
BrainFrame Python API

Aotu

Jun 26, 2020

CONTENTS

1	The BrainFrameAPI Class	3
1.1	Basic Methods	3
2	Streams	5
2.1	API Methods	5
2.2	Data Structures	7
3	Zones	9
3.1	API Methods	9
3.2	Data Structures	10
4	Alarms	11
4.1	API Methods	11
4.2	Data Structures	13
5	Analysis Results	17
5.1	API Methods	17
5.2	Data Structures	17
6	Identities	21
6.1	API Methods	21
6.2	Data Structures	24
7	Capsules	25
7.1	API Methods	25
7.2	Data Structures	26
8	Premises	29
8.1	API Methods	29
8.2	Data Structures	30
9	Storage	31
9.1	API Methods	31
10	Users	33
10.1	API Methods	33
10.2	Data Structures	34
11	Licenses	35
11.1	API Methods	35
11.2	Data Structures	35

12 Exceptions	37
13 Sorting	41
14 Indices and tables	43
Python Module Index	45
Index	47

The BrainFrame Python API is a wrapper around the BrainFrame server's REST API, making it easier for Python users to build software that uses BrainFrame as a video streaming and analysis platform. The data structures and methods available in this library correspond closely with what's described in the [REST API Documentation](#).

The documentation is split into categories, each of which have descriptions for data structures and methods. Some example code is also provided.

The BrainFrame Python API is used by the BrainFrame Client and as a result is tested heavily internally before release.

THE BRAINFRAMEAPI CLASS

The `BrainFrameAPI` class holds methods corresponding to all API endpoints that the server provides. In order to get connected to a `BrainFrame` server, start by initializing a new instance of this class.

```
from brainframe.api import BrainFrameAPI

# Log into a local BrainFrame server instance with the default admin
# credentials
api = BrainFrameAPI("http://localhost", ("admin", "admin"))
```

Take a look at the next few topics for information on all the methods the `BrainFrameAPI` class has available and their corresponding data structures.

1.1 Basic Methods

class `BrainFrameAPI` (*server_url=None, credentials: Tuple[str, str] = None*)

Provides access to `BrainFrame` API endpoints.

Parameters

- **server_url** – The URL of the `BrainFrame` instance to connect to. If `None`, it needs to be set later with `set_url` before use
- **credentials** – The username and password as a tuple. Used to authenticate with the server. If `None`, no authentication information will be provided.

close()

Clean up the API. It may no longer be used after this call.

version (*timeout=30*) → str

Returns The current `BrainFrame` version in the format `X.Y.Z`

wait_for_server_initialization (*timeout: float = None*)

Waits for the server to be ready to handle requests.

Parameters **timeout** – The maximum amount of time, in seconds, to wait for the server to start. If `None`, this method will wait indefinitely.

STREAMS

BrainFrame works by connecting to and running analysis on video streams. Streams can come from a variety of sources and have analysis be turned on or off at runtime.

Below is an example script that uploads a video file, starts streaming it, and turns on analysis for that stream.

```
from pathlib import Path

from brainframe.api import BrainFrameAPI
from brainframe.api.bf_codecs import StreamConfiguration, ConnType

api = BrainFrameAPI("http://localhost")

video_bytes = Path("video_file.mp4").read_bytes()
storage_id = api.new_storage(video_bytes, "video/mp4")

stream_config = api.set_stream_configuration(
    StreamConfiguration(
        name="My Video File",
        connection_type=ConnType.FILE,
        connection_options={"storage_id": storage_id},
        runtime_options={},
    )
)

api.start_analyzing(stream_config.id)
```

2.1 API Methods

`BrainFrameAPI.get_stream_configuration(stream_id, timeout=30)` → *brainframe.api.bf_codecs.config_codecs.StreamConfiguration*

Gets the StreamConfiguration with the given ID.

Parameters

- **stream_id** – The ID of the stream configuration to get
- **timeout** – The timeout to use for this request

Returns The stream configuration

`BrainFrameAPI.get_stream_configurations(premises_id=None, timeout=30)` → *List[brainframe.api.bf_codecs.config_codecs.StreamConfiguration]*

Get all StreamConfigurations that currently exist.

Returns [StreamConfiguration, StreamConfiguration, ...]

`BrainFrameAPI.set_stream_configuration(stream_configuration, timeout=30) → Optional[brainframe.api.bf_codecs.config_codecs.StreamConfiguration]`
Update an existing stream configuration or create a new one. If creating a new one, the `stream_configuration.id` will be `None`.

Parameters

- **stream_configuration** – `StreamConfiguration`
- **timeout** – The timeout to use for this request

Returns `StreamConfiguration`, initialized with an ID

`BrainFrameAPI.start_analyzing(stream_id, timeout=30)`
Starts analysis on this stream.

Parameters

- **stream_id** – The ID of the stream to start analysis on
- **timeout** – The timeout to use for this request

`BrainFrameAPI.stop_analyzing(stream_id, timeout=30)`
Stops analysis on this stream.

Parameters

- **stream_id** – The ID of the stream to stop analysis on
- **timeout** – The timeout to use for this request

Returns `True` or `False` if the server was able to start analysis on that stream. It could fail because: unable to start stream, or license restrictions.

`BrainFrameAPI.check_analyzing(stream_id, timeout=30) → bool`
Check if this stream is being analyzed

Parameters

- **stream_id** – The ID of the stream to check
- **timeout** – The timeout to use for this request

Returns `True` or `False` if the stream is being analyzed

`BrainFrameAPI.delete_stream_configuration(stream_id, timeout=120)`
Deletes a stream configuration with the given ID. Also stops analysis if analysis was running and closes the stream.

Parameters

- **stream_id** – The ID of the stream to delete
- **timeout** – The timeout to use for this request

`BrainFrameAPI.get_stream_url(stream_id, timeout=30) → str`
Gets the URL that the stream is available at.

Parameters

- **stream_id** – The ID of the stream to get a URL for
- **timeout** – The timeout to use for this request

Returns The URL

`BrainFrameAPI.get_runtime_options(stream_id: int, timeout=30) → Dict[str, object]`

Returns the runtime options for the stream with the given ID. This can also be read from a `StreamConfiguration` object.

Parameters

- **stream_id** – The ID of the stream to get runtime options from
- **timeout** – The timeout to use for this request

Returns Runtime options

`BrainFrameAPI.set_runtime_option_vals(stream_id: int, runtime_options: Dict[str, object], timeout=30)`

Sets a stream's runtime options to the given values.

Parameters

- **stream_id** – The ID of the stream to set runtime options for
- **runtime_options** – The runtime options
- **timeout** – The timeout to use for this request

2.2 Data Structures

```
class StreamConfiguration(name: str, premises_id: Optional[int], connection_type: brain-
    frame.api.bf_codecs.config_codecs.StreamConfiguration.ConnType,
    connection_options: dict, runtime_options: dict, metadata: dict =
    <factory>, id: Optional[int] = None)
```

Describes a video stream that BrainFrame may connect to and analyze.

```
class ConnType(value)
```

An enumeration.

```
FILE = 'file'
```

An uploaded video file

```
IP_CAMERA = 'ip_camera'
```

A network camera that uses RTSP or HTTP

```
WEBCAM = 'webcam'
```

A webcam (usually USB)

```
classmethod values()
```

```
connection_options: dict
```

Contains configuration describing how this stream can be connected to. The contents of this dict will depend on the connection type.

ConnType.IP_CAMERA: url: The URL of the IP camera to connect to

pipeline (optional): A custom GStreamer pipeline to connect to the IP camera with

ConnType.FILE: storage_id: The storage ID of the video file to stream

transcode (optional): If True, the video file will be transcoded before being streamed. By default, this value is True.

pipeline (optional): A custom GStreamer pipeline to connect to the hosted stream of this video file with

ConnType.WEBCAM: device_id: The Video4Linux device ID of the webcam

connection_type: `ConnType`

The type of stream this configuration is describing

id: `Optional[int] = None`

The unique ID of this stream

metadata: `dict`

Key-value pairs containing some custom user-defined set of data to further describe the stream

name: `str`

The human-readable name of the video stream

premises_id: `Optional[int]`

The ID of the premises that this stream is a part of, or None if this stream is not part of a premises

runtime_options: `dict`

Key-value pairs of configuration information that changes the runtime behavior of the video stream from its defaults

ZONES

BrainFrame allows you to specify areas of interest within a video stream and gain specific insights about that area. Every stream comes with a default zone called “Screen” which encompasses the full area of the stream. See the [Zones](#) section in the User Manual for more details.

