pine

JHU/APL

Aug 04, 2021

CONTENTS:

CHAPTER ONE

API REFERENCE

This page contains auto-generated API reference documentation[1].

1.1 `pine`

1.1.1 Subpackages

`pine.backend`

Subpackages

`pine.backend.admin`

This module implements all methods required for user authentication when using PINE's db authentication rather than an external Auth service

Submodules

`pine.backend.admin.bp`

Module Contents

Functions

get_users()	Get the list of all users' details (id, email, and password hash)
get_user(user_id)	Given a user_id, return the user's details (id, email, and password hash)
update_user_password(user_id)	Change the password hash stored in the database for the given user to a newly calculated password hash derived from
update_user(user_id)	Change the details stored in the database for the given user to those provided in the json body of this request.

continues on next page

[1] Created with sphinx-autoapi

Table 1 – continued from previous page

add_user()	Add a new user to PINE, with the details provided in the json body of this request (id, email, and password hash).
delete_user(user_id)	Delete the user matching the given user_id
system_export()	Export the contents of the database as a zip file
system_import()	Import the contents of the data provided in the request body to the database
init_app(app)	

pine.backend.admin.bp.**bp**

pine.backend.admin.bp.**get_users**()
Get the list of all users' details (id, email, and password hash) :return: str

pine.backend.admin.bp.**get_user**(*user_id*)
Given a user_id, return the user's details (id, email, and password hash) :param user_id: str :return: str

pine.backend.admin.bp.**update_user_password**(*user_id*)
Change the password hash stored in the database for the given user to a newly calculated password hash derived from the password provided in the json body of this request. :param user_id: :return: Response

pine.backend.admin.bp.**update_user**(*user_id*)
Change the details stored in the database for the given user to those provided in the json body of this request. :param user_id: str :return: Response

pine.backend.admin.bp.**add_user**()
Add a new user to PINE, with the details provided in the json body of this request (id, email, and password hash). This method will calculate and store a password hash based upon the provided password :return: Response

pine.backend.admin.bp.**delete_user**(*user_id*)
Delete the user matching the given user_id :param user_id: str :return: Response

pine.backend.admin.bp.**system_export**()
Export the contents of the database as a zip file :return: Response

pine.backend.admin.bp.**system_import**()
Import the contents of the data provided in the request body to the database :return: Response

pine.backend.admin.bp.**init_app**(*app*)

pine.backend.annotations

This module contains the api methods required to perform and display annotations in the front-end and store the annotations in the backend

Submodules

pine.backend.annotations.bp

Module Contents

Functions

check_document_view_by_id(doc_id: str)	Verify that a document with the given doc_id exists and that the logged in user has permissions to access the
check_document_view(doc: dict)	
check_document_annotate(doc: dict)	
get_my_annotations_for_document(doc_id)	Get the list of annotations (key, start_index, end_index) produced by the logged in user for the document matching
get_others_annotations_for_document(doc_id)	Get the list of annotations (key, start_index, end_index) produced by all other users, not including the logged in
get_annotations_for_document(doc_id)	Get the list of annotations (key, start_index, end_index) produced by all users for the document matching the
get_current_annotation(doc_id, user_id)	Get all annotations of the provided document created by the given user.
check_overlapping_annotations(collection, ner_annotations)	
set_document_to_annotated_by_user(doc_id, user_id)	Modify the parameter in the database for the document signifying that the given user has annotated the given
_make_annotations(body)	
_add_or_update_annotation(new_annotation)	
save_annotations(doc_id)	Save new NER annotations and labels to the database as an entry for the logged in user, for the document. If there
save_collection_annotations(collection_id: str)	
init_app(app)	

pine.backend.annotations.bp.**logger**

pine.backend.annotations.bp.**CONFIG_ALLOW_OVERLAPPING_NER_ANNOTATIONS = allow_overlapping_ne**

pine.backend.annotations.bp.**bp**

pine.backend.annotations.bp.**check_document_view_by_id**(*doc_id: str*)

Verify that a document with the given doc_id exists and that the logged in user has permissions to access the document :param doc_id: str :return: dict

pine.backend.annotations.bp.**check_document_view**(*doc: dict*)

pine.backend.annotations.bp.**check_document_annotate**(*doc: dict*)

pine.backend.annotations.bp.**get_my_annotations_for_document**(*doc_id*)

Get the list of annotations (key, start_index, end_index) produced by the logged in user for the document matching the provided doc_id. :param doc_id: str :return: Response

pine.backend.annotations.bp.**get_others_annotations_for_document**(*doc_id*)

Get the list of annotations (key, start_index, end_index) produced by all other users, not including the logged in user for the document matching the provided doc_id. :param doc_id: str :return: str

pine.backend.annotations.bp.**get_annotations_for_document**(*doc_id*)

Get the list of annotations (key, start_index, end_index) produced by all users for the document matching the provided doc_id. :param doc_id: str :return: str

`pine.backend.annotations.bp.`**`get_current_annotation`**`(`*doc_id*, *user_id*`)`
Get all annotations of the provided document created by the given user. :param doc_id: str :param user_id: str :return: List

`pine.backend.annotations.bp.`**`check_overlapping_annotations`**`(`*collection*, *ner_annotations*`)`

`pine.backend.annotations.bp.`**`set_document_to_annotated_by_user`**`(`*doc_id*, *user_id*`)`
Modify the parameter in the database for the document signifying that the given user has annotated the given document :param doc_id: str :param user_id: str :return: whether the update succeeded :rtype: bool

`pine.backend.annotations.bp.`**`_make_annotations`**`(`*body*`)`

`pine.backend.annotations.bp.`**`_add_or_update_annotation`**`(`*new_annotation*`)`

`pine.backend.annotations.bp.`**`save_annotations`**`(`*doc_id*`)`
Save new NER annotations and labels to the database as an entry for the logged in user, for the document. If there are already annotations, use a patch request to update with the new annotations. If there are not, use a post request to create a new entry. :param doc_id: str :return: bool

`pine.backend.annotations.bp.`**`save_collection_annotations`**`(`*collection_id: str*`)`

`pine.backend.annotations.bp.`**`init_app`**`(`*app*`)`

`pine.backend.api`

This module implements all methods required for SwaggerUI to run on the backend of PINE.

Submodules

`pine.backend.api.bp`

Module Contents

Functions

`openapi_spec`()
`swagger_ui_index`()
`swagger_ui`(file: str)
`init_app`(app)

`pine.backend.api.bp.`**`bp`**

`pine.backend.api.bp.`**`LOGGER`**

`pine.backend.api.bp.`**`openapi_spec`**`()`

`pine.backend.api.bp.`**`swagger_ui_index`**`()`

`pine.backend.api.bp.`**`swagger_ui`**`(`*file: str*`)`

`pine.backend.api.bp.`**`init_app`**`(`*app*`)`

`pine.backend.auth`

Submodules

`pine.backend.auth.bp`

Module Contents

Classes

AuthModule

Functions

is_flat()
get_logged_in_user()
login_required(view)
admin_required(view)
flask_get_module()
flask_get_flat() → flask.Response
flask_get_can_manage_users() → flask.Response
flask_get_logged_in_user() → flask.Response
flask_get_logged_in_user_details() → flask.Response
flask_get_login_form() → flask.Response
flask_post_logout() → flask.Response
init_app(app)

pine.backend.auth.bp.**CONFIG_AUTH_MODULE_KEY = AUTH_MODULE**

pine.backend.auth.bp.**bp**

pine.backend.auth.bp.**module**

pine.backend.auth.bp.**is_flat**()

pine.backend.auth.bp.**get_logged_in_user**()

pine.backend.auth.bp.**login_required**(*view*) `pine.backend.auth.bp.`**`admin_required`**`(`*view*`)`

`pine.backend.auth.bp.`**`flask_get_module`**`()`

`pine.backend.auth.bp.`**`flask_get_flat`**`()` → flask.Response

`pine.backend.auth.bp.`**`flask_get_can_manage_users`**`()` → flask.Response

`pine.backend.auth.bp.`**`flask_get_logged_in_user`**`()` → flask.Response

`pine.backend.auth.bp.`**`flask_get_logged_in_user_details`**`()` → flask.Response

`pine.backend.auth.bp.`**`flask_get_login_form`**`()` → flask.Response

`pine.backend.auth.bp.`**`flask_post_logout`**`()` → flask.Response

`class` `pine.backend.auth.bp.`**`AuthModule`**`(`*app*, *bp*`)`

Bases: `object`

`__metaclass__`

`abstract is_flat``(`*self*`)` → bool

`abstract can_manage_users``(`*self*`)` → bool

`abstract get_login_form``(`*self*`)` → *pine.backend.models.LoginForm*

`get_logged_in_user``(`*self*`)`

`get_logged_in_user_details``(`*self*`)` → *pine.backend.models.UserDetails*

`logout``(`*self*`)`

`pine.backend.auth.bp.`**`init_app`**`(`*app*`)`

pine.backend.auth.eve

Module Contents

Classes

EveUser
EveModule

`class` `pine.backend.auth.eve.`**`EveUser`**`(`*data*`)`

Bases: *`pine.backend.models.AuthUser`*

`property id``(`*self*`)`

`property username``(`*self*`)`

`property display_name``(`*self*`)`

`property is_admin``(`*self*`)`

`get_details``(`*self*`)` → *pine.backend.models.UserDetails*

`class` `pine.backend.auth.eve.`**`EveModule`**`(`*app*, *bp*`)`

Bases: *`pine.backend.auth.bp.AuthModule`*

`is_flat``(`*self*`)` → bool

can_manage_users(*self*) → bool

get_logged_in_user_details(*self*)

update_user_details(*self*)

update_user_password(*self*)

get_login_form(*self*) → *pine.backend.models.LoginForm*

login(*self*) → flask.Response

get_all_users(*self*)

`pine.backend.auth.oauth`

Module Contents

Classes

OAuthUser
OAuthModule

`pine.backend.auth.oauth.`**LOGGER**

class `pine.backend.auth.oauth.`**OAuthUser**(*data*, *id_field*, *display_field=None*)
Bases: *pine.backend.models.AuthUser*

property id(*self*)

property username(*self*)

property display_name(*self*)

property is_admin(*self*)

class `pine.backend.auth.oauth.`**OAuthModule**(*app*, *bp*, *secret*)
Bases: *pine.backend.auth.bp.AuthModule*

abstract register_oauth(*self*, *oauth*, *app*)

abstract get_login_form_button_text(*self*)

is_flat(*self*) → bool

can_manage_users(*self*) → bool

get_login_form(*self*) → *pine.backend.models.LoginForm*

make_user(*self*, *decoded*: dict) → *pine.backend.auth.oauth.OAuthUser*

login(*self*) → flask.Response

_authorize(*self*, *authorization_response*)

authorize_post(*self*)

authorize_get(*self*)

`pine.backend.auth.password`

Module Contents

Functions

hash_password(password: str) → str	
check_password(password: str, hashed_password: str)	

pine.backend.auth.password.**hash_password**(*password: str*) → str

pine.backend.auth.password.**check_password**(*password: str*, *hashed_password: str*)

`pine.backend.auth.vegas`

Module Contents

Classes

VegasAuthModule	

class pine.backend.auth.vegas.**VegasAuthModule**(*app*, *bp*)

Bases: *pine.backend.auth.oauth.OAuthModule*

register_oauth(*self*, *oauth*, *app*)

get_login_form_button_text(*self*)

make_user(*self*, *decoded: dict*) → *pine.backend.auth.oauth.OAuthUser*

Package Contents

Functions

login_required(view)	
admin_required(view)	
get_logged_in_user()	
is_flat()	

pine.backend.auth.**login_required**(*view*)

pine.backend.auth.**admin_required**(*view*) `pine.backend.auth.module`

`pine.backend.auth.get_logged_in_user()`

`pine.backend.auth.is_flat()`

`pine.backend.collections`

This module contains the api methods required to interact with, organize, create, and display collections in the front-end and store the collections in the backend

Submodules

`pine.backend.collections.bp`

Module Contents

Functions

is_cached_last_collection(collection_id)	
update_cached_last_collection(collection_id)	
user_permissions_projection()	
get_user_permissions(collection: dict) → pine.backend.models.CollectionUserPermissions	
get_user_permissions_by_id(collection_id: str) → pine.backend.models.CollectionUserPermissions	
get_user_permissions_by_ids(collection_ids: Iterable[str]) → List[models.CollectionUserPermissions]	
get_user_collections(archived, page)	Return collections for the logged in user using pagination. Returns all collections if parameter "page" is "all",
get_unarchived_user_collections(page)	Return unarchived user collections for the corresponding page value. Default value returns collections for all
get_archived_user_collections(page)	Return archived user collections for the corresponding page value. Default value returns collections for all
archive_or_unarchive_collection(collection_id, archive)	Set the "archived" boolean flag for the collection matching the provided collection_id.
archive_collection(collection_id)	Archive the collection matching the provided collection id
unarchive_collection(collection_id)	Unarchive the collection matching the provided collection id
get_collection(collection_id)	Return the collection object for the collection matching the provided collection id. This object has the fields:
download_collection(collection_id)	
add_annotator_to_collection(collection_id)	

continues on next page

Table 11 – continued from previous page

add_viewer_to_collection(collection_id)	
add_label_to_collection(collection_id)	
get_overlap_ids(collection_id: str)	Return the list of ids for overlapping documents for the collection matching the provided collection id.
_upload_documents(collection, docs)	
create_collection()	Create a new collection based upon the entries provided in the POST request's associated form fields.
_check_collection_and_get_image_dir(collection_id, path)	
get_static_collection_images(collection_id)	
get_collection_images(collection_id)	
get_collection_image(collection_id, path)	
get_collection_image_exists(collection_id, path)	
_path_split(path)	
_safe_path(path)	
endpoint_get_user_permissions(collection_id)	
_upload_collection_image_file(collection_id, path, image_file)	
post_collection_image(collection_id, path)	
init_app(app)	

pine.backend.collections.bp.**bp**

pine.backend.collections.bp.**LOGGER**

pine.backend.collections.bp.**DOCUMENTS_PER_TRANSACTION = 500**

pine.backend.collections.bp.**LAST_COLLECTION_FOR_IMAGE**

pine.backend.collections.bp.**is_cached_last_collection**(*collection_id*)

pine.backend.collections.bp.**update_cached_last_collection**(*collection_id*)

pine.backend.collections.bp.**user_permissions_projection**()

pine.backend.collections.bp.**get_user_permissions**(*collection: dict*) → *pine.backend.models.CollectionUserPermissions*

pine.backend.collections.bp.**get_user_permissions_by_id**(*collection_id: str*) → *pine.backend.models.CollectionUserPermissions*

pine.backend.collections.bp.**get_user_permissions_by_ids**(*collection_ids: Iterable[str]*) → List[models.CollectionUserPermissions]

`pine.backend.collections.bp.`**`get_user_collections`**`(`*archived*, *page*`)`
Return collections for the logged in user using pagination. Returns all collections if parameter “page” is “all”, or the collections associated with the given page. Can return archived or un-archived collections based upon the “archived” flag. :param archived: Bool :param page: str :return: Response

`pine.backend.collections.bp.`**`get_unarchived_user_collections`**`(`*page*`)`
Return unarchived user collections for the corresponding page value. Default value returns collections for all pages. :param page: str :return: Response

`pine.backend.collections.bp.`**`get_archived_user_collections`**`(`*page*`)`
Return archived user collections for the corresponding page value. Default value returns collections for all pages. :param page: str :return: Response

`pine.backend.collections.bp.`**`archive_or_unarchive_collection`**`(`*collection_id*, *archive*`)`
Set the “archived” boolean flag for the collection matching the provided collection_id. :param collection_id: str :param archive: Bool :return: Response

`pine.backend.collections.bp.`**`archive_collection`**`(`*collection_id*`)`
Archive the collection matching the provided collection id :param collection_id: str :return: Response

`pine.backend.collections.bp.`**`unarchive_collection`**`(`*collection_id*`)`
Unarchive the collection matching the provided collection id :param collection_id: str :return: Response

