.SpectrogramDataPlot](#) method), 37  
[draw\(\)](#) ([gwsomm.plot.builtin.SpectrumDataPlot](#) method), 38  
[draw\(\)](#) ([gwsomm.plot.builtin.TimeSeriesDataPlot](#) method), 39  
[draw\(\)](#) ([gwsomm.plot.builtin.TimeSeriesHistogram2dDataPlot](#) method), 39  
[draw\(\)](#) ([gwsomm.plot.builtin.TimeSeriesHistogramPlot](#) method), 40  
[draw\(\)](#) ([gwsomm.plot.core.DataPlot](#) method), 42  
[draw\(\)](#) ([gwsomm.plot.guardian.core.GuardianStatePlot](#) method), 31  
[draw\(\)](#) ([gwsomm.plot.range.RangePlotMixin](#) method), 48  
[draw\(\)](#) ([gwsomm.plot.range.SimpleTimeVolumeDataPlot](#) method), 49  
[draw\(\)](#) ([gwsomm.plot.segments.DutyDataPlot](#) method), 51  
[draw\(\)](#) ([gwsomm.plot.segments.NetworkDutyBarPlot](#) method), 51  
[draw\(\)](#) ([gwsomm.plot.segments.NetworkDutyPiePlot](#) method), 52  
[draw\(\)](#) ([gwsomm.plot.segments.ODCDataPlot](#) method), 52  
[draw\(\)](#) ([gwsomm.plot.segments.SegmentBarPlot](#) method), 53  
[draw\(\)](#) ([gwsomm.plot.segments.SegmentDataPlot](#) method), 54  
[draw\(\)](#) ([gwsomm.plot.segments.SegmentHistogramPlot](#) method), 55  
[draw\(\)](#) ([gwsomm.plot.segments.SegmentPiePlot](#) method), 55  
[draw\(\)](#) ([gwsomm.plot.segments.StateVectorDataPlot](#) method), 56  
[draw\(\)](#) ([gwsomm.plot.sei.SeiWatchDogPlot](#) method), 57  
[draw\(\)](#) ([gwsomm.plot.triggers.core.TriggerDataPlot](#) method), 33  
[draw\(\)](#) ([gwsomm.plot.triggers.core.TriggerHistogramPlot](#) method), 33  
[draw\(\)](#) ([gwsomm.plot.triggers.core.TriggerRateDataPlot](#) method), 34  
[draw\(\)](#) ([gwsomm.plot.triggers.core.TriggerTimeSeriesDataPlot](#) method), 34  
[DRAW\\_PARAMS \(gwsomm.plot.core.DataPlot attribute\)](#), 41  
[DRAW\\_PARAMS \(gwsomm.plot.range.SimpleTimeVolumeDataPlot attribute\)](#), 49  
[DRAW\\_PARAMS \(gwsomm.plot.segments.SegmentDataPlot attribute\)](#), 53  
[DRAW\\_PARAMS \(gwsomm.plot.segments.StateVectorDataPlot attribute\)](#), 55  
[draw\\_svg\(\)](#) ([gwsomm.plot.mixins.DataLabelSvgMixin](#) method), 44  
[draw\\_svg\(\)](#) ([gwsomm.plot.mixins.SegmentLabelSvgMixin](#) method), 44  
[draw\\_svg\(\)](#) ([gwsomm.plot.mixins.SvgMixin](#) method), 44  
[DutyDataPlot](#) (class in [gwsomm.plot.segments](#)), 50

## E

[elapsed\\_time\(\)](#) (in module [gwsomm.utils](#)), 106  
[empty\\_globalv\(\)](#) (in module [gwsomm.tests.test\\_archive](#)), 91  
[empty\\_globalv\\_CHANNELS\(\)](#) (in module [gwsomm.tests.common](#)), 91  
[end \(gwsomm.plot.core.DataPlot property\)](#), 42  
[end \(gwsomm.state.core.SummaryState property\)](#), 60  
[end \(gwsomm.state.SummaryState property\)](#), 63  
[Error404Tab](#) (class in [gwsomm.tabs.misc](#)), 89  
[event \(gwsomm.mode.Mode attribute\)](#), 102  
[EventTriggerTab](#) (class in [gwsomm.tabs.etg](#)), 83  
[exclude\\_short\\_trend\\_segments\(\)](#) (in module [gwsomm.data.timeseries](#)), 22  
[ExternalTab](#) (class in [gwsomm.tabs.builtin](#)), 65

## F

[fetch\(\)](#) ([gwsumm.state.core.SummaryState](#) method), 60  
[fetch\(\)](#) ([gwsumm.state.SummaryState](#) method), 63  
[fftlength](#) ([gwsumm.data.utils.FftParams](#) attribute), 26  
[FftParams](#) (class in [gwsumm.data.utils](#)), 26  
[filter\\_timeseries\(\)](#) (in module [gwsumm.data.timeseries](#)), 22  
[finalize\(\)](#) ([gwsumm.config.GWSummConfigParser](#) method), 17  
[finalize\(\)](#) ([gwsumm.plot.core.DataPlot](#) method), 42  
[finalize\(\)](#) ([gwsumm.plot.mixins.SvgMixin](#) method), 44  
[finalize\\_states\(\)](#) ([gwsumm.tabs.data.DataTab](#) method), 79  
[finalize\\_states\(\)](#) ([gwsumm.tabs.etg.EventTriggerTab](#) method), 83  
[finalize\\_svg\(\)](#) ([gwsumm.plot.mixins.SvgMixin](#) method), 44  
[find\\_best\\_frames\(\)](#) (in module [gwsumm.data.timeseries](#)), 22  
[find\\_daily\\_archives\(\)](#) (in module [gwsumm.archive](#)), 97  
[find\\_frame\\_type\(\)](#) (in module [gwsumm.data.timeseries](#)), 22  
[find\\_frames\(\)](#) (in module [gwsumm.data.timeseries](#)), 22  
[flag](#) ([gwsumm.plot.segments.StateVectorDataPlot](#) property), 56  
[flags](#) ([gwsumm.plot.segments.SegmentDataPlot](#) property), 54  
[foreword](#) ([gwsumm.tabs.builtin.PlotTab](#) property), 68  
[format\\_padding\(\)](#) (in module [gwsumm.segments](#)), 103  
[frame\\_trend\\_type\(\)](#) (in module [gwsumm.data.timeseries](#)), 23  
[frames](#) ([gwsumm.tabs.builtin.StateTab](#) property), 71  
[from\\_configparser\(\)](#) ([gwsumm.config.GWSummConfigParser](#) class method), 18  
[from\\_ini\(\)](#) ([gwsumm.plot.core.DataPlot](#) class method), 42  
[from\\_ini\(\)](#) ([gwsumm.plot.core.SummaryPlot](#) class method), 44  
[from\\_ini\(\)](#) ([gwsumm.plot.range.SimpleTimeVolumeDataPlot](#) class method), 50  
[from\\_ini\(\)](#) ([gwsumm.plot.segments.SegmentDataPlot](#) class method), 54  
[from\\_ini\(\)](#) ([gwsumm.state.core.SummaryState](#) class method), 60  
[from\\_ini\(\)](#) ([gwsumm.state.SummaryState](#) class method), 63  
[from\\_ini\(\)](#) ([gwsumm.tabs.builtin.ExternalTab](#) class method), 66  
[from\\_ini\(\)](#) ([gwsumm.tabs.builtin.PlotTab](#) class method), 68