3.1 API Methods

`BrainFrameAPI.get_zone(zone_id, timeout=30) → brainframe.api.bf_codecs.zone_codecs.Zone`
Get a specific zone.

Parameters

- **zone_id** – The ID of the zone to get
- **timeout** – The timeout to use for this request

`BrainFrameAPI.get_zones(stream_id=None, timeout=30) → List[brainframe.api.bf_codecs.zone_codecs.Zone]`
Gets all zones.

Parameters

- **stream_id** – If set, only zones in the stream with this ID are returned
- **timeout** – The timeout to use for this request

Returns Zones

`BrainFrameAPI.set_zone(zone: brainframe.api.bf_codecs.zone_codecs.Zone, timeout=30)`
Update or create a zone. If the Zone doesn’t exist, the zone.id must be None. An initialized Zone with an ID will be returned.

Parameters

- **zone** – A Zone object
- **timeout** – The timeout to use for this request

Returns Zone, initialized with an ID

`BrainFrameAPI.delete_zone(zone_id: int, timeout=30)`
Deletes a zone with the given ID.

Parameters

- **zone_id** – The ID of the zone to delete
- **timeout** – The timeout to use for this request

3.2 Data Structures

```
class Zone(name: str, stream_id: int, coords: List[List[int]], alarms:  
           List[brainframe.api.bf_codecs.alarm_codecs.ZoneAlarm] = <factory>, id: Optional[int]  
           = None)
```

The definition for a zone. It is a non-convex polygon or a line.

alarms: **List[ZoneAlarm]**

All alarms that are attached to the zone

coords: **List[List[int]]**

Coordinates that define the region the zone occupies. It is a list of lists which are two elements in size. The coordinates are in pixels where the top left of the frame is [0, 0].

Example: [[0, 0], [10, 10], [100, 500], [0, 500]]

get_alarm(*alarm_id*) → Optional[*brainframe.api.bf_codecs.alarm_codecs.ZoneAlarm*]

Parameters **alarm_id** – The ID of the alarm to search in the alarm list for

Returns The alarm with the given ID, or None if no alarm with that ID exists in this zone

id: **Optional[int] = None**

A unique identifier

name: **str**

A friendly name for the zone

stream_id: **int**

The ID of the stream this zone is in

ALARMS

Alarms allow BrainFrame to notify you when a specified condition happens within a zone. When the conditions for an alarm are met, the alarm generates alerts which contain the time at which the conditions occurred as well as a frame from the video at that time. See the [Alarms](#) section in the User Manual for details.

4.1 API Methods

`BrainFrameAPI.get_zone_alarm(alarm_id, timeout=30)` → *brainframe.api.bf_codecs.alarm_codecs.ZoneAlarm*
 Gets the zone alarm with the given ID.

Parameters

- **alarm_id** – The ID of the alarm to get
- **timeout** – The timeout to use for this request

Returns The alarm

`BrainFrameAPI.get_zone_alarms(stream_id: Optional[int] = None, zone_id: Optional[int] = None, timeout=30)` → *List[brainframe.api.bf_codecs.alarm_codecs.ZoneAlarm]*
 Gets all zone alarms that fit the given filters.

Parameters

- **stream_id** – If supplied, only zone alarms that are for a zone in this stream will be returned
- **zone_id** – if supplied, only zone alarms attached to this zone will be returned
- **timeout** – The timeout to use for this request

Returns Zone alarms

`BrainFrameAPI.set_zone_alarm(alarm: brainframe.api.bf_codecs.alarm_codecs.ZoneAlarm, timeout=30)` → *brainframe.api.bf_codecs.alarm_codecs.ZoneAlarm*
 Creates or updates a zone alarm.

A new zone alarm is created if the given zone alarm's ID is None.

Parameters

- **alarm** – The zone alarm to create or update
- **timeout** – The timeout to use for this request

Returns Created or updated zone alarm

`BrainFrameAPI.delete_zone_alarm(alarm_id, timeout=30)`

Deletes the zone alarm with the given ID.

Parameters

- **alarm_id** – The ID of the zone alarm to delete
- **timeout** – The timeout to use for this request

`BrainFrameAPI.get_alert(alert_id, timeout=30) → brainframe.api.bf_codecs.alarm_codecs.Alert`

Gets the alert with the given ID.

Parameters

- **alert_id** – The ID of the alert to get
- **timeout** – The timeout to use for this request

Returns The alert

`BrainFrameAPI.get_alerts(stream_id: Optional[int] = None, zone_id: Optional[int] = None, alarm_id: Optional[int] = None, verification: Optional[brainframe.api.stubs.alerts.AlertStubMixin.AlertVerificationQueryType] = <AlertVerificationQueryType.ALL: 7>, limit: Optional[int] = None, offset: Optional[int] = None, timeout=30) → Tuple[List[brainframe.api.bf_codecs.alarm_codecs.Alert], int]`

Gets all alerts that match a query

Parameters

- **stream_id** – The ID of the stream to get alerts for
- **zone_id** – The ID of the zone to get alerts for
- **alarm_id** – The ID of the alarm to get alerts for
- **verification** – The verification states of the alerts
- **limit** – The maximum number of alerts to return. If None, no limit will be applied
- **offset** – The offset from the most recent alerts to return. This is only useful when providing a limit.
- **timeout** – The timeout to use for this request

Returns A list of alerts, and the total number of alerts that fit this criteria, ignoring pagination (the limit and offset)

`BrainFrameAPI.set_alert_verification(alert_id, verified_as: bool, timeout=30)`

Sets an alert verified as True or False.

Parameters

- **alert_id** – The ID of the alert to set
- **verified_as** – Set verification to True or False
- **timeout** – The timeout to use for this request

Returns The modified Alert

`BrainFrameAPI.get_alert_frame(alert_id: int, timeout=30) → Optional[numpy.ndarray]`

Returns the frame saved for this alert, or None if no frame is recorded for this alert.

Parameters

- **alert_id** – The ID of the alert to get a frame for.

- **timeout** – The timeout to use for this request

Returns The image as loaded by OpenCV, or None

4.2 Data Structures

class Alert (*alarm_id: int, zone_id: int, stream_id: int, start_time: float, end_time: Optional[float], verified_as: Optional[bool], id: Optional[int] = None*)

This is sent when an Alarm has been triggered.

alarm_id: int

The ID of the alarm that this alert came from

end_time: Optional[float]

When the event stopped happening, in Unix Time (seconds), or None if the alert is ongoing.

id: Optional[int] = None

A unique identifier

start_time: float

When the event started happening, in Unix Time (seconds)

stream_id: int

The ID of the stream this alert pertains to

verified_as: Optional[bool]

- If True, then this alert was labeled by a person as legitimate
- If False, then this alert was labeled by a person as a false alarm
- If None, then this alert has not been reviewed by a person

zone_id: int

The ID of the zone this alert pertains to

class ZoneAlarm (*name: str, count_conditions: List[[brainframe.api.bf_codecs.condition_codecs.ZoneAlarmCountCondition](#)], rate_conditions: List[[brainframe.api.bf_codecs.condition_codecs.ZoneAlarmRateCondition](#)], use_active_time: bool, active_start_time: str, active_end_time: str, id: Optional[int] = None, zone_id: Optional[int] = None, stream_id: Optional[int] = None*)

This is the configuration for an alarm.

active_end_time: str

The time of day where this alarm starts being active, in the format “hh:mm:ss”

active_start_time: str

The time of day where this alarm starts being active, in the format “hh:mm:ss”

count_conditions: List[ZoneAlarmCountCondition]

All count conditions for this alarm

id: Optional[int] = None

A unique identifier

name: str

A friendly name for the zone alarm

rate_conditions: List[ZoneAlarmRateCondition]