`pine.backend.collections.bp.`**`get_collection`**`(`*collection_id*`)`
Return the collection object for the collection matching the provided collection id. This object has the fields: ‘creator_id’, ‘annotators’, ‘viewers’, ‘labels’, ‘metadata’, ‘archived’, and ‘configuration’. :param collection_id: str :return: Response

`pine.backend.collections.bp.`**`download_collection`**`(`*collection_id*`)`

`pine.backend.collections.bp.`**`add_annotator_to_collection`**`(`*collection_id*`)`

`pine.backend.collections.bp.`**`add_viewer_to_collection`**`(`*collection_id*`)`

`pine.backend.collections.bp.`**`add_label_to_collection`**`(`*collection_id*`)`

`pine.backend.collections.bp.`**`get_overlap_ids`**`(`*collection_id: str*`)`
Return the list of ids for overlapping documents for the collection matching the provided collection id. :param collection_id: str :return: tuple

`pine.backend.collections.bp.`**`_upload_documents`**`(`*collection*, *docs*`)`

`pine.backend.collections.bp.`**`create_collection`**`()`
Create a new collection based upon the entries provided in the POST request’s associated form fields. These fields include: collection - collection name overlap - ratio of overlapping documents. (0-1) with 0 being no overlap and 1 being every document has overlap, ex:

> .90 - 90% of documents overlap

train_every - automatically train a new classifier after this many documents have been annotated pipelineId - the id value of the classifier pipeline associated with this collection (spacy, opennlp, corenlp) classifierParameters - optional parameters that adjust the configuration of the chosen classifier pipeline. archived - whether or not this collection should be archived. A collection can be created with documents listed in a csv file. Each new line in the csv represents a new document. The data of this csv can be passed to this method through the POST request’s FILES field “file”. used when creating a collection based on an uploaded csv file:

> csvTextCol - column of csv containing the text of the documents (default: 0) csvHasHeader - boolean for whether or not the csv file has a header row (default: False)

A collection can also be created with a number of images through FILES fields “imageFileN” where N is an (ignored) index :return: information about the created collection pine.backend.collections.bp.**_check_collection_and_get_image_dir**(*collection_id, path*)

pine.backend.collections.bp.**get_static_collection_images**(*collection_id*)

pine.backend.collections.bp.**get_collection_images**(*collection_id*)

pine.backend.collections.bp.**get_collection_image**(*collection_id, path*)

pine.backend.collections.bp.**get_collection_image_exists**(*collection_id, path*)

pine.backend.collections.bp.**_path_split**(*path*)

pine.backend.collections.bp.**_safe_path**(*path*)

pine.backend.collections.bp.**endpoint_get_user_permissions**(*collection_id*)

pine.backend.collections.bp.**_upload_collection_image_file**(*collection_id, path, image_file*)

pine.backend.collections.bp.**post_collection_image**(*collection_id, path*)

pine.backend.collections.bp.**init_app**(*app*)

Package Contents

Functions

user_permissions_projection()	
get_user_permissions(collection: dict) → pine.backend.models.CollectionUserPermissions	
get_user_permissions_by_id(collection_id: str) → pine.backend.models.CollectionUserPermissions	
get_user_permissions_by_ids(collection_ids: Iterable[str]) → List[models.CollectionUserPermissions]	
get_overlap_ids(collection_id: str)	Return the list of ids for overlapping documents for the collection matching the provided collection id.

pine.backend.collections.**user_permissions_projection**()

pine.backend.collections.**get_user_permissions**(*collection: dict*) → *pine.backend.models.CollectionUserPermissions*

pine.backend.collections.**get_user_permissions_by_id**(*collection_id: str*) → *pine.backend.models.CollectionUserPermissions*

pine.backend.collections.**get_user_permissions_by_ids**(*collection_ids: Iterable[str]*) → List[models.CollectionUserPermissions]

pine.backend.collections.**get_overlap_ids**(*collection_id: str*)

Return the list of ids for overlapping documents for the collection matching the provided collection id. :param collection_id: str :return: tuple

`pine.backend.data`

Submodules

`pine.backend.data.bp`

Module Contents

Functions

init_app(app)	

`pine.backend.data.bp.`**`init_app`**`(`*app*`)`

`pine.backend.data.service`

Module Contents

Classes

PerformanceHistory	

Functions

_standardize_path(path: PATH_TYPE, *additional_paths: List[str]) → List[str]	
url(path: PATH_TYPE, *additional_paths: List[str]) → str	Returns a complete URL for the given eve-relative path(s).
where_params(where: dict) → dict	Returns a “where” parameters object that can be passed to eve.
params(params: dict) → dict	Returns a parameters object that can be passed to eve.
get(path: PATH_TYPE, **kwargs: dict) → requests.Response	Wraps requests.get for the given eve-relative path.
post(path: PATH_TYPE, **kwargs: dict) → requests.Response	Wraps requests.post for the given eve-relative path.
put(path: PATH_TYPE, **kwargs: dict) → requests.Response	Wraps requests.put for the given eve-relative path.
delete(path: PATH_TYPE, **kwargs: dict) → requests.Response	Wraps requests.delete for the given eve-relative path.
patch(path: PATH_TYPE, **kwargs: dict) → requests.Response	Wraps requests.patch for the given eve-relative path.
get_item_by_id(path: PATH_TYPE, item_id: str, params: dict = { }) → dict	Gets a single item by the given ID.

continues on next page

Table 15 – continued from previous page

`get_all_versions_of_item_by_id`(path: PATH_TYPE, item_id: str, params: dict = {}) → List[dict]	Gets all versions of an item by the given ID.
`get_all`(path: PATH_TYPE, params={}) → dict	Returns ALL database items, using pagination if needed. This returns the "normal" eve
`get_all_items`(path: PATH_TYPE, params={}) → List[dict]	Returns ALL database items, using pagination if needed.
`convert_response`(requests_response: requests.Response) → flask.Response	Converts a requests response to a flask response.
`remove_eve_fields`(obj: dict, remove_timestamps: bool = True, remove_versions: bool = True) → None	Removes the fields that eve adds that aren't necessarily relevant to the data. The object
`remove_nonupdatable_fields`(obj: dict) → None	Removes the non-updatable fields in the given eve object. This is currently equivalent to

pine.backend.data.service.**logger**

pine.backend.data.service.**PATH_TYPE**

Type for paths that can be passed into these messages. Either a single string, or a list-like type of strings that is combined with a '/'.

class pine.backend.data.service.**PerformanceHistory**

Bases: object

pformat (*self*, ***kwargs*)

pprint (*self*)

add (*self*, *rest_type: str*, *path: str*, *response*)

pine.backend.data.service.**PERFORMANCE_HISTORY**

pine.backend.data.service.**_standardize_path** (*path:* PATH_TYPE, **additional_paths: List[str]*) → List[str]

pine.backend.data.service.**url** (*path:* PATH_TYPE, **additional_paths: List[str]*) → str

Returns a complete URL for the given eve-relative path(s).

Parameters

- **path** – str: eve-relative path (e.g. "collections" or ["collections", id])
- **additional_paths** – str[]: any additional paths to append

Returns url

Return type str

pine.backend.data.service.**where_params** (*where: dict*) → dict

Returns a "where" parameters object that can be passed to eve.

Eve requires that dict parameters be serialized as JSON.

Parameters **where** – dict: dictionary of "where" params to pass to eve

Returns "where" params in eve-appropriate format

Return type dict

pine.backend.data.service.**params** (*params: dict*) → dict

Returns a parameters object that can be passed to eve.

Eve requires that dict parameters be serialized as JSON.

Parameters **where** – dict: dictionary of "where" params to pass to eve

Returns params in eve-appropriate format

Return type dict

`pine.backend.data.service.get`(*path:* PATH_TYPE, ***kwargs: dict*) → requests.Response
Wraps requests.get for the given eve-relative path.

Parameters

- **path** – list[str]|str: eve-relative path (e.g. ["collections", id] or "/collections")
- ****kwargs** – dict: any additional arguments to pass to requests.get

Returns server response

Return type requests.Response

`pine.backend.data.service.post`(*path:* PATH_TYPE, ***kwargs: dict*) → requests.Response
Wraps requests.post for the given eve-relative path.

Parameters

- **path** – list[str]|str: eve-relative path (e.g. ["collections", id] or "/collections")
- ****kwargs** – dict: any additional arguments to pass to requests.post

Returns server response

Return type requests.Response

`pine.backend.data.service.put`(*path:* PATH_TYPE, ***kwargs: dict*) → requests.Response
Wraps requests.put for the given eve-relative path.

Parameters

- **path** – list[str]|str: eve-relative path (e.g. ["collections", id] or "/collections")
- ****kwargs** – dict: any additional arguments to pass to requests.put

Returns server response

Return type requests.Response

`pine.backend.data.service.delete`(*path:* PATH_TYPE, ***kwargs: dict*) → requests.Response
Wraps requests.delete for the given eve-relative path.

Parameters

- **path** – list[str]|str: eve-relative path (e.g. ["collections", id] or "/collections")
- ****kwargs** – dict: any additional arguments to pass to requests.delete

Returns server response

Return type requests.Response

`pine.backend.data.service.patch`(*path:* PATH_TYPE, ***kwargs: dict*) → requests.Response
Wraps requests.patch for the given eve-relative path.

Parameters

- **path** – list[str]|str: eve-relative path (e.g. ["collections", id] or "/collections")
- ****kwargs** – dict: any additional arguments to pass to requests.patch

Returns server response

Return type requests.Response

`pine.backend.data.service.`**`get_item_by_id`**(*path:* PATH_TYPE, *item_id:* *str*, *params:* *dict* = *{}*) → dict

Gets a single item by the given ID.

Parameters

- **`path`** – list[str]|str: eve-relative path (e.g. [“collections”, id] or “/collections”)
- **`item_id`** – str: item ID
- **`params`** – dict: optional additional parameters to send in with GET

Returns the item as a dict

Return type dict

`pine.backend.data.service.`**`get_all_versions_of_item_by_id`**(*path:* PATH_TYPE, *item_id:* *str*, *params:* *dict* = *{}*) → List[dict]

Gets all versions of an item by the given ID.

Parameters

- **`path`** – list[str]|str: eve-relative path (e.g. [“collections”, id] or “/collections”)
- **`item_id`** – str: item ID
- **`params`** – dict: optional additional arguments to send in with GET

Returns the items as a list of dicts

Return type list[dict]

`pine.backend.data.service.`**`get_all`**(*path:* PATH_TYPE, *params={}*) → dict

Returns ALL database items, using pagination if needed. This returns the “normal” eve JSON with “_items”, “_meta”, etc.

Parameters

- **`path`** – list[str]|str: eve-relative path (e.g. [“collections”, id] or “/collections”)
- **`params`** – dict: optional additional parameters to send in with GET

Returns an eve collections dict with, e.g., _items

Return type dict

`pine.backend.data.service.`**`get_all_items`**(*path:* PATH_TYPE, *params={}*) → List[dict]

Returns ALL database items, using pagination if needed.

Parameters

- **`path`** – list[str]|str: eve-relative path (e.g. [“collections”, id] or “/collections”)
- **`params`** – dict: optional additional parameters to send in with GET

Returns the items as a list of dicts

Return type list[dict]

`pine.backend.data.service.`**`convert_response`**(*requests_response:* *requests.Response*) → flask.Response

Converts a requests response to a flask response.

Parameters **`requests_response`** – requests.Response: response from requests library

Returns a flask response

Return type flask.Response

pine.backend.data.service.**remove_eve_fields**(*obj: dict*, *remove_timestamps: bool = True*, *remove_versions: bool = True*) → None

Removes the fields that eve adds that aren't necessarily relevant to the data. The object that is passed in is modified in-place.

This currently includes: *_etag*, *_links*, *_created* (if *remove_timestamps*), *_updated* (if *remove_timestamps*), *_version* (if *remove_versions*), and *_latest_version* (if *remove_versions*).

Parameters

- **`obj`** – dict: the object to modify
- **`remove_timestamps`** – bool: whether to remove timestamp fields (defaults to *True*)
- **`remove_versions`** – bool: whether to remove version fields (defaults to *True*)

pine.backend.data.service.**remove_nonupdatable_fields**(*obj: dict*) → None

Removes the non-updatable fields in the given eve object. This is currently equivalent to calling . . . with all the default options.

Parameters **`obj`** – dict: the object to modify

`pine.backend.data.users`

Module Contents

Functions

get_all_users()	
get_user(user_id)	
get_user_by_email(email)	
get_user_by_id_or_email(username)	
get_user_details(user_id)	
update_user(user_id: str, details: pine.backend.models.UserDetails)	
print_users_command()	
add_admin_command(user_id, username, password)	
set_user_password_by_id(user_id, password)	
set_user_password(username, password)	
reset_user_passwords()	

pine.backend.data.users.**get_all_users**()

pine.backend.data.users.**get_user**(*user_id*)

pine.backend.data.users.**get_user_by_email**(*email*) `pine.backend.data.users.`**`get_user_by_id_or_email`**`(`*username*`)`

`pine.backend.data.users.`**`get_user_details`**`(`*user_id*`)`

`pine.backend.data.users.`**`update_user`**`(`*user_id: str, details:* pine.backend.models.UserDetails`)`

`pine.backend.data.users.`**`print_users_command`**`()`

`pine.backend.data.users.`**`add_admin_command`**`(`*user_id, username, password*`)`

`pine.backend.data.users.`**`set_user_password_by_id`**`(`*user_id, password*`)`

`pine.backend.data.users.`**`set_user_password`**`(`*username, password*`)`

`pine.backend.data.users.`**`reset_user_passwords`**`()`

pine.backend.documents

Submodules

pine.backend.documents.bp

Module Contents

Functions

_document_user_can_projection()	
get_collection_ids_for(document_ids) → List[str]	
get_user_permissions(document: dict) → pine.backend.models.CollectionUserPermissions	
get_user_permissions_by_id(document_id: str) → pine.backend.models.CollectionUserPermissions	
get_user_permissions_by_ids(document_ids: Iterable[str]) → List[models.CollectionUserPermissions]	
get_document(doc_id)	
count_documents_in_collection(col_id)	
get_all_documents_in_collection(col_id)	
get_paginated_documents_in_collection(collection_id)	
_check_documents(documents) → dict	
_get_collection_classifiers(collection_ids)	
_get_classifier_next_instances(collection_classifiers: dict)	
add_document()	

continues on next page

Table 17 – continued from previous page

endpoint_get_user_permissions(doc_id)	
update_metadata(doc_id)	
_delete_documents_by_id(doc_ids: List[str]) → bool	
delete_document(doc_id: str)	
delete_documents()	
init_app(app)	

pine.backend.documents.bp.**bp**

pine.backend.documents.bp.**_document_user_can_projection**()

pine.backend.documents.bp.**get_collection_ids_for**(*document_ids*) → List[str]

pine.backend.documents.bp.**get_user_permissions**(*document: dict*) → *pine.backend.models.CollectionUserPermissions*

pine.backend.documents.bp.**get_user_permissions_by_id**(*document_id: str*) → *pine.backend.models.CollectionUserPermissions*

pine.backend.documents.bp.**get_user_permissions_by_ids**(*document_ids: Iterable[str]*) → List[models.CollectionUserPermissions]

pine.backend.documents.bp.**get_document**(*doc_id*)

pine.backend.documents.bp.**count_documents_in_collection**(*col_id*)

pine.backend.documents.bp.**get_all_documents_in_collection**(*col_id*)

pine.backend.documents.bp.**get_paginated_documents_in_collection**(*collection_id*)

pine.backend.documents.bp.**_check_documents**(*documents*) → dict

pine.backend.documents.bp.**_get_collection_classifiers**(*collection_ids*)

pine.backend.documents.bp.**_get_classifier_next_instances**(*collection_classifiers: dict*)

pine.backend.documents.bp.**add_document**()

pine.backend.documents.bp.**endpoint_get_user_permissions**(*doc_id*)

pine.backend.documents.bp.**update_metadata**(*doc_id*)

pine.backend.documents.bp.**_delete_documents_by_id**(*doc_ids: List[str]*) → bool

pine.backend.documents.bp.**delete_document**(*doc_id: str*)

pine.backend.documents.bp.**delete_documents**()

pine.backend.documents.bp.**init_app**(*app*)

Package Contents

Functions

get_collection_ids_for(document_ids) → List[str]	
get_user_permissions(document: dict) → pine.backend.models.CollectionUserPermissions	
get_user_permissions_by_id(document_id: str) → pine.backend.models.CollectionUserPermissions	
get_user_permissions_by_ids(document_ids: Iterable[str]) → List[models.CollectionUserPermissions]	

`pine.backend.documents.`**`get_collection_ids_for`**(*document_ids*) → List[str]

`pine.backend.documents.`**`get_user_permissions`**(*document: dict*) → *pine.backend.models.CollectionUserPermissions*

`pine.backend.documents.`**`get_user_permissions_by_id`**(*document_id: str*) → *pine.backend.models.CollectionUserPermissions*

`pine.backend.documents.`**`get_user_permissions_by_ids`**(*document_ids: Iterable[str]*) → List[models.CollectionUserPermissions]

`pine.backend.job_manager`

Submodules

`pine.backend.job_manager.service`

Module Contents

Classes

ServiceJob	Data class for a service job.
ServiceManager	**type default_handler** callable

`pine.backend.job_manager.service.`**`config`**

`pine.backend.job_manager.service.`**`logger`**

`class` `pine.backend.job_manager.service.`**`ServiceJob`**(*job_id: str, request_body: dict, request_response: dict*)

Bases: `object`

Data class for a service job.

