

---

**tom\_pittgoogle**

*Release v0.1.0*

**Troy Raen**

**Oct 08, 2021**



## OVERVIEW

<b>1 Basic Overview</b>	<b>3</b>
<b>Python Module Index</b>	<b>15</b>
<b>Index</b>	<b>17</b>



This repo contains 3 proof-of-concept implementations of a TOM Toolkit `GenericBroker` class which fetch alerts from Pitt-Google.

Contact Troy Raen with questions or for authentication access (Slack @troyraen, or [troy.raen@pitt.edu](mailto:troy.raen@pitt.edu)).



## BASIC OVERVIEW

Table 1: TOM Broker 3 ways

Implementation	Connects to	Via	Comments
<i>StreamRest</i>	Pub/Sub streams	REST API	Closest to “standard” implementation using HTTP requests. Uses batch-style message pulls.
<i>StreamPython</i>	Pub/Sub streams	Python client	<b>Recommended</b> for listening to a full night’s stream. Uses a streaming pull in a background thread.
<i>DatabasePython</i>	BigQuery database	Python client	

Each implementation relies on 2 classes, a *Broker* and a *Consumer*:

Table 2: 2 classes for each implementation

<i>Broker</i>	<i>Consumer</i>
<ul style="list-style-type: none"><li>Fetches alerts from Pitt-Google using a <i>Consumer</i></li></ul>	<ul style="list-style-type: none"><li>Handles the stream/database connections and unpacks the returned data.</li></ul>
<ul style="list-style-type: none"><li>Base class: <code>tom_alerts.alerts.GenericBroker</code></li></ul>	<ul style="list-style-type: none"><li>Python methods use Google’s client APIs (<a href="#">Pub/Sub</a>, <a href="#">BigQuery</a>)</li><li>REST method uses a <code>requests_oauthlib.OAuth2Session</code> object for HTTP requests</li></ul>

Here we use *Broker* and *Consumer* generically to refer to any of the specific implementations, which have names like `BrokerStreamRest`.

### 1.1 Basic Code Workflow

Each implementation does things a bit differently, but they share a basic workflow:

The *Broker* instantiates a *Consumer* and uses it to fetch, unpack, and process alerts.

The *Consumer* can accept a user filter and return only alerts that pass.

Here is a compact but working example of a *Broker*’s `fetch_alerts` method for the *StreamRest* implementation.

```
def fetch_alerts(self):
    from consumer_stream_rest import ConsumerStreamRest

    subscription_name = "ztf-loop"
    max_messages = 10
    lighten_alerts = True # flatten the alert dict and drop extra fields. optional.
    # If you pass a callback function, the Consumer will run each alert through it.
    # Optional. Useful for user filters. Here's a basic demo.
    def user_filter(alert_dict):
        passes_filter = True
        if passes_filter:
            return alert_dict
        else:
            return None
    callback = user_filter

    consumer = ConsumerStreamRest(subscription_name)

    response = consumer.oauth2.post(
        f"{consumer.subscription_url}:pull", data={"maxMessages": max_messages},
    )

    alerts = consumer.unpack_and_ack_messages(
        response, lighten_alerts=lighten_alerts, callback=callback,
    ) # List[dict]

    return iter(alerts)
```

## 1.2 How to integrate with TOM Toolkit

This assumes that you know how to run a TOM server/site using the [TOM Toolkit](#).

1. Clone this repo and put the directory on your path. (`git clone https://github.com/mwvgroup/tom_pittgoogle.git`)
2. Add Pitt-Google to your TOM. Follow the TOM Toolkit instructions in the section [Using Our New Alert Broker](#). Our modules were written following the instructions preceding that section.