[from\\_ini\(\)](#) ([gwsumm.tabs.builtin.StateTab](#) class method), 71  
[from\\_ini\(\)](#) ([gwsumm.tabs.core.BaseTab](#) class method), 73  
[from\\_ini\(\)](#) ([gwsumm.tabs.core.TabList](#) class method), 78  
[from\\_ini\(\)](#) ([gwsumm.tabs.data.DataTab](#) class method), 79  
[from\\_ini\(\)](#) ([gwsumm.tabs.etg.EventTriggerTab](#) class method), 83  
[from\\_ini\(\)](#) ([gwsumm.tabs.fscan.FscanTab](#) class method), 85  
[from\\_ini\(\)](#) ([gwsumm.tabs.gracedb.GraceDbTab](#) class method), 86  
[from\\_ini\(\)](#) ([gwsumm.tabs.guardian.GuardianTab](#) class method), 87  
[from\\_ini\(\)](#) ([gwsumm.tabs.management.AccountingTab](#) class method), 87  
[from\\_ini\(\)](#) ([gwsumm.tabs.sei.SEIWatchDogTab](#) class method), 90  
[from\\_ini\(\)](#) ([gwsumm.tabs.stamp.StampPEMTab](#) class method), 91  
[FscanTab](#) (class in [gwsumm.tabs.fscan](#)), 85

## G

[generate\\_all\\_state\(\)](#) (in module [gwsumm.state](#)), 64  
[generate\\_all\\_state\(\)](#) (in module [gwsumm.state.all](#)), 58  
[get\\_base\(\)](#) (in module [gwsumm.mode](#)), 102  
[get\\_bins\(\)](#) ([gwsumm.plot.segments.DutyDataPlot](#) method), 51  
[get\\_bitmask\\_channels\(\)](#) ([gwsumm.plot.segments.ODCDataPlot](#) method), 52  
[get\\_channel\(\)](#) (in module [gwsumm.channels](#)), 100  
[get\\_channel\\_groups\(\)](#) ([gwsumm.plot.builtin.CoherenceSpectrumDataPlot](#) method), 36  
[get\\_channel\\_groups\(\)](#) ([gwsumm.plot.core.DataPlot](#) method), 42  
[get\\_channel\\_groups\(\)](#) ([gwsumm.plot.segments.SegmentDataPlot](#) method), 54  
[get\\_channel\\_type\(\)](#) (in module [gwsumm.data.timeseries](#)), 23  
[get\\_channels\(\)](#) ([gwsumm.tabs.data.DataTab](#) method), 80  
[get\\_channels\(\)](#) (in module [gwsumm.channels](#)), 100  
[get\\_child\(\)](#) ([gwsumm.tabs.core.BaseTab](#) method), 74  
[get\\_cmd\\_line\(\)](#) ([gwsumm.batch.GWSummDAGNode](#) method), 99  
[get\\_coherence\\_spectrogram\(\)](#) (in module [gwsumm.data.coherence](#)), 19



[get\\_coherence\\_spectrograms\(\)](#) (in module `gwsumm.data.coherence`), 19  
[get\\_coherence\\_spectrum\(\)](#) (in module `gwsumm.data.coherence`), 20  
[get\\_column\\_label\(\)](#) (in module `gwsumm.plot.utils`), 57  
[get\\_column\\_string\(\)](#) (in module `gwsumm.plot.utils`), 57  
[get\\_command\(\)](#) (`gwsumm.batch.GWSummaryJob` method), 99  
[get\\_css\(\)](#) (`gwsumm.config.GWSummConfigParser` method), 18  
[get\\_css\(\)](#) (in module `gwsumm.html.static`), 30  
[get\\_default\\_ifo\(\)](#) (in module `gwsumm.utils`), 106  
[get\\_etg\\_read\\_kwargs\(\)](#) (in module `gwsumm.triggers`), 105  
[get\\_etg\\_table\(\)](#) (in module `gwsumm.triggers`), 105  
[get\\_fftparams\(\)](#) (in module `gwsumm.data.utils`), 26  
[get\\_flags\(\)](#) (`gwsumm.tabs.data.DataTab` method), 80  
[get\\_hierarchy\(\)](#) (`gwsumm.tabs.core.TabList` method), 78  
[get\\_javascript\(\)](#) (`gwsumm.config.GWSummConfigParser` method), 18  
[get\\_js\(\)](#) (in module `gwsumm.html.static`), 30  
[get\\_mode\(\)](#) (in module `gwsumm.mode`), 103  
[get\\_odc\\_bitmask\(\)](#) (in module `gwsumm.utils`), 106  
[get\\_operator\(\)](#) (in module `gwsumm.data.mathutils`), 20  
[get\\_plot\(\)](#) (in module `gwsumm.plot.registry`), 50  
[get\\_range\(\)](#) (in module `gwsumm.data.range`), 21  
[get\\_range\\_channel\(\)](#) (in module `gwsumm.data.range`), 21  
[get\\_range\\_spectrogram\(\)](#) (in module `gwsumm.data.range`), 21  
[get\\_range\\_spectrum\(\)](#) (in module `gwsumm.data.range`), 21  
[get\\_ratio\(\)](#) (`gwsumm.plot.builtin.SpectrogramDataPlot` method), 37  
[get\\_segment\\_color\(\)](#) (`gwsumm.plot.segments.SegmentDataPlot` method), 54  
[get\\_segment\\_color\(\)](#) (`gwsumm.plot.segments.StateVectorDataPlot` method), 56  
[get\\_segments\(\)](#) (in module `gwsumm.segments`), 103  
[get\\_spectrogram\(\)](#) (in module `gwsumm.data.spectral`), 21  
[get\\_spectrograms\(\)](#) (in module `gwsumm.data.spectral`), 21  
[get\\_spectrum\(\)](#) (in module `gwsumm.data.spectral`), 21  
[get\\_state\(\)](#) (in module `gwsumm.state`), 64  
[get\\_state\(\)](#) (in module `gwsumm.state.registry`), 61  
[get\\_states\(\)](#) (in module `gwsumm.state`), 64  
[get\\_states\(\)](#) (in module `gwsumm.state.registry`), 61  
[get\\_tab\(\)](#) (in module `gwsumm.tabs.registry`), 90  
[get\\_time\\_column\(\)](#) (in module `gwsumm.triggers`), 105  
[get\\_times\(\)](#) (in module `gwsumm.triggers`), 105  
[get\\_timeseries\(\)](#) (in module `gwsumm.data.timeseries`), 23  
[get\\_timeseries\\_dict\(\)](#) (in module `gwsumm.data.timeseries`), 24  
[get\\_triggers\(\)](#) (`gwsumm.tabs.data.DataTab` method), 80  
[get\\_triggers\(\)](#) (in module `gwsumm.triggers`), 105  
[get\\_with\\_math\(\)](#) (in module `gwsumm.data.mathutils`), 20  
[gps](#) (`gwsumm.mode.Mode` attribute), 102  
[GraceDbTab](#) (class in `gwsumm.tabs.gracedb`), 86  
[group](#) (`gwsumm.tabs.core.BaseTab` property), 75  
[GuardianStatePlot](#) (class in `gwsumm.plot.guardian.core`), 31  
[GuardianTab](#) (class in `gwsumm.tabs.guardian`), 87  
[GWHelpFormatter](#) (class in `gwsumm.batch`), 99  
[GWpyTimeVolumeDataPlot](#) (class in `gwsumm.plot.range`), 46  
[gwsumm](#)  
     module, 108  
[gwsumm.archive](#)  
     module, 97  
[gwsumm.batch](#)  
     module, 99  
[gwsumm.channels](#)  
     module, 100  
[gwsumm.config](#)  
     module, 17  
[gwsumm.data](#)  
     module, 26  
[gwsumm.data.coherence](#)  
     module, 19  
[gwsumm.data.mathutils](#)  
     module, 20  
[gwsumm.data.range](#)  
     module, 21  
[gwsumm.data.spectral](#)  
     module, 21  
[gwsumm.data.timeseries](#)  
     module, 22  
[gwsumm.data.utils](#)  
     module, 26  
[gwsumm.globalv](#)  
     module, 101  
[gwsumm.html](#)  
     module, 30  
[gwsumm.html.bootstrap](#)  
     module, 28  
[gwsumm.html.html5](#)  
     module, 29