“Constructor.

Parameters

- **`job_id`** – str: job ID

- **request_body** – dict: job request body
- **request_response** – dict: job request response

class pine.backend.job_manager.service.**ServiceManager**(*default_handler=None*)
Bases: object

r_pool

r_conn

redis_key_prefix

redis_reg_key_prefix

redis_channels_key

redis_channel_ttl_key_prefix

redis_work_queue_key_prefix

redis_work_mutex_key_prefix

redis_handler_mutex_key_prefix

redis_reg_key_ttl

redis_channels_key_ttl

redis_work_queue_key_ttl

redis_work_mutex_key_ttl

redis_handler_mutex_key_ttl

handler_timeout

registration_worker_name = registration_worker

processing_worker_name = processing_worker

channel_worker_name = channel_worker

shutdown_channel = shutdown

registration_channel = registration

reserved_channels

classmethod get_results_key(*cls*, *service_name: str*, *job_id: str*) → str

classmethod get_running_jobs_key(*cls*, *service_name: str*) → str

classmethod get_registered_channels(*cls*, *include_ttl=False*)
Get list of registered channels, with registration time if requested. :type include_ttl: bool :rtype: list[str] | dict[str, datetime]

classmethod get_registered_service_details(*cls*, *service_name=None*)
Get registration details of a service. :type service_name: str :rtype: None | dict

classmethod get_registered_services(*cls*, *include_details=True*)
Get list of registered services and registration body if requested. :type include_details: bool :rtype: list[str] | list[dict]

classmethod _get_service_details(*cls*, *service_name: str*, *retry_count=10*) → dict

classmethod _get_service_channel(*cls*, *service_name: str*) → str

classmethod send_service_request(*cls*, *service_name: str*, *data*, *job_id=None*, *encoder=None*)
Queue's a job for the requested service. :type service_name: str :type data: dict :type job_id: str :type encoder: json.JSONEncoder :rtype: None | dict

classmethod get_job_response(*cls*, *service_name: str*, *job_id: str*, *timeout_in_s: int*)
Waits for a response for the given job and returns it. :param service_name: str: service name :param job_id: str: job ID :param timeout_in_s: int: wait timeout in seconds :rtype None | dict

classmethod send_service_request_and_get_response(*cls*, *service_name: str*, *data*, *timeout_in_s: int*, *job_id=None*, *encoder=None*) → *pine.backend.job_manager.service.ServiceJob*
Sends a service requests, waits for a response, and returns job data. :param service_name: str: service name :param data: job data :param timeout_in_s: int: wait timeout in seconds :param job_id: str: optional job ID (or None to auto-generate one) :param encoder: optional JSON encoder for job data :rtype None | ServiceJob

classmethod get_running_jobs(*cls*, *service_name: str*) → List[str]
Returns running jobs. :param service_name: str: service name :rtype list[str]

start_listeners(*self*)
Starts all the workers.

stop_listeners(*self*)
Stops all the workers.

_start_registration_listener(*self*)
Starts the registration worker. :rtype: bool

_start_processing_listeners(*self*)
Starts the processing workers. :rtype: bool

_start_channel_watchdog(*self*)
Starts the channel watchdog workers in an asyncio-only thread. It monitors the channel TTL's. :rtype: bool

_stop_channel_watchdog(*self*)
Stops the channel watchdog workers in an asyncio-only thread. :rtype: bool

_registration_listener(*self*)
Registration Listener Implementation.

async _channel_watchdog(*self*)
Channel Watchdog Implementation. In asyncio, it monitors the channel-ttl keys and the channels SET to expire registered services as needed. The other functionality is to register new channels as they're added to the pubsub in the processing listener.

static _thread_killer(*thread_id*)

_processing_listener_handler_wrapper(*self*, *job_id*, *job_type*, *job_queue*, *job_data*)

_processing_listener(*self*)
Processing Listener Implementation. Runs a handler when a processing message gets send over an already registered channel.

pine.backend.job_manager.service.**sm**

`pine.backend.pineiaa`

Subpackages

`pine.backend.pineiaa.bratiaa`

Submodules

`pine.backend.pineiaa.bratiaa.agree`

Module Contents

Classes

Document	
F1Agreement	

Functions

input_generator(json_list)	
compute_f1(tp, fp, fn)	
compute_f1_agreement(annotators, documents, labels, token_func=None, eval_func=None)	
iaa_report(f1_agreement, precision=3)	

pine.backend.pineiaa.bratiaa.agree.**Annotation**

pine.backend.pineiaa.bratiaa.agree.**AnnFile**

pine.backend.pineiaa.bratiaa.agree.**LOGGER**

class pine.backend.pineiaa.bratiaa.agree.**Document** (*txt*, *doc_id*)

__slots__ = ['ann_files', 'txt', 'doc_id']

pine.backend.pineiaa.bratiaa.agree.**input_generator** (*json_list*)

pine.backend.pineiaa.bratiaa.agree.**compute_f1** (*tp*, *fp*, *fn*)

class pine.backend.pineiaa.bratiaa.agree.**F1Agreement** (*annotators*, *documents*, *labels*, *eval_func=exact_match_instance_evaluation*, *token_func=None*)

property annotators (*self*)

property documents (*self*)

property labels(*self*)

_compute_tp_fp_fn(*self*, *documents*)

_increment_counts(*self*, *annotations*, *pair*, *doc*, *kind*)

mean_sd_per_label(*self*)
Mean and standard deviation of all annotator combinations' F1 scores by label.

mean_sd_per_document(*self*)
Mean and standard deviation of all annotator combinations' F1 scores per document.

mean_sd_total(*self*)
Mean and standard deviation of all annotator cominations' F1 scores.

mean_sd_per_label_one_vs_rest(*self*, *annotator*)
Mean and standard deviation of all annotator combinations' F1 scores involving given annotator per label.

mean_sd_total_one_vs_rest(*self*, *annotator*)
Mean and standard deviation of all annotator combinations' F1 scores involving given annotator.

_pairs_involving(*self*, *annotator*)

static _mean_sd(*f1_pairs*)
Mean and standard deviation along first axis.

static print_table(*row_label_header*, *row_labels*, *avg*, *stddev*, *precision=3*)

compute_total_f1_matrix(*self*)
Returns (n x n) matrix, where n is the number of annotators, containing pair-wise total F1 scores between all annotators.

By definition, the matrix is symmetric and F1 = 1 on the main diagonal.

draw_heatmap(*self*, *out_path*)
Draws heatmap based on square matrix of F1 scores.

pine.backend.pineiaa.bratiaa.agree.**compute_f1_agreement**(*annotators*, *documents*, *labels*, *token_func=None*, *eval_func=None*)

pine.backend.pineiaa.bratiaa.agree.**iaa_report**(*f1_agreement*, *precision=3*)

pine.backend.pineiaa.bratiaa.agree_cli

Module Contents

Functions

parse_args()	
main()	

pine.backend.pineiaa.bratiaa.agree_cli.**parse_args**()

pine.backend.pineiaa.bratiaa.agree_cli.**main**()

pine.backend.pineiaa.bratiaa.evaluation

Functions for computing the difference between two sets of annotations.

Module Contents

Functions

exact_match_instance_evaluation(ann_list_1, ann_list2, tokens=None)	
exact_match_token_evaluation(ann_list_1, ann_list_2, tokens=None)	Annotations are split into token-sized bits before true positives, false positives and false negatives are computed.
counter2list(c)	
_read_token_annotations(ann_list, tokens)	Yields a new annotation for each token overlapping with an annotation. If annotations are overlapping each other,

pine.backend.pineiaa.bratiaa.evaluation.**Annotation**

pine.backend.pineiaa.bratiaa.evaluation.**exact_match_instance_evaluation**(*ann_list_1*, *ann_list2*, *tokens=None*)

pine.backend.pineiaa.bratiaa.evaluation.**exact_match_token_evaluation**(*ann_list_1*, *ann_list_2*, *tokens=None*)

Annotations are split into token-sized bits before true positives, false positives and false negatives are computed.

Sub-token annotations are expanded to full tokens. Long annotations will influence the results more than short annotations. Boundary errors for adjacent annotations with the same label are ignored!

pine.backend.pineiaa.bratiaa.evaluation.**counter2list**(*c*)

pine.backend.pineiaa.bratiaa.evaluation.**_read_token_annotations**(*ann_list*, *tokens*)

Yields a new annotation for each token overlapping with an annotation. If annotations are overlapping each other, there will be multiple annotations for a single token.

pine.backend.pineiaa.bratiaa.iaa_service

Module Contents

Functions

get_items(resource)	
get_all_items(query)	

continues on next page

Table 24 – continued from previous page

get_doc_annotations(collection_id, exclude=None)	
fix_num_for_json(number)	
getIAAReportForCollection(collection_id)	

pine.backend.pineiaa.bratiaa.iaa_service.**EVE_HEADERS**

pine.backend.pineiaa.bratiaa.iaa_service.**get_items**(*resource*)

pine.backend.pineiaa.bratiaa.iaa_service.**get_all_items**(*query*)

pine.backend.pineiaa.bratiaa.iaa_service.**get_doc_annotations**(*collection_id, exclude=None*)

pine.backend.pineiaa.bratiaa.iaa_service.**fix_num_for_json**(*number*)

pine.backend.pineiaa.bratiaa.iaa_service.**getIAAReportForCollection**(*collection_id*)

pine.backend.pineiaa.bratiaa.utils

Module Contents

Classes

TokenOverlap	Data structure for quick lookup of tokens overlapping with given span.

Functions

tokenize(text)	
read(path, encoding=ENCODING, newline='\r\n', mode='r')	

pine.backend.pineiaa.bratiaa.utils.**ENCODING = utf-8**

pine.backend.pineiaa.bratiaa.utils.**TOKEN**

pine.backend.pineiaa.bratiaa.utils.**tokenize**(*text*)

pine.backend.pineiaa.bratiaa.utils.**read**(*path, encoding=ENCODING, newline='\r\n', mode='r'*)

class pine.backend.pineiaa.bratiaa.utils.**TokenOverlap**(*text, tokens*)

Data structure for quick lookup of tokens overlapping with given span. Assumes that the provided list of tokens is sorted by indices!

static compute_mapping(*text_length, tokens*)

overlapping_tokens(*self, start, end*)

Package Contents

Classes

`F1Agreement`	
`Document`	

Functions

`compute_f1_agreement`(annotators, documents, labels, token_func=None, eval_func=None)	
`iaa_report`(f1_agreement, precision=3)	
`input_generator`(json_list)	
`exact_match_instance_evaluation`(ann_list_1, ann_list2, tokens=None)	
`exact_match_token_evaluation`(ann_list_1, ann_list_2, tokens=None)	Annotations are split into token-sized bits before true positives, false positives and false negatives are computed.
`tokenize`(text)	

pine.backend.pineiaa.bratiaa.**compute_f1_agreement**(*annotators*, *documents*, *labels*, *token_func=None*, *eval_func=None*)

pine.backend.pineiaa.bratiaa.**iaa_report**(*f1_agreement*, *precision=3*)

pine.backend.pineiaa.bratiaa.**AnnFile**

class pine.backend.pineiaa.bratiaa.**F1Agreement**(*annotators*, *documents*, *labels*, *eval_func=exact_match_instance_evaluation*, *token_func=None*)

property annotators(*self*)

property documents(*self*)

property labels(*self*)

_compute_tp_fp_fn(*self*, *documents*)

_increment_counts(*self*, *annotations*, *pair*, *doc*, *kind*)

mean_sd_per_label(*self*)
Mean and standard deviation of all annotator combinations' F1 scores by label.

mean_sd_per_document(*self*)
Mean and standard deviation of all annotator combinations' F1 scores per document.

mean_sd_total(*self*)
Mean and standard deviation of all annotator cominations' F1 scores.

mean_sd_per_label_one_vs_rest (*self*, *annotator*)
: Mean and standard deviation of all annotator combinations' F1 scores involving given annotator per label.

mean_sd_total_one_vs_rest (*self*, *annotator*)
: Mean and standard deviation of all annotator combinations' F1 scores involving given annotator.

_pairs_involving (*self*, *annotator*)

static _mean_sd (*f1_pairs*)
: Mean and standard deviation along first axis.

static print_table (*row_label_header*, *row_labels*, *avg*, *stddev*, *precision=3*)

compute_total_f1_matrix (*self*)
: Returns (n x n) matrix, where n is the number of annotators, containing pair-wise total F1 scores between all annotators.

  By definition, the matrix is symmetric and F1 = 1 on the main diagonal.

draw_heatmap (*self*, *out_path*)
: Draws heatmap based on square matrix of F1 scores.

class pine.backend.pineiaa.bratiaa.**Document** (*txt*, *doc_id*)

__slots__ = ['ann_files', 'txt', 'doc_id']

pine.backend.pineiaa.bratiaa.**input_generator** (*json_list*)

pine.backend.pineiaa.bratiaa.**exact_match_instance_evaluation** (*ann_list_1*, *ann_list2*, *tokens=None*)

pine.backend.pineiaa.bratiaa.**exact_match_token_evaluation** (*ann_list_1*, *ann_list_2*, *tokens=None*)
: Annotations are split into token-sized bits before true positives, false positives and false negatives are computed.