- In your `settings.py` file:
  - Add these to the `TOM_ALERT_CLASSES` list:

```
'tom_pittgoogle.broker_stream_rest.BrokerStreamRest',
'tom_pittgoogle.broker_stream_python.BrokerStreamPython',
'tom_pittgoogle.broker_database_python.BrokerDatabasePython',
```

- Add these additional settings:

```
# see the Authentication docs for more info
GOOGLE_CLOUD_PROJECT = "pitt-broker-user-project" # user's project
PITTGOOGLE_OAUTH_CLIENT_ID = os.getenv("PITTGOOGLE_OAUTH_CLIENT_ID")
PITTGOOGLE_OAUTH_CLIENT_SECRET = os.getenv("PITTGOOGLE_OAUTH_CLIENT_SECRET")
```



3. After running `makemigrations`, etc. and authenticating yourself, navigate to the “Alerts” page on your TOM site. You should see three new broker options corresponding to the three Pitt-Google classes you added to the `TOM_ALERT_CLASSES` list.

## 1.3 Authentication

Users authenticate themselves by following an OAuth 2.0 workflow. Authentication is required to make API calls.

- *Requirements*
- *Authentication Workflow*

### 1.3.1 Requirements

1. The user must have a Google account (e.g., Gmail address) that is authorized make API calls through the project that is defined by the `GOOGLE_CLOUD_PROJECT` variable in the Django `settings.py` file. Any project can be used, as long as the user is authorized.
  - We have a test project setup that we are happy to add community members to, for as long as that remains feasible. Send Troy a request, and include your Google account info (Gmail address).
2. Since this is still in dev: Contact Troy to be added to the OAuth’s list of authorized test users, and to obtain the `PITGOOGLE_OAUTH_CLIENT_ID` and `PITGOOGLE_OAUTH_CLIENT_SECRET`. Include your Google account info (Gmail address).

### 1.3.2 Authentication Workflow

Note: Currently this is a bit painful because the user must:

- re-authenticate every time a query is run.
- interact via the command line. When running a query from the TOM site’s “Query a Broker” page, **the process will hang until the user follows the prompts on the command line and completes the authentication.** The site may temporarily crash until this is completed.

(TODO: integrate the OAuth with Django, and automatically refresh tokens)

**Workflow** - The user will:

1. Visit a URL, which will be displayed on the command line when the *Consumer* class is initialized (currently, when the *Broker*’s `fetch_alerts` is called).
2. Log in to their Google account. This authenticates their access to make API calls through the project.
3. Authorize this *PittGoogleConsumer* app/module to make API calls on their behalf. This only needs to be done once for each API access “scope” (Pub/Sub, BigQuery, and Logging).
4. Respond to the prompt on the command line by entering the full URL of the webpage they are redirected to after completing the above.

**What happens next?** - The *Consumer*:

1. Completes the instantiation of an `OAuth2Session`, which is used to either make HTTP requests directly, or instantiate a credentials object for the Python client.
2. Instantiates a `Client` object to make API calls with (Python methods only).

3. Checks that it can successfully connect to the requested resource.

## 1.4 StreamRest

- *BrokerStreamRest*
- *ConsumerStreamRest*

### 1.4.1 BrokerStreamRest

TOM Toolkit broker to listen to a Pitt-Google Pub/Sub stream via the REST API.

Relies on *ConsumerStreamRest* to manage the connections and work with data.

See especially:

<i>BrokerStreamRest.request_alerts</i>	Pull alerts using a POST request with OAuth2, unpack, apply user filter.
<i>BrokerStreamRest.user_filter</i>	Apply the filter indicated by the form's parameters.

**class** tom\_pittgoogle.broker\_stream\_rest.**BrokerStreamRest**

Pitt-Google broker class to pull alerts from a stream via the REST API.

Base class: tom\_alerts.alerts.GenericBroker

**fetch\_alerts**(parameters)

Entry point to pull and filter alerts.

**form**

alias of *tom\_pittgoogle.broker\_stream\_rest.FilterAlertsForm*

**request\_alerts**(parameters)

Pull alerts using a POST request with OAuth2, unpack, apply user filter.

**Returns** alerts (List[dict])

**to\_generic\_alert**(alert)

Map the Pitt-Google alert to a TOM *GenericAlert*.

**static user\_filter**(alert\_dict, parameters)

Apply the filter indicated by the form's parameters.

Used as the *callback* to *BrokerStreamRest.unpack\_and\_ack\_messages*.