gwsumm.html.static  
    module, 30

gwsumm.html.tests  
    module, 28

gwsumm.html.tests.test\_bootstrap  
    module, 27

gwsumm.html.tests.test\_html5  
    module, 27

gwsumm.html.tests.test\_static  
    module, 28

gwsumm.io  
    module, 102

gwsumm.mode  
    module, 16, 102

gwsumm.plot  
    module, 15, 58

gwsumm.plot.builtin  
    module, 35

gwsumm.plot.core  
    module, 41

gwsumm.plot.guardian  
    module, 32

gwsumm.plot.guardian.core  
    module, 31

gwsumm.plot.guardian.tests  
    module, 31

gwsumm.plot.guardian.tests.test\_main  
    module, 31

gwsumm.plot.mixins  
    module, 44

gwsumm.plot.noisebudget  
    module, 45

gwsumm.plot.range  
    module, 46

gwsumm.plot.registry  
    module, 50

gwsumm.plot.segments  
    module, 50

gwsumm.plot.sei  
    module, 57

gwsumm.plot.triggers  
    module, 34

gwsumm.plot.triggers.core  
    module, 32

gwsumm.plot.triggers.tests  
    module, 32

gwsumm.plot.triggers.tests.test\_main  
    module, 32

gwsumm.plot.utils  
    module, 57

gwsumm.segments  
    module, 103

gwsumm.state  
    module, 15, 62

gwsumm.state.all  
    module, 58

gwsumm.state.core  
    module, 59

gwsumm.state.registry  
    module, 61

gwsumm.tabs  
    module, 13, 91

gwsumm.tabs.builtin  
    module, 65

gwsumm.tabs.core  
    module, 73

gwsumm.tabs.data  
    module, 79

gwsumm.tabs.etg  
    module, 83

gwsumm.tabs.fscan  
    module, 85

gwsumm.tabs.gracedb  
    module, 86

gwsumm.tabs.guardian  
    module, 87

gwsumm.tabs.management  
    module, 87

gwsumm.tabs.misc  
    module, 88

gwsumm.tabs.registry  
    module, 90

gwsumm.tabs.sei  
    module, 90

gwsumm.tabs.stamp  
    module, 91

gwsumm.tests  
    module, 97

gwsumm.tests.common  
    module, 91

gwsumm.tests.test\_archive  
    module, 91

gwsumm.tests.test\_batch  
    module, 92

gwsumm.tests.test\_channels  
    module, 92

gwsumm.tests.test\_config  
    module, 92

gwsumm.tests.test\_data  
    module, 93

gwsumm.tests.test\_mode  
    module, 94

gwsumm.tests.test\_plot  
    module, 94

gwsumm.tests.test\_tabs  
    module, 96

gwsumm.tests.test\_utils  
    module, 96



gwsumm.triggers  
 module, 105  
 gwsumm.units  
 module, 106  
 gwsumm.utils  
 module, 106

GWSummDAGNode (class in gwsumm.batch), 99  
 GWSummJob (class in gwsumm.batch), 99  
 GWSummConfigParser (class in gwsumm.config), 17

## H

hash() (in module gwsumm.plot.utils), 57  
 href (gwsumm.plot.core.DataPlot property), 42  
 href (gwsumm.plot.core.SummaryPlot property), 44  
 href (gwsumm.tabs.core.BaseTab property), 75  
 html\_banner() (gwsumm.tabs.core.BaseTab method), 75  
 html\_content() (gwsumm.tabs.builtin.ExternalTab method), 66  
 html\_content() (gwsumm.tabs.builtin.PlotTab method), 68  
 html\_content() (gwsumm.tabs.builtin.StateTab static method), 71  
 html\_content() (gwsumm.tabs.core.BaseTab static method), 75  
 html\_content() (gwsumm.tabs.data.DataTab method), 80  
 html\_navbar() (gwsumm.tabs.builtin.StateTab method), 71  
 html\_navbar() (gwsumm.tabs.core.BaseTab method), 75

## I

ifo (gwsumm.plot.guardian.core.GuardianStatePlot property), 31  
 ifos (gwsumm.plot.core.DataPlot property), 42  
 ifos (gwsumm.plot.segments.SegmentDataPlot property), 54  
 index (gwsumm.tabs.core.BaseTab property), 76  
 init\_plot() (gwsumm.plot.builtin.TimeSeriesDataPlot method), 39  
 init\_plot() (gwsumm.plot.builtin.TimeSeriesHistogramPlot method), 40  
 init\_plot() (gwsumm.plot.core.DataPlot method), 42  
 init\_plot() (gwsumm.plot.segments.SegmentDataPlot method), 54  
 init\_plot() (gwsumm.plot.segments.StateVectorDataPlot method), 56  
 interpolate\_section\_names() (gwsumm.config.GWSummConfigParser method), 18  
 is\_calendar() (gwsumm.mode.Mode method), 102

## K

keep\_in\_segments() (in module gwsumm.triggers), 105  
 key (gwsumm.state.core.SummaryState property), 60  
 key (gwsumm.state.SummaryState property), 64

## L

layout (gwsumm.tabs.builtin.PlotTab property), 68  
 ldvw\_qscan() (in module gwsumm.html.html5), 29  
 load() (in module gwsumm.html.html5), 29  
 load\_channels() (gwsumm.config.GWSummConfigParser method), 18  
 load\_plugins() (gwsumm.config.GWSummConfigParser method), 18  
 load\_rcParams() (gwsumm.config.GWSummConfigParser method), 18  
 load\_state() (in module gwsumm.html.html5), 30  
 load\_states() (gwsumm.config.GWSummConfigParser method), 18  
 load\_table() (in module gwsumm.archive), 98  
 load\_units() (gwsumm.config.GWSummConfigParser method), 18  
 locate\_data() (in module gwsumm.data.timeseries), 25  
 logtag (gwsumm.batch.GWSummJob attribute), 99  
 logx (gwsumm.plot.core.DataPlot property), 43  
 logy (gwsumm.plot.core.DataPlot property), 43