  Sub-token annotations are expanded to full tokens. Long annotations will influence the results more than short annotations. Boundary errors for adjacent annotations with the same label are ignored!

pine.backend.pineiaa.bratiaa.**Annotation**

pine.backend.pineiaa.bratiaa.**tokenize** (*text*)

Submodules

pine.backend.pineiaa.bp

Module Contents

Functions

get_current_report(collection_id)	
get_iia_report_by_collection_id(collection_id)	
update_iaa_report_by_collection_id(collection_id: str) → bool	

continues on next page

Table 29 – continued from previous page

create_iaa_report_by_collection_id(collection_id)
init_app(app)

pine.backend.pineiaa.bp.**logger**

pine.backend.pineiaa.bp.**bp**

pine.backend.pineiaa.bp.**get_current_report**(*collection_id*)

pine.backend.pineiaa.bp.**get_iia_report_by_collection_id**(*collection_id*)

pine.backend.pineiaa.bp.**update_iaa_report_by_collection_id**(*collection_id: str*) → bool

pine.backend.pineiaa.bp.**create_iaa_report_by_collection_id**(*collection_id*)

pine.backend.pineiaa.bp.**init_app**(*app*)

Package Contents

Functions

update_iaa_report_by_collection_id(collection_id: str) → bool

pine.backend.pineiaa.**update_iaa_report_by_collection_id**(*collection_id: str*) → bool

pine.backend.pipelines

Submodules

pine.backend.pipelines.bp

Module Contents

Functions

_get_classifier(classifier_id: str) → dict
_clear_classifier(classifier_id: str)
_get_classifier_pipeline(classifier_id: str) → dict
_check_permissions(classifier: dict)
_get_pipeline_status(pipeline: str, classifier_id: str) → dict

continues on next page

Table 31 – continued from previous page

_get_pipeline_running_jobs(pipeline: str, classifier_id: str) → List[str]	
_train_pipeline(pipeline: str, classifier_id: str, model_name: str) → dict	
_predict_pipeline(pipeline: str, classifier_id: str, document_ids: List[str], texts: List[str], timeout_in_s: int) → dict	
get_pipelines()	
get_pipeline_by_id(pipeline_id: str)	
get_pipeline_status(pipeline_id: str) → flask.Response	
_get_classifier_metrics(classifier_id: str)	
get_classifier_metrics(classifier_id: str)	
_get_collection_classifier(collection_id: str) → dict	
get_collection_classifier(collection_id: str)	
get_classifier_status(classifier_id: str)	
get_running_jobs(classifier_id: str)	
train(classifier_id: str)	
predict(classifier_id: str)	
_get_next_instance(classifier_id: str)	
_check_instance_overlap(classifier: dict, instance: dict, user_id: str)	
get_next_by_classifier(classifier_id: str)	
advance_to_next_document_by_classifier(classifier_id: str, document_id: str)	
init_app(app)	

pine.backend.pipelines.bp.**logger**

pine.backend.pipelines.bp.**service_manager**

pine.backend.pipelines.bp.**bp**

pine.backend.pipelines.bp.**_cached_classifiers**

pine.backend.pipelines.bp.**_cached_classifier_pipelines**

pine.backend.pipelines.bp.**_get_classifier**(*classifier_id: str*) → dict

pine.backend.pipelines.bp.**_clear_classifier**(*classifier_id: str*)

pine.backend.pipelines.bp.**_get_classifier_pipeline**(*classifier_id: str*) → dict pine.backend.pipelines.bp.**_check_permissions**(*classifier: dict*)

pine.backend.pipelines.bp.**_get_pipeline_status**(*pipeline: str*, *classifier_id: str*) → dict

pine.backend.pipelines.bp.**_get_pipeline_running_jobs**(*pipeline: str*, *classifier_id: str*) → List[str]

pine.backend.pipelines.bp.**_train_pipeline**(*pipeline: str*, *classifier_id: str*, *model_name: str*) → dict

pine.backend.pipelines.bp.**_predict_pipeline**(*pipeline: str*, *classifier_id: str*, *document_ids: List[str]*, *texts: List[str]*, *timeout_in_s: int*) → dict

pine.backend.pipelines.bp.**get_pipelines**()

pine.backend.pipelines.bp.**get_pipeline_by_id**(*pipeline_id: str*)

pine.backend.pipelines.bp.**get_pipeline_status**(*pipeline_id: str*) → flask.Response

pine.backend.pipelines.bp.**_get_classifier_metrics**(*classifier_id: str*)

pine.backend.pipelines.bp.**get_classifier_metrics**(*classifier_id: str*)

pine.backend.pipelines.bp.**_get_collection_classifier**(*collection_id: str*) → dict

pine.backend.pipelines.bp.**get_collection_classifier**(*collection_id: str*)

pine.backend.pipelines.bp.**get_classifier_status**(*classifier_id: str*)

pine.backend.pipelines.bp.**get_running_jobs**(*classifier_id: str*)

pine.backend.pipelines.bp.**train**(*classifier_id: str*)

pine.backend.pipelines.bp.**predict**(*classifier_id: str*)

pine.backend.pipelines.bp.**_get_next_instance**(*classifier_id: str*)

pine.backend.pipelines.bp.**_check_instance_overlap**(*classifier: dict*, *instance: dict*, *user_id: str*)

pine.backend.pipelines.bp.**get_next_by_classifier**(*classifier_id: str*)

pine.backend.pipelines.bp.**advance_to_next_document_by_classifier**(*classifier_id: str*, *document_id: str*)

pine.backend.pipelines.bp.**init_app**(*app*)

pine.backend.shared

Submodules

pine.backend.shared.config

Module Contents

Classes

BaseConfig
TestConfig
ConfigBuilder

pine.backend.shared.config.**LOGGER**

class pine.backend.shared.config.**BaseConfig**(*root_dir=None*)

Bases: object

ROOT_DIR

BASE_DIR

BASE_CFG_FILE = config.yaml

BASE_ENV_PREFIX = AL_

DEBUG = True

TESTING = False

LOGGER_NAME

LOGGER_DIR = logs

LOGGER_FILE = debug.log

LOGGER_LEVEL

EVE_HOST = localhost

EVE_PORT = 5001

REDIS_HOST = localhost

REDIS_PORT = 6379

REDIS_USR

REDIS_PWD

REDIS_DBNUM = 0

REDIS_PREFIX = AL:

REDIS_EXPIRE = 3600

SCHEDULER_REGISTRATION_TIMEOUT

SCHEDULER_HANDLER_TIMEOUT

SCHEDULER_QUEUE_TIMEOUT

SERVICE_REGISTRATION_CHANNEL = registration

SERVICE_REGISTRATION_FREQUENCY = 60

SERVICE_LISTENING_FREQUENCY = 1

SERVICE_HANDLER_TIMEOUT = 3600

SERVICE_LIST

DATASETS_LOCAL_DIR = D:\Data\DEEPCATT2_DB_DATA\datasets

MODELS_LOCAL_DIR = D:\\Data\\DEEPCATT2_DB_DATA\\Models

classmethod _get_config_var_paths(*cls*, *root_dict=None*)

classmethod _process_paths(*cls*, *alt_path=None*)

classmethod _process_file_cfg(*cls*)

classmethod _process_env_vars(*cls*)

classmethod as_dict(*cls*)

> **Return type** dict

classmethod as_attr_dict(*cls*)

> **Return type** munch.Munch | dict

static _try_cast(*value*, *_type*, *_default=None*)

static _str2bool(*_str*, *_default=None*)

class pine.backend.shared.config.**TestConfig**(*root_dir=None*)
Bases: *pine.backend.shared.config.BaseConfig*

class pine.backend.shared.config.**ConfigBuilder**
Bases: object

__env_cfg_variable = BUILDER_CFG_PROFILE

__current_config_instance

__current_config_instance_name

__current_config_instance_print = False

__arg_parser

static __get_configs()
:rtype list[Callable[…, BaseConfig]]

static get_config_names()

> **Return type** list[str]

classmethod get_arg_parser(*cls*)

> **Return type** ArgumentParser

classmethod init_config(*cls*, *config_name=None*, *config_base=None*, *enable_terminal=True*, *as_attr_dict=True*)

classmethod get_config(*cls*, *config_name=None*, *config_base=None*, *enable_terminal=True*, *as_attr_dict=True*)

> **Return type** *BaseConfig*

classmethod set_config(*cls*, *config_name=None*, *config_base=None*, *enable_terminal=True*, *as_attr_dict=True*)

classmethod __parse_terminal_config(*cls*)

> **Return type** argparse.Namespace

pine.backend.shared.transform

Module Contents

Functions

transform_module_by_config(module_ref, config_ref, config_prefix=None)	Transforms a given module's properties based on ConfigBuilder Values.

pine.backend.shared.transform.**transform_module_by_config**(*module_ref*, *config_ref*, *config_prefix=None*)

Transforms a given module's properties based on ConfigBuilder Values. The prefix can be used to avoid blindy changing values and target a subset of matching values in config_ref. :type module_ref: ModuleType :type config_ref: dict :type config_prefix: str

Submodules

pine.backend.app

Module Contents

Functions

handle_error(e)
handle_uncaught_exception(e)
create_app(test_config=None)

pine.backend.app.**VERSION**

pine.backend.app.**LOGGER**

pine.backend.app.**handle_error**(*e*)

pine.backend.app.**handle_uncaught_exception**(*e*)

pine.backend.app.**create_app**(*test_config=None*)

pine.backend.config

Module Contents

pine.backend.config.**SECRET_KEY = Cq13XII=%**

pine.backend.config.**DEBUG = True**

pine.backend.config.**EVE_SERVER**

pine.backend.config.**REDIS_SERVER**

pine.backend.config.**REDIS_PORT**

pine.backend.config.**AUTH_MODULE**

pine.backend.config.**AUTH_MODULE = vegas**

pine.backend.config.**VEGAS_SERVER**

pine.backend.config.**VEGAS_CLIENT_ID**

pine.backend.config.**VEGAS_CLIENT_SECRET**

pine.backend.config.**DOCUMENT_IMAGE_DIR**

pine.backend.cors

Module Contents

Functions

not_found(e)	
init_app(app)	

pine.backend.cors.**not_found**(*e*)

pine.backend.cors.**init_app**(*app*)

pine.backend.log

Module Contents

Classes

Action	Generic enumeration.

Functions

setup_logging()	
get_flask_request_info()	
get_flask_logged_in_user()	
access_flask_login()	
access_flask_logout(user: dict)	

continues on next page

Table 37 – continued from previous page

access_flask_add_collection(collection: dict)	
access_flask_view_document(document: dict)	
access_flask_add_document(document: dict)	
access_flask_add_documents(documents: List[dict])	
access_flask_annotate_document(annotation)	
access_flask_annotate_documents(annotations: List[dict])	
access(action, user, request_info, message, **extra_info)	

pine.backend.log.**CONFIG_FILE_ENV = PINE_LOGGING_CONFIG_FILE**

pine.backend.log.**ACCESS_LOGGER_NAME = pine.access**

pine.backend.log.**ACCESS_LOGGER**

class pine.backend.log.**Action**

Bases: enum.Enum

Generic enumeration.

Derive from this class to define new enumerations.

LOGIN

LOGOUT

CREATE_COLLECTION

VIEW_DOCUMENT

ADD_DOCUMENT

ADD_DOCUMENTS

ANNOTATE_DOCUMENT

ANNOTATE_DOCUMENTS

pine.backend.log.**setup_logging**()

pine.backend.log.**get_flask_request_info**()

pine.backend.log.**get_flask_logged_in_user**()

pine.backend.log.**access_flask_login**()

pine.backend.log.**access_flask_logout**(*user: dict*)

pine.backend.log.**access_flask_add_collection**(*collection: dict*)

pine.backend.log.**access_flask_view_document**(*document: dict*)

pine.backend.log.**access_flask_add_document**(*document: dict*)

pine.backend.log.**access_flask_add_documents**(*documents: List[dict]*)

pine.backend.log.**access_flask_annotate_document**(*annotation*) `pine.backend.log.`**`access_flask_annotate_documents`**(*annotations: List[dict]*)

`pine.backend.log.`**`access`**(*action*, *user*, *request_info*, *message*, ***extra_info*)

`pine.backend.models`

Module Contents

Classes

LoginFormFieldType	Generic enumeration.
LoginFormField	
LoginForm	
AuthUser	
UserDetails	
CollectionUserPermissions	Collection permissions for a user as a dictionary of boolean flags.

class `pine.backend.models.`**`LoginFormFieldType`**

Bases: `enum.Enum`

Generic enumeration.

Derive from this class to define new enumerations.

TEXT = text

PASSWORD = password

class `pine.backend.models.`**`LoginFormField`**(*name: str*, *display: str*, *field_type: pine.backend.models.LoginFormFieldType*)

Bases: `object`

to_dict(*self*)

class `pine.backend.models.`**`LoginForm`**(*fields: list*, *button_text: str*)

Bases: `object`

to_dict(*self*)

class `pine.backend.models.`**`AuthUser`**

Bases: `object`

property id(*self*)

property username(*self*)

property display_name(*self*)

property is_admin(*self*)

to_dict(*self*)

class `pine.backend.models.`**`UserDetails`**(*first_name*, *last_name*, *description*)

Bases: `object`

to_dict(*self*)

class pine.backend.models.**CollectionUserPermissions**(*view=False, annotate=False, add_documents=False, add_images=False, modify_users=False, modify_labels=False, modify_document_metadata=False, download_data=False, archive=False, delete_documents=False*)

Bases: `object`

Collection permissions for a user as a dictionary of boolean flags.

view :bool
Whether the user can view the collection and documents. :type: bool

annotate :bool
Whether the user can annotate collection documents. :type: bool

add_documents :bool
Whether the user can add documents to the collection. :type: bool

add_images :bool
Whether the user can add images to the collection. :type: bool

modify_users :bool
Whether the user can modify the list of viewers/annotators for the collection. :type: bool

modify_labels :bool
Whether the user can modify the list of labels for the collection. :type: bool

modify_document_metadata :bool
Whether the user can modify document metadata (such as changing the image). :type: bool

download_data :bool
Whether the user can download the collection data :type: bool

archive :bool
Whether the user can archive or unarchive the collection. :type: bool

delete_documents :bool
Whether the user can delete documents in the collection. :type: bool

to_dict(*self*) → dict
Returns a dict version of this object for conversion to JSON.

Returns a dict version of this object for conversion to JSON

Return type dict

Package Contents

Functions

create_app(test_config=None)	

pine.backend.**create_app**(*test_config=None*)

pine.backend.**VERSION**

pine.backend.**__version__**

pine.client

PINE client module.

Submodules

pine.client.client

PINE client classes module.

Module Contents

Classes

BaseClient	Base class for a client using a REST interface.
EveClient	A client to access Eve and, optionally, its underlying MongoDB instance.
PineClient	A client to access PINE (more specifically: the backend).
LocalPineClient	A client for a local PINE instance, including an *EveClient*.

Functions

_standardize_path(path: str, *additional_paths: List[str]) → List[str]	Standardize path(s) into a list of path components.

pine.client.client.**_standardize_path**(*path: str, *additional_paths: List[str]*) → List[str]

Standardize path(s) into a list of path components.

Parameters

- **path** (*str*) – relative path, e.g. "users"
- ***additional_paths** (*list(str), optional*) – any additional path components

in a list

Returns the standardized path components in a list

Return type list(str)

class `pine.client.client.`**`BaseClient`**(*base_uri: str*, *name: str = None*, *verify_ssl: bool = True*)

Bases: `object`

Base class for a client using a REST interface.

Constructor.

Parameters

- **`base_uri`** (*`str`*) – the base URI for the server, e.g. `"http://localhost:5000"`
- **`name`** (*`str, optional`*) – optional human-readable name for the server, defaults to None
- **`verify_ssl`** (*`bool, optional`*) – whether to verify SSL/HTTPs calls; do not turn this off unless you are fully aware of the security consequences

`__metaclass__`

`base_uri :str`

The server's base URI.

Type str

`session :requests.Session`

The currently open session, or `None`.

Type requests.Session

`verify_ssl :bool`

Whether to verify SSL/HTTPS calls. If you turn this off you should be fully aware of the security consequences of such.

Type bool

`abstract is_valid`(*self*) → bool

Returns whether this client and its connection(s) are valid.

Returns whether this client and its connection(s) are valid

Return type bool

`uri`(*self*, *path: str*, **additional_paths: List[str]*) → str

Makes a complete URI from the given path(s).

Parameters

- **`path`** (*`str`*) – relative path, e.g. `"users"`
- **`*additional_paths`** (*`list(str), optional`*) – any additional path components

Returns the complete, standardized URI including the base URI, e.g. `"http://localhost:5000/users"`

Return type str

`_req`(*self*, *method: str*, *path: str*, **additional_paths: List[str]*, ***kwargs*) → requests.Response

Makes a `requests` call, checks for errors, and returns the response.