**Parameters**

- **alert\_dict** – Single alert, ZTF packet data as a dictionary. The schema depends on the value of *lighten\_alerts* passed to *BrokerStreamRest.unpack\_and\_ack\_messages*. If *lighten\_alerts=False* it is the original ZTF alert schema (<https://zwickytransientfacility.github.io/ztf-avro-alert/schema.html>). If *lighten\_alerts=True* the dict is flattened and extra fields are dropped.
- **parameters** – parameters submitted by the user through the form.

**Returns** *alert\_dict* if it passes the filter, else *None*

```
class tom_pittgoogle.broker_stream_rest.FilterAlertsForm(*args, **kwargs)
```

Basic form for filtering alerts.

Fields:

subscription\_name (CharField)

classtar\_threshold (FloatField)

classtar\_gt\_lt (ChoiceField)

max\_results (IntegerField)

**property** media

Return all media required to render the widgets on this form.

## 1.4.2 ConsumerStreamRest

Consumer class to manage Pub/Sub connections via REST, and work with message data.

Pub/Sub REST API docs: <https://cloud.google.com/pubsub/docs/reference/rest>

Used by *BrokerStreamRest*, but can be called independently.

Basic workflow:

```
consumer = ConsumerStreamRest(subscription_name)

response = consumer.oauth2.post(
    f"{consumer.subscription_url}:pull", data={"maxMessages": max_messages},
)

alerts = consumer.unpack_and_ack_messages(
    response, lighten_alerts=True, callback=user_filter,
) # List[dict]
```

See especially:

<code>ConsumerStreamRest.authenticate</code>	Guide user through authentication; create <i>OAuth2Session</i> for HTTP requests.
<code>ConsumerStreamRest.get_create_subscription</code>	Make sure the subscription exists and we can connect.
<code>ConsumerStreamRest.unpack_and_ack_messages</code>	Unpack and acknowledge messages in <i>response</i> ; run <i>callback</i> if present.

```
class tom_pittgoogle.consumer_stream_rest.ConsumerStreamRest(subscription_name)
```

Consumer class to manage Pub/Sub connections and work with messages.

Initialization does the following:

- Authenticate the user. Create an *OAuth2Session* object for the user/broker to make HTTP requests with.
- Make sure the subscription exists and we can connect. Create it, if needed.

**authenticate()**

Guide user through authentication; create *OAuth2Session* for HTTP requests.

The user will need to visit a URL, authenticate themselves, and authorize *PittGoogleConsumer* to make API calls on their behalf.

The user must have a Google account that is authorized make API calls through the project defined by the

`GOOGLE_CLOUD_PROJECT` variable in the Django `settings.py` file. Any project can be used, as long as the user has access.

Additional requirement because this is still in dev: The OAuth is restricted to users registered with Pitt-Google, so contact us.

TODO: Integrate this with Django. For now, the user interacts via command line.

#### **delete\_subscription()**

Delete the subscription.

This is provided for the user's convenience, but it is not necessary and is not automatically called.

- Storage of unacknowledged Pub/Sub messages does not result in fees.
- Unused subscriptions automatically expire; default is 31 days.

#### **get\_create\_subscription()**

Make sure the subscription exists and we can connect.

If the subscription doesn't exist, try to create one (in the user's project) that is attached to a topic of the same name in the Pitt-Google project.

Note that messages published before the subscription is created are not available.

#### **unpack\_and\_ack\_messages**(*response*, *lighten\_alerts=False*, *callback=None*, *\*\*kwargs*)

Unpack and acknowledge messages in *response*; run *callback* if present.

If *lighten\_alerts* is True, drop extra fields and flatten the alert dict.

*callback* is assumed to be a filter. It should accept an alert dict and return the dict if the alert passes the filter, else return None.

## 1.5 StreamPython

- *BrokerStreamPython*
- *ConsumerStreamPython*

### 1.5.1 BrokerStreamPython

TOM Toolkit broker to listen to a Pitt-Google Pub/Sub stream via the Python client.

Relies on *ConsumerStreamPython* to manage the connections and work with data.