All rate conditions for this alarm

stream_id: Optional[int] = None

The ID of the stream the associated zone is in

use_active_time: **bool**

If True, the alarm will only be triggered when the current time is between the active_start_time and active_end_time.

zone_id: **Optional[int] = None**

The ID of the zone this alarm is associated with

class IntersectionPointType (*value*)

The point on a detection that must be inside a zone for the detection to count as being inside the zone. The most commonly used intersection point is BOTTOM, which counts a detection as being inside a zone if the bottom center point of the detection is in the zone.

class ZoneAlarmCountCondition (*test: brainframe.api.bf_codecs.condition_codecs.ZoneAlarmCountCondition.TestType, check_value: int, with_class_name: str, with_attribute: Optional[brainframe.api.bf_codecs.detection_codecs.Attribute], window_duration: float, window_threshold: float, intersection_point: brainframe.api.bf_codecs.condition_codecs.IntersectionPointType, id: Optional[int] = None*)

A condition that must be met for an alarm to go off. Compares the number of objects in a zone to some number.

class TestType (*value*)

Defines the way a count condition compares the actual value to the alarm's test value.

classmethod values ()

check_value: **int**

The value to test the actual count against

id: **Optional[int] = None**

A unique identifier

intersection_point: **IntersectionPointType**

The point in each detection that must be within the zone in order for that detection to be counted as in that zone

test: **TestType**

The way that the check value will be compared to the actual count

window_duration: **float**

The sliding window duration for this condition

window_threshold: **float**

The portion of time during the sliding window duration that this condition must be true in order for the associated alarm to trigger

with_attribute: **Optional[Attribute]**

If provided, only objects that have this attribute value will be counted.

with_class_name: **str**

The class name of the objects to count

class ZoneAlarmRateCondition (*test: brainframe.api.bf_codecs.condition_codecs.ZoneAlarmRateCondition.TestType, duration: float, change: float, direction: brainframe.api.bf_codecs.condition_codecs.ZoneAlarmRateCondition.DirectionType, with_class_name: str, with_attribute: Optional[brainframe.api.bf_codecs.detection_codecs.Attribute], intersection_point: brainframe.api.bf_codecs.condition_codecs.IntersectionPointType, id: int = None*)

A condition that must be met for an alarm to go off. Compares the rate of change in the count of some object

against a test value.

class DirectionType (*value*)

Defines the direction of flow that a rate condition pertains to.

classmethod values ()

class TestType (*value*)

Defines the way a rate condition compares the actual rate value to the alarm's test value.

classmethod values ()

change: float

The rate change value to compare the actual rate value against

direction: DirectionType

The direction of flow this condition tests for

duration: float

The time in seconds for this rate change to occur

id: int = None

A unique identifier

intersection_point: IntersectionPointType

The point in each detection that must be within the zone in order for that detection to be counted as in that zone

test: TestType

The way that the change value will be compared to the actual rate

with_attribute: Optional[Attribute]

If provided, only objects that have this attribute will be counted in the rate calculation

with_class_name: str

The class name of the objects to measure rate of change for

ANALYSIS RESULTS

The BrainFrame server organizes detected objects by the zones that those detections were found in. Detections for a zone, alongside information on that zone's state and any active alerts for that zone, are contained in `ZoneStatus` objects. For every video frame where analysis is performed, a `ZoneStatus` object will be created for every zone in that frame.

5.1 API Methods

`BrainFrameAPI.get_latest_zone_statuses (timeout=30) → Dict[int, Dict[str, brain-frame.api.bf_codecs.zone_codecs.ZoneStatus]]`

This method gets all of the latest processed zone statuses for every zone and for every stream.

All active streams will have a key in the output dict.

Parameters `timeout` – The timeout to use for this request

Returns A dict whose keys are stream IDs and whose value is another dict. This nested dict's keys are zone names and their value is the `ZoneStatus` for that zone.

`BrainFrameAPI.get_zone_status_stream (timeout=None) → Generator[Dict[int, Dict[str, brain-frame.api.bf_codecs.zone_codecs.ZoneStatus]], None, None]`

Streams `ZoneStatus` results from the server as they are produced.

All active streams will have a key in the output dict.

Parameters `timeout` – The timeout to use for this request

Returns A generator that outputs dicts whose keys are stream IDs and whose value is another dict. This nested dict's keys are zone names and their value is the `ZoneStatus` for that zone.

5.2 Data Structures

`class Attribute (category: str, value: str)`

This holds an attribute of a detection. These should `_not_` be made on the client side

category: str

The category of attribute being described

value: str

The value for this attribute category

```
class Detection(class_name: str, coords: List[List[int]], children: List[Detection], attributes: Dict[str, str], with_identity: Optional[brainframe.api.bf_codecs.identity_codecs.Identity], extra_data: Dict[str, Any], track_id: Optional[uuid.UUID])
```

An object detected in a video frame. Detections can have attributes attached to them that provide more information about the object as well as other metadata like a unique tracking ID.

attributes: `Dict[str, str]`

A dict whose key is an attribute name and whose value is the value of that attribute. For example, a car detection may have an attribute whose key is 'type' and whose value is 'sedan'.

property center

Return the center of the detections coordinates

class_name: `str`

The class of object that was detected, like 'person' or 'car'

coords: `List[List[int]]`

The coordinates, in pixels, of the detection in the frame

extra_data: `Dict[str, Any]`

Any additional metadata describing this object

track_id: `Optional[uuid.UUID]`

If not None, this is a unique tracking ID for the object. This ID can be compared to detections from other frames to track the movement of an object over time.

with_identity: `Optional[Identity]`

If not None, this is the identity that this detection was recognized as.

```
class ZoneStatus(zone: brainframe.api.bf_codecs.zone_codecs.Zone, float, total_entered: dict, total_exited: dict, tstamp: List[brainframe.api.bf_codecs.detection_codecs.Detection], enter- ing: List[brainframe.api.bf_codecs.detection_codecs.Detection], exit- ing: List[brainframe.api.bf_codecs.detection_codecs.Detection], alerts: List[brainframe.api.bf_codecs.alarm_codecs.Alert])
```

The current status of everything going on inside a zone.

alerts: `List[Alert]`

A list of all active alerts for the zone at this frame

property detection_within_counts

The current count of each class type detected in the video.

Returns A dict whose keys are class names and whose values are the count for that class name

entering: `List[Detection]`

A list of all detections that entered the zone this frame

exiting: `List[Detection]`

A list of all detections that have exited the zone this frame

total_entered: `dict`

A dict of key-value pairs indicating how many objects have exited the zone. The key is the class name, and the value is the count.

total_exited: `dict`

A set of key-value pairs indicating how many objects have exited the zone. The key is the object type, and the value is the count.

tstamp: `float`

The time at which this ZoneStatus was created as a Unix timestamp in seconds

within: `List[Detection]`
A list of all detections within the zone

zone: `Zone`
The zone that this status pertains to

IDENTITIES

Identities are known specific instances of an object that can be later recognized. For example, a picture of a person's face can be encoded and associated with that person's identity so that they can be recognized when they're in a video stream.

Identities can also be associated with vectors directly in cases where the encoding of an object is known ahead-of-time, like in the case of a fiducial tag.

```
"""Creates an identity for a specific skew of car using a picture of that
car.
"""
from pathlib import Path

from brainframe.api import BrainFrameAPI
from brainframe.bf_codecs import Identity

api = BrainFrameAPI("http://localhost")

civic_image_bytes = Path("civic.jpg").read_bytes()
# Upload the image to the server
storage_id = api.new_storage(civic_image_bytes, "image/jpeg")

# Create the new identity for the car skew
identity = api.set_identity(Identity(
    unique_name="gl0-Rl6B-blue-sedan",
    nickname="Honda Civic 2018 Blue Sedan",
))
# Encode an image of the skew and associate that encoding with the identity
api.new_identity_image(identity.id, "car", storage_id)
```

6.1 API Methods

`BrainFrameAPI.get_identity(identity_id: int, timeout=30)` → *brain-frame.api.bf_codecs.identity_codecs.Identity*

Gets the identity with the given ID.