Parameters

- **method** (*str*) – REST method ("get", "post", etc.)
- **path** (*str*) – relative path, e.g. "users"
- ***additional_paths** (*list(str), optional*) – any additional path components
- ****kwargs** (*dict*) – any additional kwargs to send to `requests`

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the `requests requests.Response` object

Return type requests.Response

get (*self, path: str, *additional_paths: List[str], **kwargs*) → requests.Response

Makes a `requests` `GET` call, checks for errors, and returns the response.

Parameters

- **path** (*str*) – relative path, e.g. "users"
- ***additional_paths** (*list(str), optional*) – any additional path components
- ****kwargs** (*dict*) – any additional kwargs to send to `requests`

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the `requests Response` object

Return type requests.Response

put (*self, path: str, *additional_paths: List[str], **kwargs*) → requests.Response

Makes a `requests` `PUT` call, checks for errors, and returns the response.

Parameters

- **path** (*str*) – relative path, e.g. "users"
- ***additional_paths** (*list(str), optional*) – any additional path components
- ****kwargs** (*dict*) – any additional kwargs to send to `requests`

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the `requests Response` object

Return type requests.Response

patch (*self, path: str, *additional_paths: List[str], **kwargs*) → requests.Response

Makes a `requests` `PATCH` call, checks for errors, and returns the response.

Parameters

- **path** (*str*) – relative path, e.g. "users"
- ***additional_paths** (*list(str), optional*) – any additional path components
- ****kwargs** (*dict*) – any additional kwargs to send to `requests`

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the `requests` `Response` object

Return type requests.Response

post (*self, path: str, *additional_paths: List[str], **kwargs*) → requests.Response

Makes a `requests` `POST` call, checks for errors, and returns the response.

Parameters

- **path** (*str*) – relative path, e.g. `"users"`
- ***additional_paths** (*list (str), optional*) – any additional path components
- ****kwargs** (*dict*) – any additional kwargs to send to `requests`

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the `requests` `Response` object

Return type requests.Response

delete (*self, path: str, *additional_paths: List[str], **kwargs*) → requests.Response

Makes a `requests` `DELETE` call, checks for errors, and returns the response.

Parameters

- **path** (*str*) – relative path, e.g. `"users"`
- ***additional_paths** (*list (str), optional*) – any additional path components
- ****kwargs** (*dict*) – any additional kwargs to send to `requests`

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the `requests` `Response` object

Return type requests.Response

class pine.client.client.**EveClient** (*eve_base_uri: str, mongo_base_uri: str = None, mongo_dbname: str = DEFAULT_DBNAME, verify_ssl: bool = True*)

Bases: *pine.client.client.BaseClient*

A client to access Eve and, optionally, its underlying MongoDB instance.

Constructor.

Parameters

- **eve_base_uri** (*str*) – the base URI for the eve server, e.g. `"http://localhost:5001"`
- **mongo_base_uri** (*str, optional*) – the base URI for the mongodb server, e.g. `"mongodb://localhost:27018"`, defaults to `None`
- **mongo_dbname** (*str, optional*) – the DB name that PINE uses, defaults to `"pmap_nlp"`
- **verify_ssl** (*bool, optional*) – whether to verify SSL/HTTPs calls; do not turn this off unless you are fully aware of the security consequences

DEFAULT_DBNAME :str = pmap_nlp

The default DB name used by PINE.

Type str

mongo_base_uri :str
The base URI for the MongoDB server.

Type str

mongo :pymongo.MongoClient
The `pymongo.mongo_client.MongoClient` instance.

Type pymongo.mongo_client.MongoClient

mongo_db :pymongo.database.Database
The `pymongo.database.Database` instance.

Type pymongo.database.Database

is_valid(*self*) → bool
Returns whether this client and its connection(s) are valid.

Returns whether this client and its connection(s) are valid

Return type bool

ping(*self*) → Any
Pings the eve server and returns the result.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the JSON response from the server (probably `"pong"`)

about(*self*) → dict
Returns the 'about' dict from the server.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the JSON response from the server

Return type dict

get_resource(*self*, *resource: str*, *resource_id: str*) → dict
Gets a resource from eve by its ID.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the JSON object response from the server

Return type dict

_add_or_replace_resource(*self*, *resource: str*, *obj: dict*, *valid_fn: Callable[[dict, typing.Callable[[str], None]], bool] = None*) → str
Adds or replaces the given resource.

Parameters

- **resource** (*str*) – the resource type, e.g. `"users"`
- **obj** (*dict*) – the resource object
- **valid_fn** (*function, optional*) – a function to validate the resource object, defaults to `None`

Raises

- ***exceptions.PineClientValueException*** – if a valid_fn is passed in and the object fails
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the ID of the added/replaced resource

Return type str

`_add_resources`(*self*, *resource: str*, *objs: List[dict]*, *valid_fn: Callable[[dict, typing.Callable[[str], None]], bool] = None*, *replace_if_exists: bool = False*)

Tries to add all the resource objects at once, optionally falling back to individual replacement if that fails.

Parameters

- **`resource`** (*`str`*) – the resource type, e.g. `"users"`
- **`objs`** (*`list (dict)`*) – the resource objects
- **`valid_fn`** (*`function, optional`*) – a function to validate the resource object, defaults to `None`
- **`replace_if_exists`** (*`bool, optional`*) – whether to replace the resource with the given value if it already exists on the server, defaults to `False`

Raises

- ***exceptions.PineClientValueException*** – if a valid_fn is passed in and any of the objects fails
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the IDs of the added resources

Return type list(str)

`add_users`(*self*, *users: List[dict]*, *replace_if_exists=False*) → List[str]

Adds the given users.

Parameters

- **`users`** (*`list (dict)`*) – the user objects
- **`replace_if_exists`** (*`bool, optional`*) – whether to replace the resource with the given value if it already exists on the server, defaults to `False`

Raises

- ***exceptions.PineClientValueException*** – if any of the user objects are not valid, see `models.is_valid_eve_user()`
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the IDs of the added users

Return type list(str)

`get_users`(*self*)

Gets all users.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns all the users

Return type list(dict)

add_pipelines(*self*, *pipelines: List[dict]*, *replace_if_exists=False*) → List[str]

Adds the given pipelines.

Parameters

- **pipelines** (*list (dict)*) – the pipeline objects
- **replace_if_exists** (*bool, optional*) – whether to replace the resource with the given value if it already exists on the server, defaults to `False`

Raises

- ***exceptions.PineClientValueException*** – if any of the pipeline objects are not valid, see *models.is_valid_eve_pipeline()*
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the IDs of the added pipelines

Return type list(str)

class pine.client.client.**PineClient**(*backend_base_uri: str*, *verify_ssl: bool = True*)

Bases: *pine.client.client.BaseClient*

A client to access PINE (more specifically: the backend).

Constructor.

Parameters

- **backend_base_uri** (*str*) – the base URI for the backend server, e.g. `"http://localhost:5000"`
- **verify_ssl** (*bool, optional*) – whether to verify SSL/HTTPs calls; do not turn this off unless you are fully aware of the security consequences

is_valid(*self*) → bool

Returns whether this client and its connection(s) are valid.

Returns whether this client and its connection(s) are valid

Return type bool

ping(*self*) → Any

Pings the backend server and returns the result.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the JSON response from the server (probably `"pong"`)

about(*self*) → dict

Returns the 'about' dict from the server.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the JSON response from the server

Return type dict

get_logged_in_user(*self*) → dict

Returns the currently logged in user, or None if not logged in.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns currently logged in user, or None if not logged in

Return type dict

get_my_user_id(*self*) → str

Returns the ID of the logged in user, or None if not logged in.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the ID of the logged in user, or None if not logged in

Return type str

is_logged_in(*self*) → bool

Returns whether the user is currently logged in or not.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns whether the user is currently logged in or not

Return type bool

_check_login(*self*)

Checks whether user is logged in and raises an `exceptions.PineClientAuthException` if not.

Raises

- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

get_auth_module(*self*) → str

Returns the PINE authentication module, e.g. `"eve"`.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the PINE authentication module, e.g. `"eve"`

Return type str

login_eve(*self*, *username: str*, *password: str*) → bool

Logs in using eve credentials, and returns whether it was successful.

Parameters

- **username** (*str*) – username
- **password** (*str*) – password

Raises

- ***exceptions.PineClientAuthException*** – if auth module is not eve or login was not successful
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns whether the login was successful

Return type bool

authorize_vegas(*self*, *json_token: dict*) → bool

Logs in using a VEGAS token, and returns whether it was successful.

The token should be in the same format as is returned by VEGAS's oauth2/accesstoken endpoint, e.g. containing fields "access_token", "token_type", "expires_in", etc.

Parameters **`json_token`** (*`dict`*) – the token returned by VEGAS

Raises

- ***`exceptions.PineClientAuthException`*** – if auth module is not vegas or authorization was not successful
- ***`exceptions.PineClientHttpException`*** – if the HTTP request returns an error

Returns whether the authorization was successful

Return type bool

`logout` (*self*)

Logs out the current user.

Raises ***`exceptions.PineClientHttpException`*** – if the HTTP request returns an error

`get_pipelines` (*self*) → List[dict]

Returns all pipelines accessible to logged in user.

Raises

- ***`exceptions.PineClientAuthException`*** – if not logged in
- ***`exceptions.PineClientHttpException`*** – if the HTTP request returns an error

Returns all pipelines accessible to logged in user

Return type list(dict)

`get_pipeline_status` (*self*, *pipeline_id: str*) → dict

Returns status for the given pipeline.

Parameters **`pipeline_id`** – str: pipeline ID

Returns pipeline status

Return type dict

`collection_builder` (*self*, ***kwargs: dict*) → *pine.client.models.CollectionBuilder*

Makes and returns a new `models.CollectionBuilder` with the logged in user.

Parameters **`**kwargs`** (*`dict`*) – any additional args to pass in to the constructor

Returns a new *`models.CollectionBuilder`* with the logged in user

Return type *models.CollectionBuilder*

`create_collection` (*self*, *builder:* pine.client.models.CollectionBuilder) → str

Creates a collection using the current value of the given builder and returns its ID.

Parameters **`builder`** (`models.CollectionBuilder`) – collection builder

Raises

- ***`exceptions.PineClientValueException`*** – if the given collection is not valid, see *`models.is_valid_collection()`*
- ***`exceptions.PineClientAuthException`*** – if not logged in
- ***`exceptions.PineClientHttpException`*** – if the HTTP request returns an error

Returns the created collection's ID

Return type str

archive_collection (*self*, *collection_id: str*, *archive: bool = True*) → dict

Archives or unarchives the given collection.

Parameters

- **collection_id** – str: the ID of the collection
- **archive** – bool: whether to archive (True) or unarchive (False) the collection

Returns updated collection information

Return type dict

get_collection_permissions (*self*, *collection_id: str*) → *pine.client.models.CollectionUserPermissions*

Returns collection permissions for the logged in user.

Parameters **collection_id** (*str*) – the ID of the collection

Returns the collection permissions

Return type *models.CollectionUserPermissions*

get_collection_documents (*self*, *collection_id: str*, *truncate: bool*, *truncate_length: int = 30*) → List[dict]

Returns all the documents in the given collection.

Parameters

- **collection_id** (*str*) – the ID of the collection
- **truncate** (*bool*) – whether to truncate the document text (a good idea unless you need it)
- **truncate_length** (*int, optional*) – how many characters of the text you want if truncated, defaults to `30`

Returns all the documents in the given collection

Return type list(dict)

get_collection_classifier (*self*, *collection_id: str*) → dict

Returns the classifier associated with the given collection.

Parameters **collection_id** (*str*) – the ID of the collection

Returns the classifier associated with the given collection

Return type dict

get_next_document (*self*, *classifier_id: str*) → str

Returns the 'next' document associated with the given classifier.

The next document is the one that the model suggests should be annotated by the logged-in user next.

Parameters **classifier_id** – str: ID of the classifier

Returns the next document ID, or None if there are none left to annotate

Return type str

advance_next_document (*self*, *classifier_id: str*, *document_id: str*) → dict

Advances the 'next' document associated with the given classifier by marking the given document as annotated.

The next document is the one that the model suggests should be annotated by the logged-in user next.

Parameters

- **`classifier_id`** – str: ID of the classifier
- **`document_id`** – str: the ID of the document that was annotated

Returns information on the advanced instance

Return type dict

`add_document`(*self*, *document: dict = {}*, *creator_id: str = None*, *collection_id: str = None*, *overlap: int = None*, *text: str = None*, *metadata: dict = None*) → str

Adds a new document to a collection and returns its ID.

Will use the logged in user ID for the creator_id if none is given. Although all the parameters are optional, you must provide values either in the document or through the kwargs in order to make a valid document.

Parameters

- **`document`** (*`dict, optional`*) – optional document dict, will be overridden with any kwargs, defaults to `{}`
- **`creator_id`** (*`str, optional`*) – optional creator_id for the document, defaults to `None` (not set)
- **`collection_id`** (*`str, optional`*) – optional collection_id for the document, defaults to `None` (not set)
- **`overlap`** (*`int, optional`*) – optional overlap for the document, defaults to `None` (not set)
- **`text`** (*`str, optional`*) – optional text for the document, defaults to `None` (not set)
- **`metadata`** (*`dict, optional`*) – optional metadata for the document, defaults to `None` (not set)

Raises

- ***`exceptions.PineClientValueException`*** – if the given document parameters are not valid, see *`models.is_valid_eve_document()`*
- ***`exceptions.PineClientAuthException`*** – if not logged in
- ***`exceptions.PineClientHttpException`*** – if the HTTP request returns an error

Returns the created document's ID

Return type str

`add_documents`(*self*, *documents: List[dict]*, *creator_id: str = None*, *collection_id: str = None*) → List[str]

Adds multiple documents at once and returns their IDs.

Will use the logged in user ID for the creator_id if none is given.

Parameters

- **`documents`** (*`list(dict)`*) – the documents to add
- **`creator_id`** (*`str, optional`*) – optional creator_id to set in the documents, defaults to `None` (not set)
- **`collection_id`** (*`str, optional`*) – optional collection_id to set in the documents, defaults to `None` (not set)

Raises

- ***`exceptions.PineClientValueException`*** – if any of the given documents are not valid, see *`models.is_valid_eve_document()`*

- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the created documents' IDs

Return type list(str)

delete_document (*self*, *document_id: str*) → dict

Deletes the document and associated data with the given ID.

Parameters **document_id** – str: ID of the document to delete

Returns information about deleted objects

Return type dict

delete_documents (*self*, *document_ids: List[str]*) → dict

Deletes the documents and associated data with the given IDs.

Parameters **document_ids** – list[str]: IDs of the documents to delete

Returns information about deleted objects

Return type dict

annotate_document (*self*, *document_id: str*, *doc_annotations: List[str]*, *ner_annotations: List[Union[dict, list, tuple]]*, *update_iaa: bool = True*) → str

Annotates the given document with the given values.

Parameters

- **document_id** (*str*) – the document ID to annotate
- **doc_annotations** (*list (str)*) – document annotations/labels
- **ner_annotations** (*list*) – NER annotations, where each annotation is either a list or a dict
- **update_iaa** – whether to also update IAA reports related to this document, defaults to *True*

Type update_iaa: bool

Raises

- ***exceptions.PineClientValueException*** – if any of the given annotations are not valid, see *models.is_valid_annotation()*
- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the annotation ID

Return type str

annotate_collection_documents (*self*, *collection_id: str*, *document_annotations: dict*, *skip_document_updates=False*, *update_iaa=True*) → List[str]

Annotates documents in a collection.

Parameters

- **collection_id** (*str*) – the ID of the collection containing the documents to annotate
- **document_annotations** (*dict*) – a dict containing "ner" list and "doc" list

- **skip_document_updates** (*bool*) – whether to skip updating the document "has_annotated" map, defaults to `False`. This should only be `True` if you properly set the "has_annotated" map when you created the document.
- **update_iaa** (*bool*) – whether to also update IAA report for the collection, defaults to `True`

Raises

- ***exceptions.PineClientValueException*** – if any of the given annotations are not valid, see *models.is_valid_doc_annotations()*
- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the annotation IDs

Return type list(str)

get_my_document_annotations(*self*, *document_id: str*) → List[List[dict]]

Returns annotations for the given document for the logged in user.