See especially:

<code>BrokerStreamPython.fetch_alerts</code>	Entry point to pull and filter alerts.
<code>BrokerStreamPython.user_filter</code>	Apply the filter indicated by the form's parameters.

#### **class tom\_pittgoogle.broker\_stream\_python.BrokerStreamPython**

Pitt-Google broker interface to pull alerts from Pub/Sub via the Python client.

Base class: `tom_alerts.alerts.GenericBroker`

#### **fetch\_alerts**(*parameters*)

Entry point to pull and filter alerts.

Pull alerts using a Python client, unpack, apply user filter.

This demo assumes that the real use-case is to save alerts to a database rather than view them through a TOM site. Therefore, the *Consumer* currently saves the alerts in real time, and then simply returns a list of alerts after all messages are processed. That list is then coerced into an iterator here. If the user really cares about the iterator, *ConsumerStreamPython.stream\_alerts* can be tweaked to yield the alerts in real time.

**form**

alias of `tom_pittgoogle.broker_stream_python.FilterAlertsForm`

**to\_generic\_alert(alert)**

Map the Pitt-Google alert to a TOM *GenericAlert*.

**static user\_filter(alert\_dict, parameters)**

Apply the filter indicated by the form's parameters.

Used as the *callback* to *BrokerStreamPython.unpack\_and\_ack\_messages*.

**Parameters**

- **alert\_dict** – Single alert, ZTF packet data as a dictionary. The schema depends on the value of *lighten\_alerts* passed to *BrokerStreamPython.unpack\_and\_ack\_messages*. If *lighten\_alerts=False* it is the original ZTF alert schema (<https://zwickytransientfacility.github.io/ztf-avro-alert/schema.html>). If *lighten\_alerts=True* the dict is flattened and extra fields are dropped.
- **parameters** – parameters submitted by the user through the form.

**Returns** *alert\_dict* if it passes the filter, else *None*

**class** tom\_pittgoogle.broker\_stream\_python.**FilterAlertsForm**(\*args, \*\*kwargs)

Basic form for filtering alerts.

Fields:

subscription\_name (CharField)

classtar\_threshold (FloatField)

classtar\_gt\_lt (ChoiceField)

max\_results (IntegerField)

timeout (IntegerField)

max\_backlog (IntegerField)

save\_metadata (ChoiceField)

**property media**

Return all media required to render the widgets on this form.

## 1.5.2 ConsumerStreamPython

Consumer class to manage Pub/Sub connections via a Python client, and work with data.

Pub/Sub Python Client docs: <https://googleapis.dev/python/pubsub/latest/index.html>

Used by *BrokerStreamPython*, but can be called independently.

Use-case: Save alerts to a database

This demo assumes that the real use-case is to save alerts to a database rather than view them through a TOM site. Therefore, the *Consumer* currently saves the alerts in real time, and then simply returns a list of alerts after all messages are processed. That list is then coerced into an iterator by the *Broker*. If the user really cares about the *Broker*'s iterator, *stream\_alerts* can be tweaked to yield the alerts in real time.

Basic workflow:

```
consumer = ConsumerStreamPython(subscription_name)

alert_dicts_list = consumer.stream_alerts(
    lighten_alerts=True,
    user_filter=user_filter,
    parameters=parameters,
)
# alerts are processed and saved in real time. the list is returned for convenience.
```

See especially:

<code>ConsumerStreamPython.authenticate</code>	Guide user through authentication; create <i>OAuth2Session</i> for credentials.
<code>ConsumerStreamPython.get_create_subscription</code>	Make sure the subscription exists and we can connect.
<code>ConsumerStreamPython.stream_alerts</code>	Execute a streaming pull and process alerts through the <i>callback</i> .
<code>ConsumerStreamPython.callback</code>	Process a single alert; run user filter; save alert; acknowledge Pub/Sub msg.
<code>ConsumerStreamPython.save_alert</code>	Save the alert to a database.

**class** tom\_pittgoogle.consumer\_stream\_python.**ConsumerStreamPython**(*subscription\_name*)  
Consumer class to manage Pub/Sub connections and work with messages.