## M

main() (in module gwsumm.batch), 100  
 make\_globalv\_key() (in module gwsumm.data.utils), 26  
 MATH\_DEFINITION (gwsumm.state.core.SummaryState attribute), 59  
 MATH\_DEFINITION (gwsumm.state.SummaryState attribute), 63  
 method (gwsumm.data.utils.FftParams attribute), 26  
 mkdir() (in module gwsumm.utils), 106  
 Mode (class in gwsumm.mode), 102  
 mode (gwsumm.tabs.core.BaseTab property), 76  
 module  
 gwsumm, 108  
 gwsumm.archive, 97  
 gwsumm.batch, 99  
 gwsumm.channels, 100  
 gwsumm.config, 17  
 gwsumm.data, 26  
 gwsumm.data.coherence, 19  
 gwsumm.data.mathutils, 20  
 gwsumm.data.range, 21  
 gwsumm.data.spectral, 21

- `gwsumm.data.timeseries`, 22
- `gwsumm.data.utils`, 26
- `gwsumm.globalv`, 101
- `gwsumm.html`, 30
- `gwsumm.html.bootstrap`, 28
- `gwsumm.html.html5`, 29
- `gwsumm.html.static`, 30
- `gwsumm.html.tests`, 28
- `gwsumm.html.tests.test_bootstrap`, 27
- `gwsumm.html.tests.test_html5`, 27
- `gwsumm.html.tests.test_static`, 28
- `gwsumm.io`, 102
- `gwsumm.mode`, 16, 102
- `gwsumm.plot`, 15, 58
- `gwsumm.plot.builtin`, 35
- `gwsumm.plot.core`, 41
- `gwsumm.plot.guardian`, 32
- `gwsumm.plot.guardian.core`, 31
- `gwsumm.plot.guardian.tests`, 31
- `gwsumm.plot.guardian.tests.test_main`, 31
- `gwsumm.plot.mixins`, 44
- `gwsumm.plot.noisebudget`, 45
- `gwsumm.plot.range`, 46
- `gwsumm.plot.registry`, 50
- `gwsumm.plot.segments`, 50
- `gwsumm.plot.sei`, 57
- `gwsumm.plot.triggers`, 34
- `gwsumm.plot.triggers.core`, 32
- `gwsumm.plot.triggers.tests`, 32
- `gwsumm.plot.triggers.tests.test_main`, 32
- `gwsumm.plot.utils`, 57
- `gwsumm.segments`, 103
- `gwsumm.state`, 15, 62
- `gwsumm.state.all`, 58
- `gwsumm.state.core`, 59
- `gwsumm.state.registry`, 61
- `gwsumm.tabs`, 13, 91
- `gwsumm.tabs.builtin`, 65
- `gwsumm.tabs.core`, 73
- `gwsumm.tabs.data`, 79
- `gwsumm.tabs.etg`, 83
- `gwsumm.tabs.fscan`, 85
- `gwsumm.tabs.gracedb`, 86
- `gwsumm.tabs.guardian`, 87
- `gwsumm.tabs.management`, 87
- `gwsumm.tabs.misc`, 88
- `gwsumm.tabs.registry`, 90
- `gwsumm.tabs.sei`, 90
- `gwsumm.tabs.stamp`, 91
- `gwsumm.tests`, 97
- `gwsumm.tests.common`, 91
- `gwsumm.tests.test_archive`, 91
- `gwsumm.tests.test_batch`, 92
- `gwsumm.tests.test_channels`, 92

- `gwsumm.tests.test_config`, 92
- `gwsumm.tests.test_data`, 93
- `gwsumm.tests.test_mode`, 94
- `gwsumm.tests.test_plot`, 94
- `gwsumm.tests.test_tabs`, 96
- `gwsumm.tests.test_utils`, 96
- `gwsumm.triggers`, 105
- `gwsumm.units`, 106
- `gwsumm.utils`, 106

- `month` (`gwsumm.mode.Mode` attribute), 102

## N

- `name` (`gwsumm.state.core.SummaryState` property), 60
- `name` (`gwsumm.state.SummaryState` property), 64
- `name` (`gwsumm.tabs.core.BaseTab` property), 76
- `nat_sorted()` (in module `gwsumm.utils`), 106
- `nditems()` (`gwsumm.config.GWSummConfigParser` method), 18
- `ndoptions()` (`gwsumm.config.GWSummConfigParser` method), 18
- `NETWORK_COLOR` (`gwsumm.plot.segments.NetworkDutyBarPlot` attribute), 51
- `NETWORK_COLOR` (`gwsumm.plot.segments.NetworkDutyPiePlot` attribute), 52
- `NETWORK_NAME` (`gwsumm.plot.segments.NetworkDutyBarPlot` attribute), 51
- `NETWORK_NAME` (`gwsumm.plot.segments.NetworkDutyPiePlot` attribute), 52
- `NetworkDutyBarPlot` (class in `gwsumm.plot.segments`), 51
- `NetworkDutyPiePlot` (class in `gwsumm.plot.segments`), 51
- `new` (`gwsumm.plot.core.SummaryPlot` property), 44
- `new()` (`gwsumm.tests.test_config.TestGWSummConfigParser` class method), 92
- `node` (`gwsumm.plot.guardian.core.GuardianStatePlot` property), 31
- `NoiseBudgetPlot` (class in `gwsumm.plot.noisebudget`), 45
- `not_equal()` (in module `gwsumm.segments`), 104
- `notes` (`gwsumm.tabs.core.BaseTab` property), 76

## O

- `ODCDataPlot` (class in `gwsumm.plot.segments`), 52
- `OPTCRE` (`gwsumm.config.GWSummConfigParser` attribute), 17
- `optionxform` (`gwsumm.config.GWSummConfigParser` attribute), 18
- `OrderedEnum` (class in `gwsumm.mode`), 102
- `outputfile` (`gwsumm.plot.core.DataPlot` property), 43
- `outputfile` (`gwsumm.plot.sei.SeiWatchDogPlot` property), 57
- `overlap` (`gwsumm.data.utils.FftParams` attribute), 26
- `overlay` (`gwsumm.tabs.core.BaseTab` property), 76

`overlay_canvas()` (in module `gwsomm.html.html5`), 30

## P

`padding` (`gwsomm.plot.segments.SegmentDataPlot` property), 54

`parent` (`gwsomm.tabs.core.BaseTab` property), 76

`parse_hist_kwargs()` (`gwsomm.plot.builtin.TimeSeriesHistogram2dDataPlot` method), 39

`parse_legend_kwargs()` (`gwsomm.plot.core.DataPlot` method), 43

`parse_list()` (`gwsomm.plot.core.DataPlot` method), 43

`parse_math_definition()` (in module `gwsomm.data.mathutils`), 20

`parse_pcmesh_kwargs()` (`gwsomm.plot.builtin.TimeSeriesHistogram2dDataPlot` method), 40