Parameters

- **identity_id** – The ID of the identity to get
- **timeout** – The timeout to use for this request

Returns Identity

```
BrainFrameAPI.get_identities(unique_name: str = None, encoded_for_class: str = None, search: Optional[str] = None, limit: int = None, offset: int = None, sort_by: brainframe.api.bf_codecs.sorting.SortOptions = None, timeout=30) → Tuple[List[brainframe.api.bf_codecs.identity_codecs.Identity], int]
```

Returns all identities from the server.

Parameters

- **unique_name** – If provided, identities will be filtered by only those who have the given unique name
- **encoded_for_class** – If provided, identities will be filtered for only those that have been encoded at least once for the given class
- **search** – If provided, only identities that in some way contain the given search query are returned. This is intended for UI search features, and as such the specific semantics of how the search is performed are subject to change.
- **limit** – If provided, the number of returned identities is limited to this value.
- **offset** – The offset to start limiting results from. This is only useful when providing a limit.
- **sort_by** – If provided, the results will be sorted by the given configuration
- **timeout** – The timeout to use for this request

Returns A list of identities, and the total number of identities that fit this criteria, ignoring pagination (the limit and offset)

```
BrainFrameAPI.set_identity(identity: brainframe.api.bf_codecs.identity_codecs.Identity, timeout=30) → brainframe.api.bf_codecs.identity_codecs.Identity
```

Updates or creates an identity. If the identity does not already exist, identity.id must be None. The returned identity will have an assigned ID.

Parameters

- **identity** – The identity to save or create
- **timeout** – The timeout to use for this request

Returns the saved identity

```
BrainFrameAPI.delete_identity(identity_id: int, timeout=30)
```

Deletes the identity with the given ID.

Parameters

- **identity_id** – The ID of the identity to delete
- **timeout** – The timeout to use for this request

```
BrainFrameAPI.new_identity_image(identity_id: int, class_name: str, storage_id: int, timeout=30)
```

Saves and encodes an image under the identity with the given ID.

Parameters

- **identity_id** – Identity to associate the image with
- **class_name** – The type of object this image shows and should be encoded for
- **storage_id** – The ID of the image in storage to encode
- **timeout** – The timeout to use for this request

`BrainFrameAPI.new_identity_vector(identity_id: int, class_name: str, vector: List[float], timeout=30) → int`

Saves the given vector under the identity with the given ID. In this case, a vector is simply a list of one or more numbers that describe some object in an image.

Parameters

- **identity_id** – Identity to associate the vector with
- **class_name** – The type of object this vector describes
- **vector** – The vector to save
- **timeout** – The timeout to use for this request

Returns The vector ID

`BrainFrameAPI.get_encoding(encoding_id, timeout=30) → brainframe.api.bf_codecs.identity_codecs.Encoding`

Get the encoding with the given ID.

Parameters

- **encoding_id** – The encoding ID to get an encoding for
- **timeout** – The timeout to use for this request

Returns The corresponding encoding

`BrainFrameAPI.get_encodings(identity_id: Optional[int] = None, class_name: Optional[str] = None, timeout=30) → List[brainframe.api.bf_codecs.identity_codecs.Encoding]`

Gets all encodings from the server that match the given filters.

Parameters

- **identity_id** – If specified, only encodings attached to this identity will be returned
- **class_name** – If specified, only encodings for the given class name will be returned
- **timeout** – The timeout to use for this request

Returns All encodings that match this filter

`BrainFrameAPI.get_encoding_class_names(identity_id: Optional[int] = None, timeout=30) → List[str]`

Get all unique class names for encodings that match the given filter.

Parameters

- **identity_id** – If specified, only class names for encodings attached to this identity will be returned
- **timeout** – The timeout to use for this request

Returns All class names from encodings that match this filter

`BrainFrameAPI.delete_encoding(encoding_id, timeout=30)`

Deletes the encoding with the given ID.

Parameters

- **encoding_id** – The ID of the encoding to delete
- **timeout** – The timeout to use for this request

`BrainFrameAPI.delete_encodings(identity_id=None, class_name=None, timeout=30)`

Deletes all encodings that match the given filter.

Parameters

- **identity_id** – If specified, only encodings that are associated with this identity will be deleted
- **class_name** – If specified, only encodings that are for this class will be deleted
- **timeout** – The timeout to use for this request

6.2 Data Structures

class Encoding (*identity_id: int, class_name: str, from_image: Optional[int], vector: List[int], id: Optional[int] = None*)

An encoding attached to an identity.

class_name: str

The class of object this encoding is for.

from_image: Optional[int]

The storage ID of the image that this encoding was created from, or None if this encoding was not created from an image.

id: Optional[int] = None

The unique ID of the encoding.

identity_id: int

The ID of the identity this encoding is associated with.

vector: List[int]

A low-dimensional representation of the object's appearance. This is what objects found in streams will be compared to in order to decide if the object is of the identity this encoding is associated with.

class Identity (*unique_name: str, nickname: str, metadata: dict = <factory>, id: Optional[int] = None*)

A specific, recognizable object or person.

id: Optional[int] = None

A unique identifier.

metadata: dict

Any additional user-defined information about the identity.

nickname: str

A display name for the identity which may not be unique, like a person's name.

unique_name: str

The unique id of the identified detection.

Not to be confused with the id of the object which is a field used by the database.

CAPSULES

A capsule is a single file with a `.cap` file extension. It contains the code, metadata, model files, and any other files the capsule needs to operate. See the [Introduction to Capsules](#) section in the User Manual for more information.

7.1 API Methods

`BrainFrameAPI.get_plugin(name, timeout=30) → brainframe.api.bf_codecs.plugin_codecs.Plugin`

Parameters

- **name** – The name of the plugin to get
- **timeout** – The timeout to use for this request

Returns Plugin with the given name

`BrainFrameAPI.get_plugins(timeout=30) → List[brainframe.api.bf_codecs.plugin_codecs.Plugin]`

Parameters **timeout** – The timeout to use for this request

Returns All available plugins

`BrainFrameAPI.get_plugin_option_vals(plugin_name, stream_id=None, timeout=30) → Dict[str, object]`

Gets the current values for every plugin option. See the documentation for the `PluginOption` codec for more info about global and stream level options and how they interact.

Parameters

- **plugin_name** – The plugin to find options for
- **stream_id** – The ID of the stream. If this value is `None`, then the global options are returned for that plugin
- **timeout** – The timeout to use for this request

Returns A dict where the key is the option name and the value is the option's current value

`BrainFrameAPI.set_plugin_option_vals(*, plugin_name, stream_id=None, option_vals: Dict[str, object], timeout=30)`

Sets option values for a plugin.

Parameters

- **plugin_name** – The name of the plugin whose options to set
- **stream_id** – The ID of the stream, if these are stream-level options. If this value is `None`, then the global options are set

- **option_vals** – A dict where the key is the name of the option to set, and the value is the value to set that option to
- **timeout** – The timeout to use for this request

`BrainFrameAPI.patch_plugin_option_vals(*, plugin_name, stream_id=None, option_vals: Dict[str, object], timeout=30)`

Patches option values for a plugin. Only the provided options are changed. To unset an option, provide that option with a value of None.

Parameters

- **plugin_name** – The name of the plugin whose options to set
- **stream_id** – The ID of the stream, if these are stream-level options. If this value is None, then the global options are set
- **option_vals** – A dict where the key is the name of the option to set, and the value is the value to set that option to
- **timeout** – The timeout to use for this request

`BrainFrameAPI.is_plugin_active(plugin_name, stream_id=None, timeout=30) → bool`

Returns True if the plugin is active. If a plugin is not marked as active, it will not run. Like plugin options, this can be configured globally and on a per-stream level.

Parameters

- **plugin_name** – The name of the plugin to get activity for
- **stream_id** – The ID of the stream, if you want the per-stream active setting
- **timeout** – The timeout to use for this request

Returns True if the plugin is active

`BrainFrameAPI.set_plugin_active(*, plugin_name, stream_id=None, active: Optional[bool], timeout=30)`

Sets whether or not the plugin is active. If a plugin is active, it will be run on frames.