Parameters **document_id** – the ID of the document to get annotations for

Raises

- ***exceptions.PineClientValueException*** – if the document ID is not a valid string
- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the annotations for the given document for the logged in user

Return type list(list(dict))

get_others_document_annotations(*self*, *document_id: str*) → List[List]

Returns annotations for the given document for users other than the logged in user.

Parameters **document_id** – the ID of the document to get annotations for

Raises

- ***exceptions.PineClientValueException*** – if the document ID is not a valid string
- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the annotations for the given document for users other than the logged in user

Return type list(list(dict))

list_collections(*self*, *include_archived: bool = False*) → List[dict]

Returns a list of user's collections.

Parameters **include_archived** (*bool*) – whether to include archived collections, defaults to `False`

Raises

- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns user's collections

Return type list(dict)

get_collection(*self*, *collection_id: str*) → dict

Returns the collection with the given ID.

Parameters **collection_id** – str: the ID of the collection

Returns the collection data

Return type dict

get_collection_iaa_report(*self*, *collection_id: str*) → dict

Returns IAA (inter-annotation agreement) report for the given collection.

Parameters **collection_id** – the ID of the collection for which to get report

Type collection_id: str

Returns report data

Return type dict

download_collection_data(*self*, *collection_id: str*, *include_collection_metadata: bool = True*, *include_document_metadata: bool = True*, *include_document_text: bool = True*, *include_annotations: bool = True*, *include_annotation_latest_version_only: bool = True*)

Downloads collection data.

Parameters

- **collection_id** (*str*) – the ID of the collection for which to download data
- **include_collection_metadata** (*bool*) – whether to include collection metadata, defaults to `True`
- **include_document_metadata** (*bool*) – whether to include document metadata, defaults to `True`
- **include_document_text** (*bool*) – whether to include document text, defaults to `True`
- **include_annotations** (*bool*) – whether to include annotations, defaults to `True`
- **include_annotation_latest_version_only** (*bool*) – whether to include only the latest version of annotations (`True`) or all versions (`False`), defaults to `True`

Raises

- ***exceptions.PineClientValueException*** – if given empty collection ID
- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error, such as if the collection doesn't exist

Returns collection data

Return type dict

get_classifier_status(*self*, *classifier_id: str*) → dict

Returns the status for the given classifier.

Param classifier_id: str: classifier ID

Returns status for the given classifier

Return type dict

classifier_train(*self*, *classifier_id: str*, *model_name: str = 'auto-trained'*) → dict
Trains the given classifier (using collection documents).

Note that training is done asynchronously, so this method should return very quickly. One of the things returned in the dict will be a job ID. If you want to know when the training has finished, you can periodically poll `get_classifier_running_jobs()` and check for that job ID.

Parameters

- **classifier_id** – str: classifier ID
- **model_name** – str: name of model corresponding to filename on disk, defaults to `"auto-trained"` which is the same as the annotation-based model training

Return type dict

classifier_has_trained(*self*, *classifier_id: str*) → bool
Returns whether the given classifier has been trained or not.

If False, future calls to predict will fail.

Param classifier_id: str: classifier ID

Return type bool

classifier_predict(*self*, *classifier_id*, *document_ids: List[str]*, *texts: List[str]*, *timeout_in_s: int = 36000*) → dict
Runs classifier prediction on the given documents. At least one of document_ids and texts must be non-empty.

This prediction uses the last-trained model name for that classifier. This method will block until the prediction has finished and then return the results.

Parameters

- **classifier_id** – str: classifier ID
- **document_ids** – list[str]: a list of document IDs to run prediction on
- **texts** – list[str]: a list of direct document texts to run prediction on
- **timeout_in_s** – int: max timeout in seconds before erroring out and returning, defaults to `36000`

Return type dict

get_classifier_running_jobs(*self*, *classifier_id: str*) → List[str]
Gets the list of running job IDs for the given classifier.

Parameters **classifier_id** – str: classifier ID

Return type list[str]

class pine.client.client.**LocalPineClient**(*backend_base_uri: str*, *eve_base_uri: str*, *mongo_base_uri: str = None*, *mongo_dbname: str = EveClient.DEFAULT_DBNAME*, *verify_ssl: bool = True*)
Bases: `pine.client.client.PineClient`

A client for a local PINE instance, including an `EveClient`.

Constructor.

Parameters

- **backend_base_uri** (*str*) – the base URI for the backend server, e.g. `"http://localhost:5000"`
- **eve_base_uri** (*str*) – the base URI for the eve server, e.g. `"http://localhost:5001"`
- **mongo_base_uri** (*str, optional*) – the base URI for the mongodb server, e.g. `"mongodb://localhost:27018"`, defaults to `None`
- **mongo_dbname** (*str, optional*) – the DB name that PINE uses, defaults to `"pmap_nlp"`
- **verify_ssl** (*bool, optional*) – whether to verify SSL/HTTPs calls; do not turn this off unless you are fully aware of the security consequences

eve :EveClient
The local *EveClient* instance.

Type *EveClient*

is_valid(*self*) → bool
Returns whether this client and its connection(s) are valid.

Returns whether this client and its connection(s) are valid

Return type bool

pine.client.exceptions

PINE client exceptions module.

Module Contents

exception pine.client.exceptions.**PineClientException**(*message: str*, *cause: Exception = None*)
Bases: `Exception`

Base class for PINE client exceptions.

Constructor.

Parameters

- **message** (*str*) – the message
- **cause** (*Exception, optional*) – optional cause, defaults to `None`

message
The message.

Type str

exception pine.client.exceptions.**PineClientHttpException**(*method: str*, *path: str*, *resp: requests.Response*)
Bases: *pine.client.exceptions.PineClientException*

A PINE client exception caused by an underlying HTTP exception.

Constructor.

Parameters

- **method** (*str*) – the REST method (`"get"`, `"post"`, etc.)

- **path** (*str*) – the human-readable path that caused the exception
- **resp** (*requests.Response*) – the `Response` with the error info

resp :requests.Response
The `Response` with the error info

Type requests.Response

property status_code (*self*)

exception `pine.client.exceptions.`**PineClientValueException** (*obj: dict*, *obj_type: str*)
Bases: *pine.client.exceptions.PineClientException*

A PINE client exception caused by passing invalid data.

Constructor.

Parameters

- **obj** (*dict*) – the error data
- **obj_type** (*str*) – human-readable type of object

exception `pine.client.exceptions.`**PineClientAuthException** (*message: str*, *cause: Exception = None*)
Bases: *pine.client.exceptions.PineClientException*

Base class for PINE client exceptions.

Constructor.

Parameters

- **message** (*str*) – the message
- **cause** (*Exception, optional*) – optional cause, defaults to `None`

pine.client.log

Module Contents

Functions

setup_logging()	Sets up logging, if configured to do so.

`pine.client.log.`**CONFIG_FILE_ENV :str = PINE_LOGGING_CONFIG_FILE**
The environment variable that optionally contains the file to use for logging configuration.

Type str

`pine.client.log.`**setup_logging** ()
Sets up logging, if configured to do so.

The environment variable named by *CONFIG_FILE_ENV* is checked and, if present, is passed to `logging.config.dictConfig()`.

pine.client.models

Module Contents

Classes

CollectionBuilder	A class that can build the form and files fields that are necessary to create a collection.
CollectionUserPermissions	Collection permissions for a user as a dictionary of boolean flags.

Functions

_check_field_required_bool(obj: dict, field: str) → bool	Checks that the given field is in the object and is a bool.
_check_field_int(obj: dict, field: str) → bool	Checks that if the given field is in the object, that it is an int.
_check_field_required_int(obj: dict, field: str) → bool	Checks that the given field is in the object and is an int.
_check_field_float(obj: dict, field: str) → bool	Checks that if the given field is in the object, that it is a float.
_check_field_required_float(obj: dict, field: str) → bool	Checks that the given field is in the object and is a float.
_check_field_string(obj: dict, field: str) → bool	Checks that if the given field is in the object, that it is a string.
_check_field_required_string(obj: dict, field: str) → bool	Checks that the given field is in the object and is a string.
_check_field_string_list(obj: dict, field: str, min_length: int = 0) → bool	Checks that if the given field is in the object, that it is a string list.
_check_field_required_string_list(obj: dict, field: str, min_length: int = 0) → bool	Checks that the given field is in the object and is a string list.
_check_field_dict(obj: dict, field: str) → bool	Checks that if the given field is in the object, that it is a dict.
_check_field_required_dict(obj: dict, field: str) → bool	Checks that the given field is in the object and is a dict.
_check_field_bool(obj: dict, field: str) → bool	Checks that if the given field is in the object, that it is a bool.
is_valid_eve_user(user: dict, error_callback: Callable[([str], None)] = None) → bool	Checks whether the given user object is valid.
is_valid_eve_pipeline(pipeline: dict, error_callback: Callable[([str], None)] = None) → bool	Checks whether the given pipeline object is valid.
is_valid_eve_collection(collection: dict, error_callback: Callable[([str], None)] = None) → bool	Checks whether the given collection object is valid.
is_valid_collection(form: dict, files: dict, error_callback: Callable[([str], None)] = None) → bool	Checks whether the given form and files parameters are valid for creating a collection.
is_valid_eve_document(document: dict, error_callback: Callable[([str], None)] = None) → bool	Checks whether the given document object is valid.

continues on next page

Table 44 – continued from previous page

is_valid_doc_annotation(ann: Any, error_callback: Callable[([str], None)] = None) → bool	Checks whether the given annotation is a valid document label/annotation.
is_valid_ner_annotation(ann: Any, error_callback: Callable[([str], None)] = None) → bool	Checks whether the given annotation is a valid document NER annotation.
is_valid_annotation(body: dict, error_callback: Callable[([str], None)] = None) → bool	Checks whether the given body is valid to create an annotation.
is_valid_doc_annotations(doc_annotations: dict, error_callback: Callable[([str], None)] = None) → bool	Checks whether the given document annotations are valid.
remove_eve_fields(obj: dict, remove_timestamps: bool = True, remove_versions: bool = True)	Removes fields inserted by eve from the given object.
remove_nonupdatable_fields(obj: dict)	Removes all non-updatable fields from the given object.

pine.client.models.**ID_FIELD :str = _id**
The field used to store database ID.

Type str

pine.client.models.**ITEMS_FIELD :str = _items**
The field used to access the items in a multi-item database response.

Type str

pine.client.models.**_check_field_required_bool** (*obj: dict*, *field: str*) → bool
Checks that the given field is in the object and is a bool.

Parameters

- **obj** (*dict*) – the object to check
- **field** (*str*) – the field to check

Returns whether the given field is in the object and is a bool

Return type bool

pine.client.models.**_check_field_int** (*obj: dict*, *field: str*) → bool
Checks that if the given field is in the object, that it is an int.

Parameters

- **obj** (*dict*) – the object to check
- **field** (*str*) – the field to check

Returns if the given field is in the object, that it is an int

Return type bool

pine.client.models.**_check_field_required_int** (*obj: dict*, *field: str*) → bool
Checks that the given field is in the object and is an int.

Parameters

- **obj** (*dict*) – the object to check
- **field** (*str*) – the field to check

Returns whether the given field is in the object and is an int

Return type bool

`pine.client.models._check_field_float(obj: dict, field: str) → bool`

Checks that if the given field is in the object, that it is a float.

Parameters

- **obj** (*dict*) – the object to check
- **field** (*str*) – the field to check

Returns if the given field is in the object, that it is a float

Return type bool

`pine.client.models._check_field_required_float(obj: dict, field: str) → bool`

Checks that the given field is in the object and is a float.

Parameters

- **obj** (*dict*) – the object to check
- **field** (*str*) – the field to check

Returns whether the given field is in the object and is a float

Return type bool

`pine.client.models._check_field_string(obj: dict, field: str) → bool`

Checks that if the given field is in the object, that it is a string.

Parameters

- **obj** (*dict*) – the object to check
- **field** (*str*) – the field to check

Returns if the given field is in the object, that it is a string

Return type bool

`pine.client.models._check_field_required_string(obj: dict, field: str) → bool`

Checks that the given field is in the object and is a string.

Parameters

- **obj** (*dict*) – the object to check
- **field** (*str*) – the field to check

Returns whether the given field is in the object and is a string

Return type bool

`pine.client.models._check_field_string_list(obj: dict, field: str, min_length: int = 0) → bool`

Checks that if the given field is in the object, that it is a string list.

Parameters

- **obj** (*dict*) – the object to check
- **field** (*str*) – the field to check
- **min_length** (*int, optional*) – the minimum length of the list (if > 0), defaults to 0

Returns if the given field is in the object, that it is a string list

Return type bool

`pine.client.models.`**`_check_field_required_string_list`**(*obj: dict*, *field: str*, *min_length: int = 0*) → bool

Checks that the given field is in the object and is a string list.

Parameters

- **`obj`** (*`dict`*) – the object to check
- **`field`** (*`str`*) – the field to check
- **`min_length`** (*`int, optional`*) – the minimum length of the list (if > 0), defaults to 0

Returns if the given field is in the object, that it is a string list

Return type bool

`pine.client.models.`**`_check_field_dict`**(*obj: dict*, *field: str*) → bool

Checks that if the given field is in the object, that it is a dict.

Parameters

- **`obj`** (*`dict`*) – the object to check
- **`field`** (*`str`*) – the field to check

Returns if the given field is in the object, that it is a dict

Return type bool

`pine.client.models.`**`_check_field_required_dict`**(*obj: dict*, *field: str*) → bool

Checks that the given field is in the object and is a dict.

Parameters

- **`obj`** (*`dict`*) – the object to check
- **`field`** (*`str`*) – the field to check

Returns whether the given field is in the object and is a dict

Return type bool

`pine.client.models.`**`_check_field_bool`**(*obj: dict*, *field: str*) → bool

Checks that if the given field is in the object, that it is a bool.

Parameters

- **`obj`** (*`dict`*) – the object to check
- **`field`** (*`str`*) – the field to check

Returns if the given field is in the object, that it is a bool

Return type bool

`pine.client.models.`**`is_valid_eve_user`**(*user: dict*, *error_callback: Callable[[str], None] = None*) → bool

Checks whether the given user object is valid.

A valid user object has an `_id`, `firstname`, and `lastname` that are non-empty string fields. If `email`, `description`, or `passwdhash` are present, they are string fields. If `role` is present, it is a list of strings that are either `administrator` or `user`.

Parameters

- **`user`** (*`dict`*) – user object
- **`error_callback`** (*`function, optional`*) – optional callback that is called with any error messages, defaults to `None`

Returns whether the given user object is valid

Return type bool

`pine.client.models.`**`is_valid_eve_pipeline`**(*pipeline: dict, error_callback: Callable[[str], None] = None*) → bool

Checks whether the given pipeline object is valid.

A valid pipeline has an `_id`, `title`, and `name` that are non-empty string fields. If `description` is provided, it is a string field. If `parameters` are provided, it is a dict field.

Parameters

- **`pipeline`** (*`dict`*) – pipeline object
- **`error_callback`** (*`function, optional`*) – optional callback that is called with any error messages, defaults to `None`

Returns whether the given pipeline object is valid

Return type bool

`pine.client.models.`**`is_valid_eve_collection`**(*collection: dict, error_callback: Callable[[str], None] = None*) → bool

Checks whether the given collection object is valid.

A valid collection has a `creator_id` that is a non-empty string field. It has a `labels` that is a non-empty list of strings. If `annotators` or `viewers` are provided, they are lists of strings. If `metadata` or `configuration` are provided, they are dicts. If `archived` is provided, it is a bool.