Initialization does the following:

- Authenticate the user via OAuth 2.0.
- Create a *google.cloud.pubsub\_v1.SubscriberClient* object.
- Create a *queue.Queue* object to communicate with the background thread running the streaming pull.
- Make sure the subscription exists and we can connect. Create it, if needed.

To view logs, visit: <https://console.cloud.google.com/logs>

- Make sure you are logged in, and your project is selected in the dropdown at the top.
- Click the “Log name” dropdown and select the subscription name you instantiate this consumer with.

TODO: Give the user a standard logger.

#### **authenticate()**

Guide user through authentication; create *OAuth2Session* for credentials.

The user will need to visit a URL, authenticate themselves, and authorize *PittGoogleConsumer* to make API calls on their behalf.

The user must have a Google account that is authorized make API calls through the project defined by the *GOOGLE\_CLOUD\_PROJECT* variable in the Django *settings.py* file. Any project can be used, as long as the user has access.

Additional requirement because this is still in dev: The OAuth is restricted to users registered with Pitt-Google, so contact us.

TODO: Integrate this with Django. For now, the user interacts via command line.

#### **callback**(*message*)

Process a single alert; run user filter; save alert; acknowledge Pub/Sub msg.

Used as the callback for the streaming pull.

#### **delete\_subscription**()

Delete the subscription.

This is provided for the user's convenience, but it is not necessary and is not automatically called.

- Storage of unacknowledged Pub/Sub messages does not result in fees.
- Unused subscriptions automatically expire; default is 31 days.

#### **get\_create\_subscription**()

Make sure the subscription exists and we can connect.

If the subscription doesn't exist, try to create one (in the user's project) that is attached to a topic of the same name in the Pitt-Google project.

Note that messages published before the subscription is created are not available.

#### **save\_alert**(*alert*)

Save the alert to a database.

#### **stream\_alerts**(*lighten\_alerts=False, user\_filter=None, parameters=None*)

Execute a streaming pull and process alerts through the *callback*.

The streaming pull happens in a background thread. A *queue.Queue* is used to communicate between threads and enforce the stopping condition(s).

#### **Parameters**

- **lighten\_alerts** (*bool*) – If True, drop extra fields and flatten the alert dict
- **user\_filter** (*Callable*) – Used by *callback* to filter alerts before saving. It should accept a single alert (ZTF packet data) as a dictionary. The schema depends on the value of *lighten\_alerts*. If *lighten\_alerts=False* it is the original ZTF alert schema (<https://zwockytransientfacility.github.io/ztf-avro-alert/schema.html>). If *lighten\_alerts=True* the dict is flattened and extra fields are dropped. It should return the alert dict if it passes the filter, else None.
- **parameters** (*dict*) – User's parameters. Must include the parameters defined in the *Broker's FilterAlertsForm*.

## 1.6 DatabasePython

- *BrokerDatabasePython*
- *ConsumerDatabasePython*

### 1.6.1 BrokerDatabasePython

TOM Toolkit broker to query a BigQuery table via the Python API.

Relies on *ConsumerDatabasePython* to manage the connections and work with data.

See especially:

---

<i>BrokerDatabasePython.request_alerts</i>	Query alerts using the user filter and unpack.
--	--

---

**class** tom\_pittgoogle.broker\_database\_python.**BrokerDatabasePython**

Pitt-Google broker to query alerts from the database via the Python client.

Base class: tom\_alerts.alerts.GenericBroker

**fetch\_alerts**(parameters)

Entry point to query and filter alerts.

**form**

alias of *tom\_pittgoogle.broker\_database\_python.FilterAlertsForm*

**request\_alerts**(parameters)

Query alerts using the user filter and unpack.

The SQL statement returned by the *Consumer* implements the current user filter.

**Returns** alerts (List[dict])

**to\_generic\_alert**(alert)

Map the Pitt-Google alert to a TOM *GenericAlert*.

**class** tom\_pittgoogle.broker\_database\_python.**FilterAlertsForm**(\*args, \*\*kwargs)

Basic form for filtering alerts; currently implemented in the SQL statement.