Parameters

- **plugin_name** – The name of the plugin to set activity for
- **stream_id** – The ID of the stream, if you want to set the per-stream active setting
- **active** – True if the plugin should be set to active
- **timeout** – The timeout to use for this request

7.2 Data Structures

`class NodeDescription (size: brainframe.api.bf_codecs.plugin_codecs.NodeDescription.Size, detections: List[str], attributes: Dict[str, List[str]], encoded: bool, tracked: bool, extra_data: List[str])`

A description of a DetectionNode, used by plugins to define what kinds of inputs and outputs a plugin uses.

`class Size (value)`

Describes the amount of DetectionNodes a plugin takes as input or provides as output.

ALL = 'all'

Input: The plugin takes all instances of a class as input, like for a tracker.

Output: The plugin provides all instances of a class as output, like for a detector.

NONE = 'none'

Input: The plugin takes nothing as input, like for an object detector.

Output: Plugins cannot have a NONE output.

SINGLE = 'single'

Input: The plugin takes a single `DetectionNode` as input, like for a classifier.

Output: The plugin provides a single modified `DetectionNode` as output, like for a classifier.

classmethod values()

attributes: Dict[str, List[str]]

Key-value pairs whose key is the classification type and whose value is a list of possible attributes. A `DetectionNode` that meets this description must have a classification for each classification type listed here.

detections: List[str]

A list of detection class names, like “person” or “vehicle”. A `DetectionNode` that meets this description must have a class name that is present in this list.

encoded: bool

If True, the `DetectionNode` must be encoded to meet this description

extra_data: List[str]

A list of keys in a `NodeDescription`’s `extra_data`. A `DetectionNode` that meets this description must have extra data for each name listed here.

size: Size

Describes the amount of `DetectionNodes` the node either takes in as input or provides as output

tracked: bool

If True, the `DetectionNode` must be tracked to meet this description

```
class Plugin (name: str, version: int, description: str, input_type: brain-
                frame.api.bf_codecs.plugin_codecs.NodeDescription, output_type: brain-
                frame.api.bf_codecs.plugin_codecs.NodeDescription, capability: brain-
                frame.api.bf_codecs.plugin_codecs.NodeDescription, options: Dict[str, brain-
                frame.api.bf_codecs.plugin_codecs.PluginOption])
```

Metadata on a loaded plugin.

capability: NodeDescription

A `NodeDescription` which describes what this plugin does to its input. It is the difference between the input and output `NodeDescriptions`. This field is useful for inspecting a plugin to find what it can do.

description: str

A human-readable description of what the plugin does

input_type: NodeDescription

Describes the type of inference data that this plugin takes as input

name: str

The name of the plugin

options: Dict[str, PluginOption]

A dict describing the configurable options of this plugin

output_type: NodeDescription

Describes the type of inference data that this plugin produces

version: int

The plugin’s version

```
class PluginOption (type: brainframe.api.bf_codecs.plugin_codecs.PluginOption.Type, default: Any,  
                    constraints: dict, description: str)
```

A single configuration option for a plugin. Defines what type of option it is and its potential values.

There are two kinds of plugin options. Stream plugin options apply only to the stream they are attached to. Global plugin options apply to all streams, but are overridden by stream plugin options.

```
class Type (value)
```

The data type of a plugin option

```
classmethod values ()
```

```
constraints: dict
```

Describes the range of valid values for this option. The content of this dict depends on the type field.

OptionType.FLOAT: max_val: The maximum valid float value

min_val: The minimum valid float value

OptionType.INT: max_val: The maximum valid int value

min_val: The minimum valid int value

OptionType.ENUM: choices: A list of strings. The option's value must be one of these strings.

OptionType.BOOL: This object has no constraints.

```
default: Any
```

The default value for this option

```
description: str
```

A human-readable description of the plugin's capabilities

```
type: Type
```

The data type of this option's value

PREMISES

By default, the BrainFrame server connects to IP cameras directly. This necessitates that the IP camera is either on the same network as the server, or is made accessible through other means, like port forwarding.

For situations where this is undesirable, a stream may optionally be configured to be part of a premises. BrainFrame will connect to these streams through the StreamGateway instead, which acts as a proxy to bypass firewalls. See the [Premises](#) section in the User Manual for details.

8.1 API Methods

`BrainFrameAPI.get_premises (premises_id: int, timeout=30) → brainframe.api.bf_codecs.premises_codecs.Premises`
Gets the premises with the given ID.

Parameters

- **premises_id** – The ID of the premises to get
- **timeout** – The timeout to use for this request

Returns The premises

`BrainFrameAPI.get_all_premises (timeout=30) → List[brainframe.api.bf_codecs.premises_codecs.Premises]`
Gets all premises.

`BrainFrameAPI.set_premises (premises: brainframe.api.bf_codecs.premises_codecs.Premises, timeout=30)`

Update or create a premises. If the Premises doesn't exist, the premises.id must be None. An initialized Premises with an ID will be returned.

Premises are more often created using the StreamGateway client tool.

Parameters

- **premises** – A Premises object
- **timeout** – The timeout to use for this request

Returns Premises, initialized with an ID

`BrainFrameAPI.delete_premises (premises_id: int, timeout=30)`
Delete a premises and the streams and data connected to it.

Parameters

- **premises_id** – The ID of the premises to delete
- **timeout** – The timeout to use for this request

8.2 Data Structures

```
class Premises (name: str, id: Optional[int] = None)  
    Information about a specific Premises  
  
    id: Optional[int] = None  
        The unique ID of the premises  
  
    name: str  
        The friendly name of the premises
```

STORAGE

The BrainFrame server has a generic API for file storage that is used by other API endpoints. If a method asks for a `storage_id` field, it's looking for a file uploaded using these methods.

9.1 API Methods

`BrainFrameAPI.get_storage_data(storage_id, timeout=30) → Tuple[bytes, str]`

Returns the data with the given storage ID.

Parameters

- **storage_id** – The ID of the storage object to get
- **timeout** – The timeout to use for this request

Returns The data and MIME type of that data

`BrainFrameAPI.get_storage_data_as_image(storage_id, timeout=30) → numpy.ndarray`

Gets the data with the given storage ID and attempts to load it as an image with OpenCV.

Parameters

- **storage_id** – The ID of the storage object to get
- **timeout** – The timeout to use for this request

Returns A numpy array in OpenCV format

`BrainFrameAPI.new_storage(data: Union[bytes, BinaryIO, Iterable], mime_type: str, timeout=30) → int`

Stores the given data.

Parameters

- **data** – The data to store, either as bytes or as a file-like
- **mime_type** – The MIME type of the data
- **timeout** – The timeout to use for this request

Returns The storage ID

`BrainFrameAPI.new_storage_as_image(data: bytes, timeout=30) → int`

Stores the given image data, and inspects it to figure out what the MIME type of the data.

Parameters

- **data** – The image data to store
- **timeout** – The timeout to use for this request

Returns The storage ID

`BrainFrameAPI.delete_storage(storage_id, timeout=30)`

Deletes the storage object with the given ID. Deleting storage objects that are in use, like for an encoding, is not recommended.

Parameters

- **storage_id** – The ID of the storage object to delete
- **timeout** – The timeout to use for this request

A single BrainFrame server can have multiple users associated with it. Each user has their own username and password, and can be assigned roles that grant them different levels of access to the server.

Note that authorization is disabled by default for BrainFrame servers. See the [Authorization Configuration](#) section in the User Manual for details.

10.1 API Methods

`BrainFrameAPI.get_user(user_id, timeout=30) → brainframe.api.bf_codecs.user_codecs.User`
Gets the user with the given ID.

Parameters

- **user_id** – The ID of the user to get
- **timeout** – The timeout to use for this request

Returns The user

`BrainFrameAPI.get_users(timeout=30) → List[brainframe.api.bf_codecs.user_codecs.User]`
Gets all users.

Parameters **timeout** – The timeout to use for this request

Returns Users

`BrainFrameAPI.set_user(user, timeout=30) → brainframe.api.bf_codecs.user_codecs.User`
Creates or updates a user. Only admin users may make this request.

A new user is created if the given user's ID is None.

Parameters

- **user** – The user to create or update
- **timeout** – The timeout to use for this request

Returns Created or updated user

`BrainFrameAPI.delete_user(user_id, timeout=30)`
Deletes a user. Only admin users may make this request.