Parameters

- **`collection`** (*`dict`*) – collection object
- **`error_callback`** (*`function, optional`*) – optional callback that is called with any error messages, defaults to `None`

Returns whether the given collection object is valid

Return type bool

`pine.client.models.`**`is_valid_collection`**(*form: dict, files: dict, error_callback: Callable[[str], None] = None*) → bool

Checks whether the given form and files parameters are valid for creating a collection.

A valid form has a `collection` that is a dict field and is valid via *`is_valid_eve_collection()`*. Additionally, the collection has string `title` and `description` fields in its `metadata`. It also has at least one element for `labels`, `viewers`, and `annotators`, and the `creator_id` must be in both `viewers` and `annotators`.

The form also has `overlap` as a float field between 0 and 1 (inclusive), `train_every` as an int field that is at least 5, and `pipelineId` as a string field.

If files are provided, file `file` and any files starting with `imageFile` are checked. If a file `file` is provided, the form must also have a boolean `csvHasHeader` field and an int `csvTextCol` field.

Parameters

- **`form`** (*`dict`*) – form data to send to backend
- **`files`** (*`dict`*) – file data to send to backend
- **`error_callback`** (*`function, optional`*) – optional callback that is called with any error messages, defaults to `None`

Returns whether the given form and files parameters are valid for creating a collection

Return type bool

`pine.client.models.`**`is_valid_eve_document`**(*document: dict*, *error_callback: Callable[[str], None] = None*) → bool

Checks whether the given document object is valid.

A valid document has a `creator_id` and `collection_id` that are non-empty string fields. Optionally, it may have an int `overlap` field, string `text``field, and dict ``metadata` and `has_annotated` fields.

Parameters

- **`document`** (*`dict`*) – document object
- **`error_callback`** (*`function, optional`*) – optional callback that is called with any error messages, defaults to `None`

Returns whether the given document object is valid

Return type bool

`pine.client.models.`**`is_valid_doc_annotation`**(*ann: Any*, *error_callback: Callable[[str], None] = None*) → bool

Checks whether the given annotation is a valid document label/annotation.

This means that it is a non-empty string.

Parameters

- **`ann`** – annotation
- **`error_callback`** (*`function, optional`*) – optional callback that is called with any error messages, defaults to `None`

Returns whether the given annotation is a valid document label/annotation

Return type bool

`pine.client.models.`**`is_valid_ner_annotation`**(*ann: Any*, *error_callback: Callable[[str], None] = None*) → bool

Checks whether the given annotation is a valid document NER annotation.

Valid NER annotations take one of two forms: a `dict` or a `list`/`tuple` of size 3.

A valid NER `dict` has the following fields:

- `start`: an `int` that is >= 0
- `end`: an `int` that is >= 0
- `label`: a non-empty `str`

A valid NER `list`/`tuple` has the following elements:

- element `0`: an `int` that is >= 0
- element `1`: an `int` that is >= 0
- element `2`: a non-empty `str`

Parameters

- **`ann`** – annotation
- **`error_callback`** (*`function, optional`*) – optional callback that is called with any error messages, defaults to `None`

Returns whether the given annotation is a valid document label/annotation

Return type bool

`pine.client.models.`**`is_valid_annotation`**(*body: dict*, *error_callback: Callable[[str], None] = None*) → bool

Checks whether the given body is valid to create an annotation.

A valid body is a `dict` with two fields:

- `doc`: a list of valid doc annotations (see *`is_valid_doc_annotation()`*)
- `ner`: a list of valid NER annotations (see *`is_valid_ner_annotation()`*)

Parameters

- **`body`** (*`dict`*) – annotation body
- **`error_callback`** (*`function, optional`*) – optional callback that is called with any error messages, defaults to `None`

Returns whether the given body is valid to create an annotation

Return type bool

`pine.client.models.`**`is_valid_doc_annotations`**(*doc_annotations: dict*, *error_callback: Callable[[str], None] = None*) → bool

Checks whether the given document annotations are valid.

A valid document annotations object is a `dict`, where the keys are `str` document IDs, and the values are valid annotation bodies (see *`is_valid_annotation()`*).

Parameters

- **`doc_annotations`** – document annotations
- **`error_callback`** (*`function, optional`*) – optional callback that is called with any error messages, defaults to `None`

Returns whether the given body is valid to create an annotation

Return type bool

`class` `pine.client.models.`**`CollectionBuilder`**(*collection: dict = None*, *creator_id: str = None*, *viewers: List[str] = None*, *annotators: List[str] = None*, *labels: List[str] = None*, *title: str = None*, *description: str = None*, *allow_overlapping_ner_annotations: bool = None*, *pipelineId: str = None*, *overlap: float = None*, *train_every: int = None*, *classifierParameters: dict = None*, *document_csv_file: str = None*, *document_csv_file_has_header: bool = None*, *document_csv_file_text_column: int = None*, *image_files: List[str] = None*)

Bases: `object`

A class that can build the form and files fields that are necessary to create a collection.

Constructor.

Parameters

- **`collection`** (*`dict, optional`*) – starting parameters for the collection, defaults to `None` (not set)

- **creator_id** (*str, optional*) – user ID for the creator, see `creator_id()`, defaults to `None` (not set)
- **viewers** (*list(str), optional*) – viewer IDs for the collection, see `viewer()`, defaults to `None` (not set)
- **annotators** (*list(str), optional*) – annotator IDs for the collection, see `annotator()`, defaults to `None` (not set)
- **labels** (*list(str), optional*) – labels for the collection, see `label()`, defaults to `None` (not set)
- **title** (*str, optional*) – metadata title, see `title()`, defaults to `None` (not set)
- **description** (*str, optional*) – metadata description, see `description()`, defaults to `None` (not set)
- **allow_overlapping_ner_annotations** (*bool*) – optional configuration for allowing overlapping NER annotations, see `allow_overlapping_ner_annotations()`, defaults to `None` (not set)
- **pipelineId** (*str, optional*) – the ID of the pipeline from which to create the classifier, see `classifier()`, defaults to `None` (not set)
- **overlap** (*float, optional*) – the classifier overlap, see `classifier()`, defaults to `None` (not set)
- **train_every** (*int, optional*) – train the model after this many documents are annotated, see `classifier()`, defaults to `None` (not set)
- **classifierParameters** (*dict, optional*) – any parameters to pass to the classifier, see `classifier()`, defaults to `None` (not set)
- **document_csv_file** (*str, optional*) – the filename of the local document CSV file, see `document_csv_File()`, defaults to `None` (not set)
- **document_csv_file_has_header** (*bool, optional*) – whether the document CSV file has a header, see `document_csv_File()`, defaults to `None` (not set)
- **document_csv_file_text_column** (*int, optional*) – if the document CSV file has headers, the document text can be found in this column index (the others are used for document metadata), see `document_csv_File()`, defaults to `None` (not set)
- **image_files** (*list(str)*) – any image files to add to the collection, see `image_file()`, defaults to `None` (not set)

form

The form data.

Type dict

files

The files data.

Type dict

property collection(*self*) → dict

Returns the collection information from the form.

Returns collection information from the form

Return type dict

property form_json(*self*) → dict

Returns the form where the values have been JSON-encoded.

Returns the form where the values have been JSON-encoded

Return type dict

creator_id (*self*, *user_id: str*) → *pine.client.models.CollectionBuilder*
Sets the creator_id to the given, and adds to viewers and annotators.

Parameters **user_id** (*str*) – the user ID to use for the creator_id

Returns self

Return type *models.CollectionBuilder*

viewer (*self*, *user_id: str*) → *pine.client.models.CollectionBuilder*
Adds the given user to the list of viewers.

Parameters **user_id** (*str*) – the user ID to add as a viewer

Returns self

Return type *models.CollectionBuilder*

annotator (*self*, *user_id: str*) → *pine.client.models.CollectionBuilder*
Adds the given user to the list of annotators.

Parameters **user_id** (*str*) – the user ID to add as an annotator

Returns self

Return type *models.CollectionBuilder*

label (*self*, *label: str*) → *pine.client.models.CollectionBuilder*
Adds the given label to the collection.

Parameters **label** (*str*) – label to add

Returns self

Return type *models.CollectionBuilder*

metadata (*self*, *key: str*, *value: Any*) → *pine.client.models.CollectionBuilder*
Adds the given metadata key/value to the collection.

Parameters

- **key** (*str*) – metadata key
- **value** – metadata value

Returns self

Return type *models.CollectionBuilder*

title (*self*, *title: str*) → *pine.client.models.CollectionBuilder*
Sets the metadata title to the given.

Parameters **title** (*str*) – collection title

Returns self

Return type *models.CollectionBuilder*

description (*self*, *description: str*) → *pine.client.models.CollectionBuilder*
Sets the metadata description to the given.

Parameters **description** (*str*) – collection description

Returns self

Return type *models.CollectionBuilder*

`configuration`(*self*, *key: str*, *value: Any*) → *pine.client.models.CollectionBuilder*
Adds the given configuration key/value to the collection.

Parameters

- **`key`** (`str`) – configuration key
- **`value`** – configuration value

Returns self

Return type *models.CollectionBuilder*

`allow_overlapping_ner_annotations`(*self*, *allow_overlapping_ner_annotations: bool*)
Sets the configuration value for allow_overlapping_ner_annotations to the given.

Parameters **`allow_overlapping_ner_annotations`** (`bool`) – whether to allow overlapping NER annotations

Returns self

Return type *models.CollectionBuilder*

`classifier`(*self*, *pipelineId: str*, *overlap: float = 0*, *train_every: int = 100*, *classifierParameters: dict = {}*) → *pine.client.models.CollectionBuilder*
Sets classifier information for the created collection.

Parameters

- **`pipelineId`** (`str`) – the ID of the pipeline from which to create the classifier
- **`overlap`** (`float, optional`) – the classifier overlap, defaults to *0*
- **`train_every`** (`int, optional`) – train the model after this many documents are annotated, defaults to *100*
- **`classifierParameters`** (`dict, optional`) – any parameters to pass to the classifier, defaults to `{}`

Returns self

Return type *models.CollectionBuilder*

`document_csv_file`(*self*, *csv_filename: str*, *has_header: bool*, *text_column: int*) → *pine.client.models.CollectionBuilder*
Sets the CSV file used to create documents to the given.

May raise an Exception if there is a problem opening the indicated file.

Parameters

- **`csv_filename`** (`str`) – the filename of the local CSV file
- **`has_header`** (`bool`) – whether the CSV file has a header
- **`text_column`** (`int`) – if the CSV file has headers, the document text can be found in this column index (the others are used for document metadata)

Returns self

Return type *models.CollectionBuilder*

`image_file`(*self*, *image_filename: str*) → *pine.client.models.CollectionBuilder*
Adds the given image file to the collection.

May raise an Exception if there is a problem opening the indicated file.

Parameters **`image_filename`** (*str*) – the filename of the local image file

Returns self

Return type *models.CollectionBuilder*

`is_valid`(*self*, *error_callback: Callable[[str], None] = None*)

Checks whether the currently set values are valid or not.

See `is_valid_collection()`.

Parameters **`error_callback`** (*function, optional*) – optional callback that is called with any error messages, defaults to `None`

Returns whether the currently set values are valid or not

Return type bool

class `pine.client.models.`**`CollectionUserPermissions`**(*view=False*, *annotate=False*, *add_documents=False*, *add_images=False*, *modify_users=False*, *modify_labels=False*, *modify_document_metadata=False*, *download_data=False*, *archive=False*, *delete_documents=False*)

Bases: `object`

Collection permissions for a user as a dictionary of boolean flags.

`view :bool`

Whether the user can view the collection and documents. :type: bool

`annotate :bool`

Whether the user can annotate collection documents. :type: bool

`add_documents :bool`

Whether the user can add documents to the collection. :type: bool

`add_images :bool`

Whether the user can add images to the collection. :type: bool

`modify_users :bool`

Whether the user can modify the list of viewers/annotators for the collection. :type: bool

`modify_labels :bool`

Whether the user can modify the list of labels for the collection. :type: bool

`modify_document_metadata :bool`

Whether the user can modify document metadata (such as changing the image). :type: bool

`download_data :bool`

Whether the user can download the collection data :type: bool

`archive :bool`

Whether the user can archive or unarchive the collection. :type: bool

`delete_documents :bool`

Whether the user can delete documents in the collection. :type: bool

`to_dict`(*self*) → dict

Returns a dict version of this object for conversion to JSON.

Returns a dict version of this object for conversion to JSON

Return type dict

`pine.client.models.`**`remove_eve_fields`**`(`*obj: dict, remove_timestamps: bool = True, remove_versions: bool = True*`)`

Removes fields inserted by eve from the given object.

Parameters

- **`obj`** (*`dict`*) – the object
- **`remove_timestamps`** (*`bool`*) – whether to remove the timestamp fields, defaults to `True`
- **`remove_versions`** (*`bool`*) – whether to remove the version fields, defaults to `True`

`pine.client.models.`**`remove_nonupdatable_fields`**`(`*obj: dict*`)`

Removes all non-updatable fields from the given object.

These fields would cause a `PUT`/`PATCH` to be rejected because they are not user-modifiable.

Parameters **`obj`** (*`dict`*) – the object

`pine.client.password`

Module Contents

Functions

`hash_password`(password: str) → str	Hashes the given password for use in user object.
`check_password`(password: str, hashed_password: str) → str	Checks the given password against the given hash.

`pine.client.password.`**`hash_password`**`(`*password: str*`)` → str

Hashes the given password for use in user object.

Parameters **`password`** (*`str`*) – password

Returns hashed password

Return type str

`pine.client.password.`**`check_password`**`(`*password: str, hashed_password: str*`)` → str

Checks the given password against the given hash.

Parameters

- **`password`** (*`str`*) – password to check
- **`hashed_password`** (*`str`*) – hashed password to check against

Returns whether the password matches the hash

Return type bool

Package Contents

Classes

PineClient	A client to access PINE (more specifically: the backend).
LocalPineClient	A client for a local PINE instance, including an `EveClient`.
CollectionBuilder	A class that can build the form and files fields that are necessary to create a collection.

Functions

setup_logging()	Sets up logging, if configured to do so.

class pine.client.**PineClient**(*backend_base_uri: str*, *verify_ssl: bool = True*)

Bases: *pine.client.client.BaseClient*

A client to access PINE (more specifically: the backend).

Constructor.

Parameters

- **backend_base_uri** (*str*) – the base URI for the backend server, e.g. `"http://localhost:5000"`
- **verify_ssl** (*bool, optional*) – whether to verify SSL/HTTPs calls; do not turn this off unless you are fully aware of the security consequences

is_valid(*self*) → bool

Returns whether this client and its connection(s) are valid.

Returns whether this client and its connection(s) are valid

Return type bool

ping(*self*) → Any

Pings the backend server and returns the result.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the JSON response from the server (probably `"pong"`)

about(*self*) → dict

Returns the 'about' dict from the server.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the JSON response from the server

Return type dict

get_logged_in_user(*self*) → dict

Returns the currently logged in user, or None if not logged in.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns currently logged in user, or None if not logged in

Return type dict

get_my_user_id(*self*) → str

Returns the ID of the logged in user, or None if not logged in.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the ID of the logged in user, or None if not logged in

Return type str

is_logged_in(*self*) → bool

Returns whether the user is currently logged in or not.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns whether the user is currently logged in or not

Return type bool

_check_login(*self*)

Checks whether user is logged in and raises an `exceptions.PineClientAuthException` if not.

Raises

- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

get_auth_module(*self*) → str

Returns the PINE authentication module, e.g. `"eve"`.

Raises ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the PINE authentication module, e.g. `"eve"`

Return type str

login_eve(*self*, *username: str*, *password: str*) → bool

Logs in using eve credentials, and returns whether it was successful.

Parameters

- **username** (`str`) – username
- **password** (`str`) – password

Raises

- ***exceptions.PineClientAuthException*** – if auth module is not eve or login was not successful
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns whether the login was successful

Return type bool

`authorize_vegas`(*self*, *json_token: dict*) → bool

Logs in using a VEGAS token, and returns whether it was successful.

The token should be in the same format as is returned by VEGAS's oauth2/accesstoken endpoint, e.g. containing fields "access_token", "token_type", "expires_in", etc.