`parse_plot_kwargs()` (`gwsomm.plot.builtin.TimeSeriesHistogramPlot` method), 40

`parse_plot_kwargs()` (`gwsomm.plot.core.DataPlot` method), 43

`parse_plot_kwargs()` (`gwsomm.plot.range.SimpleTimeVolumeDataPlot` method), 50

`parse_plot_kwargs()` (`gwsomm.plot.segments.DutyDataPlot` method), 51

`parse_plot_kwargs()` (`gwsomm.plot.segments.SegmentDataPlot` method), 54

`parse_plot_kwargs()` (`gwsomm.plot.segments.SegmentHistogramPlot` method), 55

`parse_plot_kwargs()` (`gwsomm.plot.segments.SegmentPiePlot` method), 55

`parse_plot_kwargs()` (`gwsomm.plot.segments.StateVectorDataPlot` method), 56

`parse_rcParams()` (`gwsomm.plot.core.DataPlot` method), 43

`parse_references()` (`gwsomm.plot.builtin.SpectrumDataPlot` method), 38

`parse_residual_params()` (`gwsomm.plot.noisebudget.NoiseBudgetPlot` method), 45

`parse_sum_params()` (`gwsomm.plot.noisebudget.NoiseBudgetPlot` method), 45

`parse_variance_kwargs()` (`gwsomm.plot.builtin.SpectralVarianceDataPlot` method), 37

`parse_wedge_kwargs()` (`gwsomm.plot.segments.SegmentPiePlot` method), 55

`PARSER` (`gwsomm.tests.test_config.TestGWSummConfigParser` attribute), 92

`pid` (`gwsomm.plot.builtin.SpectrogramDataPlot` property), 37

`pid` (`gwsomm.plot.core.DataPlot` property), 43

`pid` (`gwsomm.plot.range.SimpleTimeVolumeDataPlot` property), 50

`pid` (`gwsomm.plot.segments.DutyDataPlot` property), 51

`pid` (`gwsomm.plot.segments.ODCDataPlot` property), 53

`pid` (`gwsomm.plot.segments.SegmentDataPlot` property), 54

`pid` (`gwsomm.plot.segments.StateVectorDataPlot` property), 56

`pid` (`gwsomm.plot.triggers.core.TriggerDataPlot` property), 33

`pid` (`gwsomm.plot.triggers.core.TriggerHistogramPlot` property), 33

`pid` (`gwsomm.plot.triggers.core.TriggerPlotMixin` property), 33

`pid` (`gwsomm.plot.triggers.core.TriggerRateDataPlot` property), 34

`plot()` (`gwsomm.tests.test_plot.TestSummaryPlot` class method), 95

`PlotTab` (class in `gwsomm.tabs.builtin`), 67

`print_segments()` (`gwsomm.tabs.data.DataTab` static method), 80

`process()` (`gwsomm.plot.core.DataPlot` method), 43

`process()` (`gwsomm.tabs.data.DataTab` method), 80

`process()` (`gwsomm.tabs.data.ProcessedTab` method), 82

`process()` (`gwsomm.tabs.etg.EventTriggerTab` method), 84

`process()` (`gwsomm.tabs.fscan.FscanTab` method), 85

`process()` (`gwsomm.tabs.gracedb.GraceDbTab` method), 86

`process()` (`gwsomm.tabs.guardian.GuardianTab` method), 87

`process()` (`gwsomm.tabs.management.AccountingTab` method), 87

`process()` (`gwsomm.tabs.sei.SEIWatchDogTab` method), 90

`process()` (`gwsomm.tabs.stamp.StampPEMTab` method), 91

`process_state()` (`gwsomm.tabs.data.DataTab` method), 81

`process_state()` (`gwsomm.tabs.etg.EventTriggerTab` method), 84

`process_state()` (`gwsomm.tabs.gracedb.GraceDbTab` method), 86

`ProcessedTab` (class in `gwsomm.tabs.data`), 82

## R

RangeCumulativeHistogramPlot (class in gw-summ.plot.range), 46  
RangeCumulativeSpectrumDataPlot (class in gw-summ.plot.range), 46  
RangeDataHistogramPlot (class in gw-summ.plot.range), 47  
RangeDataPlot (class in gwsumm.plot.range), 47  
RangePlotMixin (class in gwsumm.plot.range), 48  
RangeSpectrogramDataPlot (class in gw-summ.plot.range), 48  
RangeSpectrumDataPlot (class in gw-summ.plot.range), 48  
RayleighSpectrogramDataPlot (class in gw-summ.plot.builtin), 36  
RayleighSpectrumDataPlot (class in gw-summ.plot.builtin), 36  
read() (gwsumm.config.GWSummConfigParser method), 18  
read\_cache() (in module gwsumm.triggers), 105  
read\_data\_archive() (in module gwsumm.archive), 98  
read\_frequencyseries() (in module gwsumm.io), 102  
register\_plot() (in module gwsumm.plot.registry), 50  
register\_state() (in module gwsumm.state), 65  
register\_state() (in module gwsumm.state.registry), 61  
register\_tab() (in module gwsumm.tabs.registry), 90  
RelativeNoiseBudgetPlot (class in gw-summ.plot.noisebudget), 45  
resample\_timeseries\_dict() (in module gw-summ.data.timeseries), 25

## S

safe\_eval() (in module gwsumm.utils), 107  
scaffold\_plots() (gwsumm.tabs.builtin.PlotTab method), 69  
SCALE\_UNIT (gwsumm.plot.segments.SegmentBarPlot attribute), 53  
scheme (gwsumm.data.utils.FftParams attribute), 26  
SegmentBarPlot (class in gwsumm.plot.segments), 53  
SegmentDataPlot (class in gwsumm.plot.segments), 53  
SegmentHistogramPlot (class in gw-summ.plot.segments), 55  
SegmentLabelSvgMixin (class in gwsumm.plot.mixins), 44  
SegmentPiePlot (class in gwsumm.plot.segments), 55  
segments\_from\_array() (in module gwsumm.archive), 98  
segments\_to\_array() (in module gwsumm.archive), 98  
SeiWatchDogPlot (class in gwsumm.plot.sei), 57  
SEIWatchDogTab (class in gwsumm.tabs.sei), 90  
set\_command() (gwsumm.batch.GWSummJob method), 100