Parameters

- **user_id** – The ID of the user to delete
- **timeout** – The timeout to use for this request

10.2 Data Structures

```
class User (username: str, password: Optional[str], role: brain-  
             frame.api.bf_codecs.user_codecs.User.Role, id: Optional[int] = None)  
    Contains information on a user.  
  
    class Role (value)  
        Controls the level of access a user has to API endpoints.  
  
        ADMIN = 'admin'  
            A user that can access all endpoints.  
  
        EDITOR = 'editor'  
            A user that can access most endpoints but cannot do administrative tasks like adding users and man-  
            aging the license.  
  
        classmethod values ()  
  
    id: Optional[int] = None  
        The user's unique ID  
  
    password: Optional[str]  
        This field will be None when retrieving users from the server. It should only be set by the client when  
        creating a new user or updating a user's password.  
  
    role: Role  
        The user's role  
  
    username: str  
        The username used for login
```

LICENSES

The license controls the terms of access to the BrainFrame server. These APIs allow for BrainFrame instances to be set up with licenses through an automated system. The average user will probably not need these APIs.

11.1 API Methods

`BrainFrameAPI.get_license_info (timeout=30) → brainframe.api.bf_codecs.license_codecs.LicenseInfo`
Gets licensing information from the server.

Parameters `timeout` – The timeout to use for this request

Returns License info

`BrainFrameAPI.set_license_key (license_key: str, timeout=30) → brainframe.api.bf_codecs.license_codecs.LicenseInfo`
Uploads a license key to the server and applies it.

Parameters

- **license_key** – A base64-encoded string of license data
- **timeout** – The timeout to use for this request

Returns License info after the key is applied

11.2 Data Structures

`DATE_FORMAT = '%Y-%m-%d'`

ISO 8601 date format used by the API

`class LicenseInfo (state: brainframe.api.bf_codecs.license_codecs.LicenseInfo.State, terms: Optional[brainframe.api.bf_codecs.license_codecs.LicenseTerms])`

Information on the licensing status of the server

`class State (value)`

An enumeration.

`EXPIRED = 'expired'`

A license was provided, but it has expired

`INVALID = 'invalid'`

A license was provided, but did not pass validation

`MISSING = 'missing'`

No license was provided

VALID = 'valid'

A valid license is loaded, features should be enabled

state: State

The licensing state of the server.

terms: Optional[LicenseTerms]

The active license terms of the server, or None if no license is loaded.

class LicenseTerms (*online_checkin: bool, max_streams: int, journal_max_allowed_age: float, expiration_date: Optional[datetime.date]*)

The terms of the currently active license.

expiration_date: Optional[date]

The date that this license expires, or None if the license is perpetual.

journal_max_allowed_age: float

The maximum amount of time in seconds that the server may hold data in the journal for.

max_streams: int

The maximum number of streams that may have analysis enabled at any given time.

online_checkin: bool

If True, the server will check in with a remote licensing server to verify license terms.

EXCEPTIONS

These are all errors that can be reported by API requests. These errors mirror those used in the REST API.

exception AdminMustExistError (*description*)

There was an attempt to delete the only remaining admin account.

exception AlertNotFound (*description*)

An Alert specified by the client could not be found.

exception AnalysisLimitExceededError (*description*)

There was an attempt to start analysis on a stream, but the maximum amount of streams that may have analysis run on them at once has already been reached.

exception BaseAPIError (*description*)

All API errors subclass this error.

exception DuplicateIdentityNameError (*description*)

There was an attempt to create a new identity with the same name as another identity.

exception DuplicateStreamSourceError (*description*)

There was an attempted to create a stream configuration with the same source as an existing one.

exception DuplicateUsernameError (*description*)

The requested username already exists.

exception DuplicateVectorError (*description*)

There was an attempt to add a vector that already exists for the given identity and class.

exception DuplicateZoneNameError (*description*)

There was an attempt to make a zone with the same name as another zone within the same stream.

exception EncodingNotFound (*description*)

There was an attempt to access an encoding that does not exist.

exception FrameNotFoundForAlertError (*description*)

There was an attempt to get a frame for an alert that has no frame.

exception IdentityNotFound (*description*)

An identity specified by the client could not be found.

exception ImageAlreadyEncodedError (*description*)

There was an attempt to encode an image that has already been encoded for a given identity and a given class.

exception ImageNotFoundForIdentityError (*description*)

An image specified by the client could not be found for the specified identity.

exception InsufficientRoleError (*description*)

A user attempted an operation that they don't have permission to do.

exception InvalidFormatError (*description*)

The request was parsed, but some value within the request is invalid.

exception InvalidImageTypeError (*description*)

An image could not be decoded by OpenCV

exception InvalidPluginOptionError (*description*)

The provided plugin options do not work for the given plugin. This could be because the option does not exist or the value for that option doesn't fit the constraints.

exception InvalidRuntimeOptionError (*description*)

There was an attempt to set a runtime option that is not supported or is of the wrong type.

exception InvalidSessionError (*description*)

There was an attempt to access the API with an invalid session ID, either because the session expired or no session with that ID has ever existed. The client should authenticate again to get a new session.

exception InvalidSyntaxError (*description*)

The syntax of the request could not be parsed.

exception LicenseExpiredError (*description*)

There was an attempt to upload a license that is expired.

exception LicenseInvalidError (*description*)

There was an attempt to upload a license that is in an invalid format.

exception LicenseRequiredError (*description*)

There was an attempt to access a resource that requires an active license while no valid license is loaded.

exception NoDetectionsInImageError (*description*)

There was an attempt to encode an image with no objects of the given class in the frame.

exception NoDetectorForClassError (*description*)

There was an attempt to use a class name that is not detectable.

exception NoEncoderForClassError (*description*)

There was an attempt to create an identity for a class that is not encodable.

exception NotImplementedInAPIError (*description*)

The client requested something that is not currently implemented.

exception PluginNotFoundError (*description*)

There was an attempt to reference a plugin that does not exist.

exception PremisesNotFoundError (*description*)

A Zone specified by the client could not be found.

exception RemoteConnectionError (*description*)

The server encountered an error while connecting to a remote resource that is required for the requested operation.

exception StorageNotFoundError (*description*)

There was an attempt to access a storage object that does not exist.

exception StreamConfigNotFoundError (*description*)

A StreamConfiguration specified by the client could not be found.

exception StreamNotOpenedError (*description*)

A stream failed to open when it was required to.

exception TooManyDetectionsInImageError (*description*)

There was an attempt to encode an image with more than one object of the given class in the frame, causing ambiguity on which one to encode.

exception UnauthorizedError (*description*)

There was an attempt to access the API without proper authorization.

exception UnknownError (*description*, *status_code=None*)

Something unexpected happened. The server may be in an invalid state.

exception UserNotFoundError (*description*)

There was an attempt to access a user by ID that does not exist.

exception VectorTooLongError (*description*)

The provided encoding vector is longer than the maximum allowed length.

exception ZoneAlarmNotFoundError (*description*)

There was an attempt to access a zone alarm that does not exist.

exception ZoneNotDeletableError (*description*)

A client tried to delete a default Zone

exception ZoneNotFoundError (*description*)

A Zone specified by the client could not be found.

SORTING

Some operations can have their results sorted by the value of their fields. These helper data structures are used to define the desired order.

class Ordering (*value*)

Specifies in what order a field should be sorted by.