**Fields:** objectId (CharField)

candid (IntegerField)

max\_results (IntegerField)

**property media**

Return all media required to render the widgets on this form.

### 1.6.2 ConsumerDatabasePython

Consumer class to manage BigQuery connections via Python client, and work with data.

BigQuery Python Client docs: <https://googleapis.dev/python/bigquery/latest/index.html>

Used by *BrokerDatabasePython*, but can be called independently.

Basic workflow:

```
consumer = ConsumerDatabasePython(table_name)

sql_stmt, job_config = consumer.create_sql_stmt(parameters)
query_job = consumer.client.query(sql_stmt, job_config=job_config)

alerts = consumer.unpack_query(query_job)  # List[dict]
```



See especially:

<code>ConsumerDatabasePython.authenticate</code>	Guide user through authentication; create <i>OAuth2Session</i> for credentials.
<code>ConsumerDatabasePython.create_sql_stmt</code>	Create the SQL statement and a job config with the user's <i>parameters</i> .
<code>ConsumerDatabasePython.unpack_query</code>	Unpack alerts from <i>query_job</i> ; run <i>callback</i> if present.

**class** tom\_pittgoogle.consumer\_database\_python.**ConsumerDatabasePython**(*table\_name*)

Consumer class to query alerts from BigQuery, and manipulate them.

Initialization does the following:

- Authenticate the user via OAuth 2.0.
- Create a *google.cloud.bigquery.Client* object for the user/broker to query database with.
- Check that the table exists and we can connect.

To view logs, visit: <https://console.cloud.google.com/logs>

- Make sure you are logged in, and your project is selected in the dropdown at the top.
- Click the “Log name” dropdown and select the table name you instantiate this consumer with.

TODO: Give the user a standard logger.

**authenticate()**

Guide user through authentication; create *OAuth2Session* for credentials.

The user will need to visit a URL, authenticate themselves, and authorize *PittGoogleConsumer* to make API calls on their behalf.

The user must have a Google account that is authorized make API calls through the project defined by the *GOOGLE\_CLOUD\_PROJECT* variable in the Django *settings.py* file. Any project can be used, as long as the user has access.

Additional requirement because this is still in dev: The OAuth is restricted to users registered with Pitt-Google, so contact us.

TODO: Integrate this with Django. For now, the user interacts via command line.

**create\_sql\_stmt**(*parameters*)

Create the SQL statement and a job config with the user's *parameters*.

**unpack\_query**(*query\_job*, *callback=None*, *\*\*kwargs*)

Unpack alerts from *query\_job*; run *callback* if present.

A basic filter is implemented directly in the SQL statement produced by *create\_sql\_stmt*. More complex filters could be implemented here via a *callback* function.



## PYTHON MODULE INDEX

### t

`tom_pittgoogle.broker_database_python`, [12](#)  
`tom_pittgoogle.broker_stream_python`, [8](#)  
`tom_pittgoogle.broker_stream_rest`, [6](#)  
`tom_pittgoogle.consumer_database_python`, [12](#)  
`tom_pittgoogle.consumer_stream_python`, [9](#)  
`tom_pittgoogle.consumer_stream_rest`, [7](#)



## INDEX

### A

authenticate() (tom\_pittgoogle.consumer\_database\_python.ConsumerDatabasePython method), 13

authenticate() (tom\_pittgoogle.consumer\_stream\_python.ConsumerStreamPython method), 10

authenticate() (tom\_pittgoogle.consumer\_stream\_rest.ConsumerStreamRest method), 7

fetch\_alerts() (tom\_pittgoogle.broker\_stream\_python.BrokerStreamPython method), 8

fetch\_alerts() (tom\_pittgoogle.broker\_stream\_rest.BrokerStreamRest method), 6