set\_date\_options() (gw-summ.config.GWSummConfigParser method), 18  
set\_ifo\_options() (gw-summ.config.GWSummConfigParser method), 19  
set\_layout() (gwsumm.tabs.builtin.PlotTab method), 69  
set\_mode() (in module gwsumm.mode), 103  
set\_parent() (gwsumm.tabs.core.BaseTab method), 76  
setup\_class() (gwsumm.tests.test\_data.TestData class method), 93  
setup\_class() (gwsumm.tests.test\_plot.TestSummaryPlot class method), 95  
setup\_class() (gwsumm.tests.test\_tabs.TestTab class method), 96  
shortname (gwsumm.tabs.core.BaseTab property), 76  
shorttitle (gwsumm.tabs.core.BaseTab property), 77  
sieve\_cache() (in module gwsumm.data.timeseries), 25  
SimpleTimeVolumeDataPlot (class in gw-summ.plot.range), 49  
size\_for\_spectrogram() (in module gw-summ.data.spectral), 22  
sort() (gwsumm.tabs.core.TabList method), 78  
span (gwsumm.plot.core.DataPlot property), 43  
SpectralVarianceDataPlot (class in gw-summ.plot.builtin), 36  
SpectrogramDataPlot (class in gwsumm.plot.builtin), 37  
SpectrumDataPlot (class in gwsumm.plot.builtin), 38  
split() (in module gwsumm.channels), 100  
split\_combination() (in module gwsumm.channels), 101  
split\_compound\_flag() (in module gw-summ.segments), 104  
src (gwsumm.plot.core.SummaryPlot property), 44  
StampPEMTab (class in gwsumm.tabs.stamp), 91  
start (gwsumm.plot.core.DataPlot property), 43  
start (gwsumm.state.core.SummaryState property), 60  
start (gwsumm.state.SummaryState property), 64  
state (gwsumm.plot.core.DataPlot property), 43  
state\_switcher() (in module gw-summ.html.bootstrap), 29  
states (gwsumm.tabs.builtin.StateTab property), 72  
StateTab (class in gwsumm.tabs.builtin), 70  
StateVectorDataPlot (class in gw-summ.plot.segments), 55  
static (gwsumm.mode.Mode attribute), 102  
stride (gwsumm.data.utils.FftParams attribute), 26  
SummaryPlot (class in gwsumm.plot.core), 43  
SummaryState (class in gwsumm.state), 62  
SummaryState (class in gwsumm.state.core), 59  
SvgMixin (class in gwsumm.plot.mixins), 44



## T

- Tab (class in `gwsomm.tabs.core`), 77
- TabList (class in `gwsomm.tabs.core`), 78
- tag (`gwsomm.plot.core.DataPlot` property), 43
- tag (`gwsomm.state.core.SummaryState` property), 60
- tag (`gwsomm.state.SummaryState` property), 64
- teardown\_class() (`gwsomm.tests.test_data.TestData` class method), 93
- teardown\_module() (in module `gwsomm.tests.test_channels`), 92
- teardown\_module() (in module `gwsomm.tests.test_mode`), 94
- test\_add\_channel() (`gwsomm.tests.test_plot.TestDataPlot` method), 94
- test\_add\_plot() (`gwsomm.tests.test_tabs.TestPlotTab` method), 96
- test\_add\_timeseries() (`gwsomm.tests.test_data.TestData` method), 93
- test\_allchannels() (`gwsomm.tests.test_plot.TestDataPlot` method), 94
- test\_apply\_parameters() (`gwsomm.tests.test_plot.TestDataPlot` method), 94
- test\_archive\_load\_table() (in module `gwsomm.tests.test_archive`), 91
- test\_banner() (in module `gwsomm.html.tests.test_bootstrap`), 27
- test\_base\_map\_dropdown() (in module `gwsomm.html.tests.test_bootstrap`), 27
- test\_calendar() (in module `gwsomm.html.tests.test_bootstrap`), 27
- test\_calendar\_no\_mode() (in module `gwsomm.html.tests.test_bootstrap`), 27
- test\_channels() (`gwsomm.tests.test_plot.TestDataPlot` method), 94
- test\_comments\_box() (in module `gwsomm.html.tests.test_html5`), 27
- test\_configdir() (`gwsomm.tests.test_config.TestGWSummConfigParser` method), 92
- test\_dialog\_box() (in module `gwsomm.html.tests.test_html5`), 27
- test\_elapsed\_time() (in module `gwsomm.tests.test_utils`), 96
- test\_end() (`gwsomm.tests.test_plot.TestDataPlot` method), 94
- test\_eq() (`gwsomm.tests.test_plot.TestSummaryPlot` method), 95
- test\_expand\_path() (in module `gwsomm.html.tests.test_html5`), 27
- test\_finalize() (`gwsomm.tests.test_config.TestGWSummConfigParser` method), 92
- test\_finalize() (`gwsomm.tests.test_plot.TestDataPlot` method), 94
- test\_find\_frame\_type() (`gwsomm.tests.test_data.TestData` method), 93
- test\_from\_configparser() (`gwsomm.tests.test_config.TestGWSummConfigParser` method), 92
- test\_from\_ini() (`gwsomm.tests.test_plot.TestDataPlot` method), 94
- test\_get\_base() (in module `gwsomm.tests.test_mode`), 94
- test\_get\_channel() (in module `gwsomm.tests.test_channels`), 92
- test\_get\_channel\_groups() (`gwsomm.tests.test_plot.TestDataPlot` method), 94
- test\_get\_channel\_trend() (in module `gwsomm.tests.test_channels`), 92
- test\_get\_channel\_type() (`gwsomm.tests.test_data.TestData` method), 93
- test\_get\_channels() (in module `gwsomm.tests.test_channels`), 92
- test\_get\_coherence\_spectrogram() (`gwsomm.tests.test_data.TestData` method), 93
- test\_get\_coherence\_spectrum() (`gwsomm.tests.test_data.TestData` method), 93
- test\_get\_column\_label() (in module `gwsomm.tests.test_plot`), 95
- test\_get\_css() (`gwsomm.tests.test_config.TestGWSummConfigParser` method), 93
- test\_get\_css() (in module `gwsomm.html.tests.test_static`), 28
- test\_get\_default\_ifo() (in module `gwsomm.tests.test_utils`), 96
- test\_get\_fftparams() (`gwsomm.tests.test_data.TestData` method), 93
- test\_get\_javascript() (`gwsomm.tests.test_config.TestGWSummConfigParser` method), 93
- test\_get\_js() (in module `gwsomm.html.tests.test_static`), 28
- test\_get\_mode() (in module `gwsomm.tests.test_mode`), 94
- test\_get\_odc\_bitmask() (in module `gwsomm.tests.test_utils`), 96
- test\_get\_plot() (in module `gwsomm.tests.test_plot`), 95
- test\_get\_spectrogram() (`gwsomm.tests.test_data.TestData` method), 93
- test\_get\_spectrum() (`gwsomm.tests.test_data.TestData` method), 93
- test\_get\_tab() (in module `gwsomm.tests.test_tabs`), 96