ASC = 'asc'

The order of the field should be ascending (from low to high)

DESC = 'desc'

The order of the field should be descending (from high to low)

class SortOptions (*field_name: str, ordering: [brainframe.api.bf_codecs.sorting.Ordering](#)*)

A sorting configuration. Used by some APIs that provide many of a certain object.

field_name: str

The name of the field to sort by

ordering: Ordering

The order to sort the field by

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

b

brainframe.api.bf_codecs.alarm_codecs,
13
brainframe.api.bf_codecs.condition_codecs,
14
brainframe.api.bf_codecs.config_codecs,
7
brainframe.api.bf_codecs.detection_codecs,
17
brainframe.api.bf_codecs.identity_codecs,
24
brainframe.api.bf_codecs.license_codecs,
35
brainframe.api.bf_codecs.plugin_codecs,
26
brainframe.api.bf_codecs.premises_codecs,
30
brainframe.api.bf_codecs.sorting, 41
brainframe.api.bf_codecs.user_codecs,
34
brainframe.api.bf_errors, 37

A

active_end_time (*ZoneAlarm attribute*), 13
 active_start_time (*ZoneAlarm attribute*), 13
 ADMIN (*User.Role attribute*), 34
 AdminMustExistError, 37
 alarm_id (*Alert attribute*), 13
 alarms (*Zone attribute*), 10
 Alert (class in *brainframe.api.bf_codecs.alarm_codecs*), 13
 AlertNotFoundError, 37
 alerts (*ZoneStatus attribute*), 18
 ALL (*NodeDescription.Size attribute*), 26
 AnalysisLimitExceededError, 37
 ASC (*Ordering attribute*), 41
 Attribute (class in *brainframe.api.bf_codecs.detection_codecs*), 17
 attributes (*Detection attribute*), 18
 attributes (*NodeDescription attribute*), 27

B

BaseAPIError, 37
 brainframe.api.bf_codecs.alarm_codecs
 module, 13
 brainframe.api.bf_codecs.condition_codecs
 module, 14
 brainframe.api.bf_codecs.config_codecs
 module, 7
 brainframe.api.bf_codecs.detection_codecs
 module, 17
 brainframe.api.bf_codecs.identity_codecs
 module, 24
 brainframe.api.bf_codecs.license_codecs
 module, 35
 brainframe.api.bf_codecs.plugin_codecs
 module, 26
 brainframe.api.bf_codecs.premises_codecs
 module, 30
 brainframe.api.bf_codecs.sorting
 module, 41
 brainframe.api.bf_codecs.user_codecs
 module, 34
 brainframe.api.bf_errors

module, 37

BrainFrameAPI (class in *brainframe.api*), 3

C

capability (*Plugin attribute*), 27
 category (*Attribute attribute*), 17
 center () (*Detection property*), 18
 change (*ZoneAlarmRateCondition attribute*), 15
 check_analyzing () (*BrainFrameAPI method*), 6
 check_value (*ZoneAlarmCountCondition attribute*), 14
 class_name (*Detection attribute*), 18
 class_name (*Encoding attribute*), 24
 close () (*BrainFrameAPI method*), 3
 connection_options (*StreamConfiguration attribute*), 7
 connection_type (*StreamConfiguration attribute*), 7
 constraints (*PluginOption attribute*), 28
 coords (*Detection attribute*), 18
 coords (*Zone attribute*), 10
 count_conditions (*ZoneAlarm attribute*), 13

D

DATE_FORMAT (in module *brainframe.api.bf_codecs.license_codecs*), 35
 default (*PluginOption attribute*), 28
 delete_encoding () (*BrainFrameAPI method*), 23
 delete_encodings () (*BrainFrameAPI method*), 23
 delete_identity () (*BrainFrameAPI method*), 22
 delete_premises () (*BrainFrameAPI method*), 29
 delete_storage () (*BrainFrameAPI method*), 32
 delete_stream_configuration () (*BrainFrameAPI method*), 6
 delete_user () (*BrainFrameAPI method*), 33
 delete_zone () (*BrainFrameAPI method*), 9
 delete_zone_alarm () (*BrainFrameAPI method*), 11
 DESC (*Ordering attribute*), 41
 description (*Plugin attribute*), 27
 description (*PluginOption attribute*), 28
 Detection (class in *brainframe.api.bf_codecs.detection_codecs*), 17

`detection_within_counts()` (*ZoneStatus* property), 18
`detections` (*NodeDescription* attribute), 27
`direction` (*ZoneAlarmRateCondition* attribute), 15
`DuplicateIdentityNameError`, 37
`DuplicateStreamSourceError`, 37
`DuplicateUsernameError`, 37
`DuplicateVectorError`, 37
`DuplicateZoneNameError`, 37
`duration` (*ZoneAlarmRateCondition* attribute), 15

E

`EDITOR` (*User.Role* attribute), 34
`encoded` (*NodeDescription* attribute), 27
`Encoding` (class in *brainframe.api.bf_codecs.identity_codecs*), 24
`EncodingNotFoundError`, 37
`end_time` (*Alert* attribute), 13
`entering` (*ZoneStatus* attribute), 18
`exiting` (*ZoneStatus* attribute), 18
`expiration_date` (*LicenseTerms* attribute), 36
`EXPIRED` (*LicenseInfo.State* attribute), 35
`extra_data` (*Detection* attribute), 18
`extra_data` (*NodeDescription* attribute), 27

F

`field_name` (*SortOptions* attribute), 41
`FILE` (*StreamConfiguration.ConnType* attribute), 7
`FrameNotFoundForAlertError`, 37
`from_image` (*Encoding* attribute), 24

G

`get_alarm()` (*Zone* method), 10
`get_alert()` (*BrainFrameAPI* method), 12
`get_alert_frame()` (*BrainFrameAPI* method), 12
`get_alerts()` (*BrainFrameAPI* method), 12
`get_all_premises()` (*BrainFrameAPI* method), 29
`get_encoding()` (*BrainFrameAPI* method), 23
`get_encoding_class_names()` (*BrainFrameAPI* method), 23
`get_encodings()` (*BrainFrameAPI* method), 23
`get_identities()` (*BrainFrameAPI* method), 21
`get_identity()` (*BrainFrameAPI* method), 21
`get_latest_zone_statuses()` (*BrainFrameAPI* method), 17
`get_license_info()` (*BrainFrameAPI* method), 35
`get_plugin()` (*BrainFrameAPI* method), 25
`get_plugin_option_vals()` (*BrainFrameAPI* method), 25
`get_plugins()` (*BrainFrameAPI* method), 25
`get_premises()` (*BrainFrameAPI* method), 29
`get_runtime_options()` (*BrainFrameAPI* method), 6
`get_storage_data()` (*BrainFrameAPI* method), 31

`get_storage_data_as_image()` (*BrainFrameAPI* method), 31
`get_stream_configuration()` (*BrainFrameAPI* method), 5
`get_stream_configurations()` (*BrainFrameAPI* method), 5
`get_stream_url()` (*BrainFrameAPI* method), 6
`get_user()` (*BrainFrameAPI* method), 33
`get_users()` (*BrainFrameAPI* method), 33
`get_zone()` (*BrainFrameAPI* method), 9
`get_zone_alarm()` (*BrainFrameAPI* method), 11
`get_zone_alarms()` (*BrainFrameAPI* method), 11
`get_zone_status_stream()` (*BrainFrameAPI* method), 17
`get_zones()` (*BrainFrameAPI* method), 9

I

`id` (*Alert* attribute), 13
`id` (*Encoding* attribute), 24
`id` (*Identity* attribute), 24
`id` (*Premises* attribute), 30
`id` (*StreamConfiguration* attribute), 8
`id` (*User* attribute), 34
`id` (*Zone* attribute), 10
`id` (*ZoneAlarm* attribute), 13
`id` (*ZoneAlarmCountCondition* attribute), 14
`id` (*ZoneAlarmRateCondition* attribute), 15
`Identity` (class in *brainframe.api.bf_codecs.identity_codecs*), 24
`identity_id` (*Encoding* attribute), 24
`IdentityNotFoundError`, 37
`ImageAlreadyEncodedError`, 37
`ImageNotFoundForIdentityError`, 37
`input_type` (*Plugin* attribute), 27
`InsufficientRoleError`, 37
`intersection_point` (*ZoneAlarmCountCondition* attribute), 14
`intersection_point` (*ZoneAlarmRateCondition* attribute), 15
`IntersectionPointType` (class in *brainframe.api.bf_codecs.condition_codecs*), 14
`INVALID` (*LicenseInfo.State* attribute), 35
`InvalidFormatError`, 37
`InvalidImageTypeError`, 38
`InvalidPluginOptionError`, 38
`InvalidRuntimeOptionError`, 38
`InvalidSessionError`, 38
`InvalidSyntaxError`, 38
`IP_CAMERA` (*StreamConfiguration.ConnType* attribute), 7
`is_plugin_active()` (*BrainFrameAPI* method), 26