FilterAlertsForm (class in tom\_pittgoogle.broker\_database\_python), 12

### B

BrokerDatabasePython (class in tom\_pittgoogle.broker\_database\_python), 12

BrokerStreamPython (class in tom\_pittgoogle.broker\_stream\_python), 8

BrokerStreamRest (class in tom\_pittgoogle.broker\_stream\_rest), 6

FilterAlertsForm (class in tom\_pittgoogle.broker\_stream\_python), 9

FilterAlertsForm (class in tom\_pittgoogle.broker\_stream\_rest), 6

form (tom\_pittgoogle.broker\_database\_python.BrokerDatabasePython attribute), 12

form (tom\_pittgoogle.broker\_stream\_python.BrokerStreamPython attribute), 9

form (tom\_pittgoogle.broker\_stream\_rest.BrokerStreamRest attribute), 6

### C

callback() (tom\_pittgoogle.consumer\_stream\_python.ConsumerStreamPython method), 11

ConsumerDatabasePython (class in tom\_pittgoogle.consumer\_database\_python), 13

ConsumerStreamPython (class in tom\_pittgoogle.consumer\_stream\_python), 10

ConsumerStreamRest (class in tom\_pittgoogle.consumer\_stream\_rest), 7

create\_sql\_stmt() (tom\_pittgoogle.consumer\_database\_python.ConsumerDatabasePython method), 13

get\_create\_subscription() (tom\_pittgoogle.consumer\_stream\_python.ConsumerStreamPython method), 11

get\_create\_subscription() (tom\_pittgoogle.consumer\_stream\_rest.ConsumerStreamRest method), 8

media (tom\_pittgoogle.broker\_database\_python.FilterAlertsForm property), 12

media (tom\_pittgoogle.broker\_stream\_python.FilterAlertsForm property), 9

media (tom\_pittgoogle.broker\_stream\_rest.FilterAlertsForm property), 7

### D

delete\_subscription() (tom\_pittgoogle.consumer\_stream\_python.ConsumerStreamPython method), 11

delete\_subscription() (tom\_pittgoogle.consumer\_stream\_rest.ConsumerStreamRest method), 8

module tom\_pittgoogle.broker\_database\_python, 12

tom\_pittgoogle.broker\_stream\_python, 8

tom\_pittgoogle.broker\_stream\_rest, 6

tom\_pittgoogle.consumer\_database\_python, 12

tom\_pittgoogle.consumer\_stream\_python, 9

tom\_pittgoogle.consumer\_stream\_rest, 7

### F

fetch\_alerts() (tom\_pittgoogle.broker\_database\_python.BrokerDatabasePython method), 12

## R

`request_alerts()` (*tom\_pittgoogle.broker\_database\_python.BrokerDatabasePython*  
*method*), 12

`request_alerts()` (*tom\_pittgoogle.broker\_stream\_rest.BrokerStreamRest*  
*method*), 6

## S

`save_alert()` (*tom\_pittgoogle.consumer\_stream\_python.ConsumerStreamPython*  
*method*), 11

`stream_alerts()` (*tom\_pittgoogle.consumer\_stream\_python.ConsumerStreamPython*  
*method*), 11

## T

`to_generic_alert()` (*tom\_pittgoogle.broker\_database\_python.BrokerDatabasePython*  
*method*), 12

`to_generic_alert()` (*tom\_pittgoogle.broker\_stream\_python.BrokerStreamPython*  
*method*), 9

`to_generic_alert()` (*tom\_pittgoogle.broker\_stream\_rest.BrokerStreamRest*  
*method*), 6

`tom_pittgoogle.broker_database_python`  
module, 12

`tom_pittgoogle.broker_stream_python`  
module, 8

`tom_pittgoogle.broker_stream_rest`  
module, 6

`tom_pittgoogle.consumer_database_python`  
module, 12

`tom_pittgoogle.consumer_stream_python`  
module, 9

`tom_pittgoogle.consumer_stream_rest`  
module, 7

## U

`unpack_and_ack_messages()`  
(*tom\_pittgoogle.consumer\_stream\_rest.ConsumerStreamRest*  
*method*), 8

`unpack_query()` (*tom\_pittgoogle.consumer\_database\_python.ConsumerDatabasePython*  
*method*), 13

`user_filter()` (*tom\_pittgoogle.broker\_stream\_python.BrokerStreamPython*  
*static method*), 9

`user_filter()` (*tom\_pittgoogle.broker\_stream\_rest.BrokerStreamRest*  
*static method*), 6