<code>test_get_timeseries()</code> (gw-summ.tests.test_data.TestData method), 94	<code>test_main()</code> (in module gw-summ.plot.triggers.tests.test_main), 32
<code>test_href()</code> (gwsumm.tests.test_plot.TestDataPlot method), 94	<code>test_main()</code> (in module gwsumm.tests.test_batch), 92
<code>test_href()</code> (gwsumm.tests.test_plot.TestSummaryPlot method), 95	<code>test_main_invalid_columns()</code> (in module gw-summ.plot.triggers.tests.test_main), 32
<code>test_ifos()</code> (gwsumm.tests.test_plot.TestDataPlot method), 94	<code>test_main_invalid_modes()</code> (in module gw-summ.tests.test_batch), 92
<code>test_index()</code> (gwsumm.tests.test_tabs.TestTab method), 96	<code>test_main_loop_over_modes()</code> (in module gw-summ.tests.test_batch), 92
<code>test_init()</code> (gwsumm.tests.test_config.TestGWSummConfigParser method), 93	<code>test_main_with_cache_and_tiles()</code> (in module gw-summ.plot.triggers.tests.test_main), 32
<code>test_init()</code> (gwsumm.tests.test_plot.TestDataPlot method), 94	<code>test_make_globalv_key()</code> (gw-summ.tests.test_data.TestData method), 94
<code>test_init()</code> (gwsumm.tests.test_plot.TestSummaryPlot method), 95	<code>test_mkdir()</code> (in module gwsumm.tests.test_utils), 96
<code>test_init()</code> (gwsumm.tests.test_tabs.TestExternalTab method), 96	<code>test_nat_sorted()</code> (in module gw-summ.tests.test_utils), 96
<code>test_init()</code> (gwsumm.tests.test_tabs.TestPlotTab method), 96	<code>test_nditems()</code> (gw-summ.tests.test_config.TestGWSummConfigParser method), 93
<code>test_init()</code> (gwsumm.tests.test_tabs.TestTab method), 96	<code>test_ndoptions()</code> (gw-summ.tests.test_config.TestGWSummConfigParser method), 93
<code>test_interpolate_section_names()</code> (gw-summ.tests.test_config.TestGWSummConfigParser method), 93	<code>test_new()</code> (gwsumm.tests.test_plot.TestSummaryPlot method), 95
<code>test_layout()</code> (gwsumm.tests.test_tabs.TestPlotTab method), 96	<code>test_outputfile()</code> (gw-summ.tests.test_plot.TestDataPlot method), 94
<code>test_ldvw_qscan_batch()</code> (in module gw-summ.html.tests.test_html5), 27	<code>test_overlay_canvas()</code> (in module gw-summ.html.tests.test_html5), 27
<code>test_ldvw_qscan_single()</code> (in module gw-summ.html.tests.test_html5), 27	<code>test_parse_legend_kwargs()</code> (gw-summ.tests.test_plot.TestDataPlot method), 95
<code>test_load()</code> (in module gwsumm.html.tests.test_html5), 27	<code>test_parse_math_definition()</code> (gw-summ.tests.test_data.TestData method), 94
<code>test_load_channels()</code> (gw-summ.tests.test_config.TestGWSummConfigParser method), 93	<code>test_parse_plot_kwargs()</code> (gw-summ.tests.test_plot.TestDataPlot method), 95
<code>test_load_custom()</code> (in module gw-summ.html.tests.test_html5), 27	<code>test_parse_plot_kwargs_labels()</code> (gw-summ.tests.test_plot.TestDataPlot method), 95
<code>test_load_plugins()</code> (gw-summ.tests.test_config.TestGWSummConfigParser method), 93	<code>test_parse_rcParams()</code> (gw-summ.tests.test_plot.TestDataPlot method), 95
<code>test_load_rcParams()</code> (gw-summ.tests.test_config.TestGWSummConfigParser method), 93	<code>test_read()</code> (gwsumm.tests.test_config.TestGWSummConfigParser method), 93
<code>test_load_state()</code> (in module gw-summ.html.tests.test_html5), 27	<code>test_read_archive()</code> (in module gw-summ.tests.test_archive), 91
<code>test_load_states()</code> (gw-summ.tests.test_config.TestGWSummConfigParser method), 93	<code>test_register_tab()</code> (in module gw-summ.tests.test_tabs), 96
<code>test_load_units()</code> (gw-summ.tests.test_config.TestGWSummConfigParser method), 93	<code>test_registry_plot()</code> (in module gw-summ.tests.test_plot), 95
<code>test_main()</code> (in module gw-summ.plot.guardian.tests.test_main), 31	<code>test_repr()</code> (gwsumm.tests.test_plot.TestSummaryPlot method), 95
	<code>test_safe_eval()</code> (in module gwsumm.tests.test_utils),

96				32			
test_safe_eval_2()	(in module gw-	TriggerHistogramPlot	(class in gw-				
	summ.tests.test_utils), 97		summ.plot.triggers.core), 33				
test_set_date_options()	(gw-	TriggerPlotMixin	(class in gw-				
	summ.tests.test_config.TestGWSummConfigParser		summ.plot.triggers.core), 33				
	method), 93	TriggerRateDataPlot	(class in gw-				
test_set_ifo_options()	(gw-		summ.plot.triggers.core), 33				
	summ.tests.test_config.TestGWSummConfigParser	TriggerTimeSeriesDataPlot	(class in gw-				
	method), 93		summ.plot.triggers.core), 34				
test_set_mode()	(in module gwsumm.tests.test_mode),	type (gwsumm.plot.builtin.CoherenceSpectrogramDataPlot					
	94	attribute), 35					
test_shortcode()	(gwsumm.tests.test_tabs.TestTab	type (gwsumm.plot.builtin.CoherenceSpectrumDataPlot					
	method), 96	attribute), 36					
test_span()	(gwsumm.tests.test_plot.TestDataPlot	type (gwsumm.plot.builtin.RayleighSpectrogramDataPlot					
	method), 95	attribute), 36					
test_split()	(in module gwsumm.tests.test_channels),	type (gwsumm.plot.builtin.RayleighSpectrumDataPlot					
	92	attribute), 36					
test_split_combination()	(in module gw-	type (gwsumm.plot.builtin.SpectralVarianceDataPlot at-					
	summ.tests.test_channels), 92	tribute), 37					
test_src()	(gwsumm.tests.test_plot.TestSummaryPlot	type (gwsumm.plot.builtin.SpectrogramDataPlot at-					
	method), 95	tribute), 37					
test_start()	(gwsumm.tests.test_plot.TestDataPlot	type (gwsumm.plot.builtin.SpectrumDataPlot attribute),					
	method), 95	38					
test_state()	(gwsumm.tests.test_plot.TestDataPlot	type (gwsumm.plot.builtin.TimeSeriesDataPlot at-					
	method), 95	tribute), 39					
test_state_switcher()	(in module gw-	type (gwsumm.plot.builtin.TimeSeriesHistogram2dDataPlot					
	summ.html.tests.test_bootstrap), 27	attribute), 40					
test_str()	(gwsumm.tests.test_plot.TestSummaryPlot	type (gwsumm.plot.builtin.TimeSeriesHistogramPlot at-					
	method), 95	tribute), 40					
test_tag()	(gwsumm.tests.test_plot.TestDataPlot	type (gwsumm.plot.core.DataPlot attribute), 43					
	method), 95	type (gwsumm.plot.core.SummaryPlot attribute), 44					
test_update_missing_channel_params()	(in mod-	type (gwsumm.plot.guardian.core.GuardianStatePlot at-					
	ule gwsumm.tests.test_channels), 92	tribute), 31					
test_vprint()	(in module gwsumm.tests.test_utils), 97	type (gwsumm.plot.noisebudget.NoiseBudgetPlot at-					
test_wrap_content()	(in module gw-	tribute), 45					
	summ.html.tests.test_bootstrap), 27	type (gwsumm.plot.noisebudget.RelativeNoiseBudgetPlot					
test_write_archive()	(in module gw-	attribute), 46					
	summ.tests.test_archive), 92	type (gwsumm.plot.range.GWpyTimeVolumeDataPlot at-					
TestData	(class in gwsumm.tests.test_data), 93	tribute), 46					
TestDataPlot	(class in gwsumm.tests.test_plot), 94	type (gwsumm.plot.range.RangeCumulativeHistogramPlot					
TestExternalTab	(class in gwsumm.tests.test_tabs), 96	attribute), 46					
TestGWSummConfigParser	(class in gw-	type (gwsumm.plot.range.RangeCumulativeSpectrumDataPlot					
	summ.tests.test_config), 92	attribute), 47					
TestPlotTab	(class in gwsumm.tests.test_tabs), 96	type (gwsumm.plot.range.RangeDataHistogramPlot at-					
TestSummaryPlot	(class in gwsumm.tests.test_plot), 95	tribute), 47					
TestTab	(class in gwsumm.tests.test_tabs), 96	type (gwsumm.plot.range.RangeDataPlot attribute), 48					
TimeSeriesDataPlot	(class in gwsumm.plot.builtin), 38	type (gwsumm.plot.range.RangeSpectrogramDataPlot					
TimeSeriesHistogram2dDataPlot	(class in gw-	attribute), 48					
	summ.plot.builtin), 39	type (gwsumm.plot.range.RangeSpectrumDataPlot at-					
TimeSeriesHistogramPlot	(class in gw-	tribute), 49					
	summ.plot.builtin), 40	type (gwsumm.plot.range.SimpleTimeVolumeDataPlot					
tint_hex()	(in module gwsumm.plot.segments), 56	attribute), 50					
title	(gwsumm.tabs.core.BaseTab property), 77	type (gwsumm.plot.segments.DutyDataPlot attribute), 51					
TriggerDataPlot	(class in gwsumm.plot.triggers.core),	type (gwsumm.plot.segments.NetworkDutyBarPlot					