J

`journal_max_allowed_age` (*LicenseTerms* at-

tribute), 36

L

LicenseExpiredError, 38

LicenseInfo (class in brainframe.api.bf_codecs.license_codecs), 35

LicenseInfo.State (class in brainframe.api.bf_codecs.license_codecs), 35

LicenseInvalidError, 38

LicenseRequiredError, 38

LicenseTerms (class in brainframe.api.bf_codecs.license_codecs), 36

M

max_streams (LicenseTerms attribute), 36

metadata (Identity attribute), 24

metadata (StreamConfiguration attribute), 8

MISSING (LicenseInfo.State attribute), 35

module

brainframe.api.bf_codecs.alarm_codecs,

13

brainframe.api.bf_codecs.condition_codecs,

14

brainframe.api.bf_codecs.config_codecs,

7

brainframe.api.bf_codecs.detection_codecs,

17

brainframe.api.bf_codecs.identity_codecs,

24

brainframe.api.bf_codecs.license_codecs,

35

brainframe.api.bf_codecs.plugin_codecs,

26

brainframe.api.bf_codecs.premises_codecs,

30

brainframe.api.bf_codecs.sorting,

41

brainframe.api.bf_codecs.user_codecs,

34

brainframe.api.bf_errors, 37

N

name (Plugin attribute), 27

name (Premises attribute), 30

name (StreamConfiguration attribute), 8

name (Zone attribute), 10

name (ZoneAlarm attribute), 13

new_identity_image() (BrainFrameAPI method), 22

new_identity_vector() (BrainFrameAPI method), 23

new_storage() (BrainFrameAPI method), 31

new_storage_as_image() (BrainFrameAPI method), 31

nickname (Identity attribute), 24

NodeDescription (class in brainframe.api.bf_codecs.plugin_codecs), 26

NodeDescription.Size (class in brainframe.api.bf_codecs.plugin_codecs), 26

NoDetectionsInImageError, 38

NoDetectorForClassError, 38

NoEncoderForClassError, 38

NONE (NodeDescription.Size attribute), 26

NotImplementedInAPIError, 38

O

online_checkin (LicenseTerms attribute), 36

options (Plugin attribute), 27

Ordering (class in brainframe.api.bf_codecs.sorting), 41

ordering (SortOptions attribute), 41

output_type (Plugin attribute), 27

P

password (User attribute), 34

patch_plugin_option_vals() (BrainFrameAPI method), 26

Plugin (class in brainframe.api.bf_codecs.plugin_codecs), 27

PluginNotFoundError, 38

PluginOption (class in brainframe.api.bf_codecs.plugin_codecs), 27

PluginOption.Type (class in brainframe.api.bf_codecs.plugin_codecs), 28

Premises (class in brainframe.api.bf_codecs.premises_codecs), 30

premises_id (StreamConfiguration attribute), 8

PremisesNotFoundError, 38

R

rate_conditions (ZoneAlarm attribute), 13

RemoteConnectionError, 38

role (User attribute), 34

runtime_options (StreamConfiguration attribute), 8

S

set_alert_verification() (BrainFrameAPI method), 12

set_identity() (BrainFrameAPI method), 22

set_license_key() (BrainFrameAPI method), 35

set_plugin_active() (BrainFrameAPI method), 26

set_plugin_option_vals() (BrainFrameAPI method), 25

set_premises() (BrainFrameAPI method), 29

set_runtime_option_vals() (BrainFrameAPI method), 7

set_stream_configuration() (BrainFrameAPI method), 5

`set_user()` (*BrainFrameAPI* method), 33
`set_zone()` (*BrainFrameAPI* method), 9
`set_zone_alarm()` (*BrainFrameAPI* method), 11
`SINGLE` (*NodeDescription.Size* attribute), 27
`size` (*NodeDescription* attribute), 27
`SortOptions` (class in *brainframe.api.bf_codecs.sorting*), 41
`start_analyzing()` (*BrainFrameAPI* method), 6
`start_time` (*Alert* attribute), 13
`state` (*LicenseInfo* attribute), 36
`stop_analyzing()` (*BrainFrameAPI* method), 6
`StorageNotFoundError`, 38
`stream_id` (*Alert* attribute), 13
`stream_id` (*Zone* attribute), 10
`stream_id` (*ZoneAlarm* attribute), 13
`StreamConfigNotFoundError`, 38
`StreamConfiguration` (class in *brainframe.api.bf_codecs.config_codecs*), 7
`StreamConfiguration.ConnType` (class in *brainframe.api.bf_codecs.config_codecs*), 7
`StreamNotOpenedError`, 38

T

`terms` (*LicenseInfo* attribute), 36
`test` (*ZoneAlarmCountCondition* attribute), 14
`test` (*ZoneAlarmRateCondition* attribute), 15
`TooManyDetectionsInImageError`, 38
`total_entered` (*ZoneStatus* attribute), 18
`total_exited` (*ZoneStatus* attribute), 18
`track_id` (*Detection* attribute), 18
`tracked` (*NodeDescription* attribute), 27
`tstamp` (*ZoneStatus* attribute), 18
`type` (*PluginOption* attribute), 28

U

`UnauthorizedError`, 38
`unique_name` (*Identity* attribute), 24
`UnknownError`, 39
`use_active_time` (*ZoneAlarm* attribute), 13
`User` (class in *brainframe.api.bf_codecs.user_codecs*), 34
`User.Role` (class in *brainframe.api.bf_codecs.user_codecs*), 34
`username` (*User* attribute), 34
`UserNotFoundError`, 39

V

`VALID` (*LicenseInfo.State* attribute), 35
`value` (*Attribute* attribute), 17
`values()` (*NodeDescription.Size* class method), 27
`values()` (*PluginOption.Type* class method), 28
`values()` (*StreamConfiguration.ConnType* class method), 7
`values()` (*User.Role* class method), 34

`values()` (*ZoneAlarmCountCondition.TestType* class method), 14
`values()` (*ZoneAlarmRateCondition.DirectionType* class method), 15
`values()` (*ZoneAlarmRateCondition.TestType* class method), 15
`vector` (*Encoding* attribute), 24
`VectorTooLongError`, 39
`verified_as` (*Alert* attribute), 13
`version` (*Plugin* attribute), 27
`version()` (*BrainFrameAPI* method), 3

W

`wait_for_server_initialization()` (*BrainFrameAPI* method), 3
`WEBCAM` (*StreamConfiguration.ConnType* attribute), 7
`window_duration` (*ZoneAlarmCountCondition* attribute), 14
`window_threshold` (*ZoneAlarmCountCondition* attribute), 14
`with_attribute` (*ZoneAlarmCountCondition* attribute), 14
`with_attribute` (*ZoneAlarmRateCondition* attribute), 15
`with_class_name` (*ZoneAlarmCountCondition* attribute), 14
`with_class_name` (*ZoneAlarmRateCondition* attribute), 15
`with_identity` (*Detection* attribute), 18
`within` (*ZoneStatus* attribute), 18

Z

`Zone` (class in *brainframe.api.bf_codecs.zone_codecs*), 10
`zone` (*ZoneStatus* attribute), 19
`zone_id` (*Alert* attribute), 13
`zone_id` (*ZoneAlarm* attribute), 14
`ZoneAlarm` (class in *brainframe.api.bf_codecs.alarm_codecs*), 13
`ZoneAlarmCountCondition` (class in *brainframe.api.bf_codecs.condition_codecs*), 14
`ZoneAlarmCountCondition.TestType` (class in *brainframe.api.bf_codecs.condition_codecs*), 14
`ZoneAlarmNotFoundError`, 39
`ZoneAlarmRateCondition` (class in *brainframe.api.bf_codecs.condition_codecs*), 14
`ZoneAlarmRateCondition.DirectionType` (class in *brainframe.api.bf_codecs.condition_codecs*), 15
`ZoneAlarmRateCondition.TestType` (class in *brainframe.api.bf_codecs.condition_codecs*), 15
`ZoneNotDeletableError`, 39

ZoneNotFoundError, [39](#)
ZoneStatus (class in *brain-*
frame.api.bf_codecs.zone_codecs), [18](#)