attribute), 51  
 type (gwsomm.plot.segments.NetworkDutyPiePlot attribute), 52  
 type (gwsomm.plot.segments.ODCDataPlot attribute), 53  
 type (gwsomm.plot.segments.SegmentBarPlot attribute), 53  
 type (gwsomm.plot.segments.SegmentDataPlot attribute), 54  
 type (gwsomm.plot.segments.SegmentHistogramPlot attribute), 55  
 type (gwsomm.plot.segments.SegmentPiePlot attribute), 55  
 type (gwsomm.plot.segments.StateVectorDataPlot attribute), 56  
 type (gwsomm.plot.sei.SeiWatchDogPlot attribute), 57  
 type (gwsomm.plot.triggers.core.TriggerDataPlot attribute), 33  
 type (gwsomm.plot.triggers.core.TriggerHistogramPlot attribute), 33  
 type (gwsomm.plot.triggers.core.TriggerRateDataPlot attribute), 34  
 type (gwsomm.plot.triggers.core.TriggerTimeSeriesDataPlot attribute), 34  
 type (gwsomm.tabs.builtin.ExternalTab attribute), 67  
 type (gwsomm.tabs.builtin.PlotTab attribute), 69  
 type (gwsomm.tabs.builtin.StateTab attribute), 72  
 type (gwsomm.tabs.core.Tab attribute), 78  
 type (gwsomm.tabs.data.DataTab attribute), 81  
 type (gwsomm.tabs.data.ProcessedTab attribute), 82  
 type (gwsomm.tabs.etg.EventTriggerTab attribute), 85  
 type (gwsomm.tabs.fscan.FscanTab attribute), 85  
 type (gwsomm.tabs.gracedb.GraceDbTab attribute), 86  
 type (gwsomm.tabs.guardian.GuardianTab attribute), 87  
 type (gwsomm.tabs.management.AccountingTab attribute), 88  
 type (gwsomm.tabs.misc.AboutTab attribute), 88  
 type (gwsomm.tabs.misc.Error404Tab attribute), 89  
 type (gwsomm.tabs.sei.SEIWatchDogTab attribute), 90  
 type (gwsomm.tabs.stamp.StampPEMTab attribute), 91  
 TYPE (gwsomm.tests.test\_plot.TestDataPlot attribute), 94  
 TYPE (gwsomm.tests.test\_plot.TestSummaryPlot attribute), 95  
 TYPE (gwsomm.tests.test\_tabs.TestExternalTab attribute), 96  
 TYPE (gwsomm.tests.test\_tabs.TestPlotTab attribute), 96  
 TYPE (gwsomm.tests.test\_tabs.TestTab attribute), 96

## U

undo\_demodulation() (in module gwsomm.plot.builtin), 40  
 update\_channel\_params() (in module gwsomm.channels), 101

update\_missing\_channel\_params() (in module gwsomm.channels), 101  
 url (gwsomm.tabs.builtin.ExternalTab property), 67  
 use\_configparser() (in module gwsomm.data.utils), 26  
 use\_segmentlist() (in module gwsomm.data.utils), 26

## V

vprint() (in module gwsomm.utils), 107

## W

week (gwsomm.mode.Mode attribute), 102  
 window (gwsomm.data.utils.FftParams attribute), 26  
 window (gwsomm.tabs.sei.SEIWatchDogTab attribute), 90  
 wrap\_content() (in module gwsomm.html.bootstrap), 29  
 write\_data\_archive() (in module gwsomm.archive), 99  
 write\_html() (gwsomm.tabs.builtin.ExternalTab method), 67  
 write\_html() (gwsomm.tabs.builtin.PlotTab method), 69  
 write\_html() (gwsomm.tabs.builtin.StateTab method), 72  
 write\_html() (gwsomm.tabs.core.BaseTab method), 77  
 write\_html() (gwsomm.tabs.data.DataTab method), 81  
 write\_html() (gwsomm.tabs.misc.AboutTab method), 88  
 write\_html() (gwsomm.tabs.misc.Error404Tab method), 89  
 write\_state\_html() (gwsomm.tabs.builtin.StateTab method), 73  
 write\_state\_html() (gwsomm.tabs.data.DataTab method), 82  
 write\_state\_html() (gwsomm.tabs.etg.EventTriggerTab method), 85  
 write\_state\_html() (gwsomm.tabs.fscan.FscanTab method), 85  
 write\_state\_html() (gwsomm.tabs.gracedb.GraceDbTab method), 86  
 write\_state\_html() (gwsomm.tabs.guardian.GuardianTab method), 87  
 write\_state\_html() (gwsomm.tabs.management.AccountingTab method), 88  
 write\_state\_html() (gwsomm.tabs.sei.SEIWatchDogTab method), 90  
 write\_state\_html() (gwsomm.tabs.stamp.StampPEMTab method),



91

`write_state_information()` (gw-  
*summ.tabs.data.DataTab method*), 82

`write_state_placeholder()` (gw-  
*summ.tabs.data.DataTab method*), 82

`write_sub_file()` (gwsumm.batch.GWSummaryJob  
*method*), 100

## Y

`year` (gwsumm.mode.Mode attribute), 102