Parameters **`json_token`** (*`dict`*) – the token returned by VEGAS

Raises

- ***`exceptions.PineClientAuthException`*** – if auth module is not vegas or authorization was not successful
- ***`exceptions.PineClientHttpException`*** – if the HTTP request returns an error

Returns whether the authorization was successful

Return type bool

`logout`(*self*)

Logs out the current user.

Raises ***`exceptions.PineClientHttpException`*** – if the HTTP request returns an error

`get_pipelines`(*self*) → List[dict]

Returns all pipelines accessible to logged in user.

Raises

- ***`exceptions.PineClientAuthException`*** – if not logged in
- ***`exceptions.PineClientHttpException`*** – if the HTTP request returns an error

Returns all pipelines accessible to logged in user

Return type list(dict)

`get_pipeline_status`(*self*, *pipeline_id: str*) → dict

Returns status for the given pipeline.

Parameters **`pipeline_id`** – str: pipeline ID

Returns pipeline status

Return type dict

`collection_builder`(*self*, ***kwargs: dict*) → *pine.client.models.CollectionBuilder*

Makes and returns a new `models.CollectionBuilder` with the logged in user.

Parameters **`**kwargs`** (*`dict`*) – any additional args to pass in to the constructor

Returns a new `models.CollectionBuilder` with the logged in user

Return type *models.CollectionBuilder*

`create_collection`(*self*, *builder:* pine.client.models.CollectionBuilder) → str

Creates a collection using the current value of the given builder and returns its ID.

Parameters **`builder`** (`models.CollectionBuilder`) – collection builder

Raises

- ***`exceptions.PineClientValueException`*** – if the given collection is not valid, see `models.is_valid_collection()`
- ***`exceptions.PineClientAuthException`*** – if not logged in

- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the created collection's ID

Return type str

archive_collection (*self*, *collection_id: str*, *archive: bool = True*) → dict

Archives or unarchives the given collection.

Parameters

- **collection_id** – str: the ID of the collection
- **archive** – bool: whether to archive (True) or unarchive (False) the collection

Returns updated collection information

Return type dict

get_collection_permissions (*self*, *collection_id: str*) → *pine.client.models.CollectionUserPermissions*

Returns collection permissions for the logged in user.

Parameters **collection_id** (*str*) – the ID of the collection

Returns the collection permissions

Return type *models.CollectionUserPermissions*

get_collection_documents (*self*, *collection_id: str*, *truncate: bool*, *truncate_length: int = 30*) → List[dict]

Returns all the documents in the given collection.

Parameters

- **collection_id** (*str*) – the ID of the collection
- **truncate** (*bool*) – whether to truncate the document text (a good idea unless you need it)
- **truncate_length** (*int, optional*) – how many characters of the text you want if truncated, defaults to `30`

Returns all the documents in the given collection

Return type list(dict)

get_collection_classifier (*self*, *collection_id: str*) → dict

Returns the classifier associated with the given collection.

Parameters **collection_id** (*str*) – the ID of the collection

Returns the classifier associated with the given collection

Return type dict

get_next_document (*self*, *classifier_id: str*) → str

Returns the 'next' document associated with the given classifier.

The next document is the one that the model suggests should be annotated by the logged-in user next.

Parameters **classifier_id** – str: ID of the classifier

Returns the next document ID, or None if there are none left to annotate

Return type str

advance_next_document (*self*, *classifier_id: str*, *document_id: str*) → dict

Advances the 'next' document associated with the given classifier by marking the given document as annotated.

The next document is the one that the model suggests should be annotated by the logged-in user next.

Parameters

- **classifier_id** – str: ID of the classifier
- **document_id** – str: the ID of the document that was annotated

Returns information on the advanced instance

Return type dict

add_document (*self*, *document: dict = {}*, *creator_id: str = None*, *collection_id: str = None*, *overlap: int = None*, *text: str = None*, *metadata: dict = None*) → str

Adds a new document to a collection and returns its ID.

Will use the logged in user ID for the creator_id if none is given. Although all the parameters are optional, you must provide values either in the document or through the kwargs in order to make a valid document.

Parameters

- **document** (*dict, optional*) – optional document dict, will be overridden with any kwargs, defaults to `{}`
- **creator_id** (*str, optional*) – optional creator_id for the document, defaults to `None` (not set)
- **collection_id** (*str, optional*) – optional collection_id for the document, defaults to `None` (not set)
- **overlap** (*int, optional*) – optional overlap for the document, defaults to `None` (not set)
- **text** (*str, optional*) – optional text for the document, defaults to `None` (not set)
- **metadata** (*dict, optional*) – optional metadata for the document, defaults to `None` (not set)

Raises

- ***exceptions.PineClientValueException*** – if the given document parameters are not valid, see `models.is_valid_eve_document()`
- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the created document's ID

Return type str

add_documents (*self*, *documents: List[dict]*, *creator_id: str = None*, *collection_id: str = None*) → List[str]

Adds multiple documents at once and returns their IDs.

Will use the logged in user ID for the creator_id if none is given.

Parameters

- **documents** (*list(dict)*) – the documents to add
- **creator_id** (*str, optional*) – optional creator_id to set in the documents, defaults to `None` (not set)

- **collection_id** (*str, optional*) – optional collection_id to set in the documents, defaults to `None` (not set)

Raises

- ***exceptions.PineClientValueException*** – if any of the given documents are not valid, see *models.is_valid_eve_document()*
- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the created documents' IDs

Return type list(str)

delete_document (*self*, *document_id: str*) → dict

Deletes the document and associated data with the given ID.

Parameters **document_id** – str: ID of the document to delete

Returns information about deleted objects

Return type dict

delete_documents (*self*, *document_ids: List[str]*) → dict

Deletes the documents and associated data with the given IDs.

Parameters **document_ids** – list[str]: IDs of the documents to delete

Returns information about deleted objects

Return type dict

annotate_document (*self*, *document_id: str*, *doc_annotations: List[str]*, *ner_annotations: List[Union[dict, list, tuple]]*, *update_iaa: bool = True*) → str

Annotates the given document with the given values.

Parameters

- **document_id** (*str*) – the document ID to annotate
- **doc_annotations** (*list(str)*) – document annotations/labels
- **ner_annotations** (*list*) – NER annotations, where each annotation is either a list or a dict
- **update_iaa** – whether to also update IAA reports related to this document, defaults to *True*

Type update_iaa: bool

Raises

- ***exceptions.PineClientValueException*** – if any of the given annotations are not valid, see *models.is_valid_annotation()*
- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the annotation ID

Return type str

annotate_collection_documents(*self*, *collection_id*: *str*, *document_annotations*: *dict*, *skip_document_updates=False*, *update_iaa=True*) → List[str]

Annotates documents in a collection.

Parameters

- **collection_id** (*str*) – the ID of the collection containing the documents to annotate
- **document_annotations** (*dict*) – a dict containing "ner" list and "doc" list
- **skip_document_updates** (*bool*) – whether to skip updating the document "has_annotated" map, defaults to `False`. This should only be `True` if you properly set the "has_annotated" map when you created the document.
- **update_iaa** (*bool*) – whether to also update IAA report for the collection, defaults to `True`

Raises

- ***exceptions.PineClientValueException*** – if any of the given annotations are not valid, see *models.is_valid_doc_annotations()*
- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the annotation IDs

Return type list(str)

get_my_document_annotations(*self*, *document_id*: *str*) → List[List[dict]]

Returns annotations for the given document for the logged in user.

Parameters **document_id** – the ID of the document to get annotations for

Raises

- ***exceptions.PineClientValueException*** – if the document ID is not a valid string
- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the annotations for the given document for the logged in user

Return type list(list(dict))

get_others_document_annotations(*self*, *document_id*: *str*) → List[List]

Returns annotations for the given document for users other than the logged in user.

Parameters **document_id** – the ID of the document to get annotations for

Raises

- ***exceptions.PineClientValueException*** – if the document ID is not a valid string
- ***exceptions.PineClientAuthException*** – if not logged in
- ***exceptions.PineClientHttpException*** – if the HTTP request returns an error

Returns the annotations for the given document for users other than the logged in user

Return type list(list(dict))

`list_collections` (*self*, *include_archived: bool = False*) → List[dict]

Returns a list of user's collections.

Parameters **`include_archived`** (*`bool`*) – whether to include archived collections, defaults to `False`

Raises

- ***`exceptions.PineClientAuthException`*** – if not logged in
- ***`exceptions.PineClientHttpException`*** – if the HTTP request returns an error

Returns user's collections

Return type list(dict)

`get_collection` (*self*, *collection_id: str*) → dict

Returns the collection with the given ID.

Parameters **`collection_id`** – str: the ID of the collection

Returns the collection data

Return type dict

`get_collection_iaa_report` (*self*, *collection_id: str*) → dict

Returns IAA (inter-annotation agreement) report for the given collection.

Parameters **`collection_id`** – the ID of the collection for which to get report

Type collection_id: str

Returns report data

Return type dict

`download_collection_data` (*self*, *collection_id: str*, *include_collection_metadata: bool = True*, *include_document_metadata: bool = True*, *include_document_text: bool = True*, *include_annotations: bool = True*, *include_annotation_latest_version_only: bool = True*)

Downloads collection data.

Parameters

- **`collection_id`** (*`str`*) – the ID of the collection for which to download data
- **`include_collection_metadata`** (*`bool`*) – whether to include collection metadata, defaults to `True`
- **`include_document_metadata`** (*`bool`*) – whether to include document metadata, defaults to `True`
- **`include_document_text`** (*`bool`*) – whether to include document text, defaults to `True`
- **`include_annotations`** (*`bool`*) – whether to include annotations, defaults to `True`
- **`include_annotation_latest_version_only`** (*`bool`*) – whether to include only the latest version of annotations (`True`) or all versions (`False`), defaults to `True`

Raises

- ***`exceptions.PineClientValueException`*** – if given empty collection ID
- ***`exceptions.PineClientAuthException`*** – if not logged in
- ***`exceptions.PineClientHttpException`*** – if the HTTP request returns an error, such as if the collection doesn't exist

Returns collection data

Return type dict

get_classifier_status(*self*, *classifier_id: str*) → dict
Returns the status for the given classifier.

Param classifier_id: str: classifier ID

Returns status for the given classifier

Return type dict

classifier_train(*self*, *classifier_id: str*, *model_name: str = 'auto-trained'*) → dict
Trains the given classifier (using collection documents).

Note that training is done asynchronously, so this method should return very quickly. One of the things returned in the dict will be a job ID. If you want to know when the training has finished, you can periodically poll `get_classifier_running_jobs()` and check for that job ID.

Parameters

- **classifier_id** – str: classifier ID
- **model_name** – str: name of model corresponding to filename on disk, defaults to `"auto-trained"` which is the same as the annotation-based model training

Return type dict

classifier_has_trained(*self*, *classifier_id: str*) → bool
Returns whether the given classifier has been trained or not.

If False, future calls to predict will fail.

Param classifier_id: str: classifier ID

Return type bool

classifier_predict(*self*, *classifier_id*, *document_ids: List[str]*, *texts: List[str]*, *timeout_in_s: int = 36000*) → dict
Runs classifier prediction on the given documents. At least one of document_ids and texts must be non-empty.

This prediction uses the last-trained model name for that classifier. This method will block until the prediction has finished and then return the results.

Parameters

- **classifier_id** – str: classifier ID
- **document_ids** – list[str]: a list of document IDs to run prediction on
- **texts** – list[str]: a list of direct document texts to run prediction on
- **timeout_in_s** – int: max timeout in seconds before erroring out and returning, defaults to `36000`

Return type dict

get_classifier_running_jobs(*self*, *classifier_id: str*) → List[str]
Gets the list of running job IDs for the given classifier.

Parameters **classifier_id** – str: classifier ID

Return type list[str]

class pine.client.**LocalPineClient**(*backend_base_uri: str, eve_base_uri: str, mongo_base_uri: str = None, mongo_dbname: str = EveClient.DEFAULT_DBNAME, verify_ssl: bool = True*)

Bases: `pine.client.client.PineClient`

A client for a local PINE instance, including an `EveClient`.

Constructor.

Parameters

- **backend_base_uri** (*str*) – the base URI for the backend server, e.g. `"http://localhost:5000"`
- **eve_base_uri** (*str*) – the base URI for the eve server, e.g. `"http://localhost:5001"`
- **mongo_base_uri** (*str, optional*) – the base URI for the mongodb server, e.g. `"mongodb://localhost:27018"`, defaults to `None`
- **mongo_dbname** (*str, optional*) – the DB name that PINE uses, defaults to `"pmap_nlp"`
- **verify_ssl** (*bool, optional*) – whether to verify SSL/HTTPs calls; do not turn this off unless you are fully aware of the security consequences

eve :EveClient

The local `EveClient` instance.

Type *EveClient*

is_valid(*self*) → bool

Returns whether this client and its connection(s) are valid.

Returns whether this client and its connection(s) are valid

Return type bool

pine.client.**setup_logging**()

Sets up logging, if configured to do so.

The environment variable named by `CONFIG_FILE_ENV` is checked and, if present, is passed to `logging.config.dictConfig()`.

class pine.client.**CollectionBuilder**(*collection: dict = None, creator_id: str = None, viewers: List[str] = None, annotators: List[str] = None, labels: List[str] = None, title: str = None, description: str = None, allow_overlapping_ner_annotations: bool = None, pipelineId: str = None, overlap: float = None, train_every: int = None, classifierParameters: dict = None, document_csv_file: str = None, document_csv_file_has_header: bool = None, document_csv_file_text_column: int = None, image_files: List[str] = None*)

Bases: `object`

A class that can build the form and files fields that are necessary to create a collection.

Constructor.

Parameters

- **collection** (*dict, optional*) – starting parameters for the collection, defaults to `None` (not set)

- **`creator_id`** (*`str`, `optional`*) – user ID for the creator, see *`creator_id()`*, defaults to `None` (not set)
- **`viewers`** (*`list(str)`, `optional`*) – viewer IDs for the collection, see *`viewer()`*, defaults to `None` (not set)
- **`annotators`** (*`list(str)`, `optional`*) – annotator IDs for the collection, see *`annotator()`*, defaults to `None` (not set)
- **`labels`** (*`list(str)`, `optional`*) – labels for the collection, see *`label()`*, defaults to `None` (not set)
- **`title`** (*`str`, `optional`*) – metadata title, see *`title()`*, defaults to `None` (not set)
- **`description`** (*`str`, `optional`*) – metadata description, see *`description()`*, defaults to `None` (not set)
- **`allow_overlapping_ner_annotations`** (*`bool`*) – optional configuration for allowing overlapping NER annotations, see *`allow_overlapping_ner_annotations()`*, defaults to `None` (not set)
- **`pipelineId`** (*`str`, `optional`*) – the ID of the pipeline from which to create the classifier, see *`classifier()`*, defaults to `None` (not set)
- **`overlap`** (*`float`, `optional`*) – the classifier overlap, see *`classifier()`*, defaults to `None` (not set)
- **`train_every`** (*`int`, `optional`*) – train the model after this many documents are annotated, see *`classifier()`*, defaults to `None` (not set)
- **`classifierParameters`** (*`dict`, `optional`*) – any parameters to pass to the classifier, see *`classifier()`*, defaults to `None` (not set)
- **`document_csv_file`** (*`str`, `optional`*) – the filename of the local document CSV file, see `document_csv_File()`, defaults to `None` (not set)
- **`document_csv_file_has_header`** (*`bool`, `optional`*) – whether the document CSV file has a header, see `document_csv_File()`, defaults to `None` (not set)
- **`document_csv_file_text_column`** (*`int`, `optional`*) – if the document CSV file has headers, the document text can be found in this column index (the others are used for document metadata), see `document_csv_File()`, defaults to `None` (not set)
- **`image_files`** (*`list(str)`*) – any image files to add to the collection, see *`image_file()`*, defaults to `None` (not set)

`form`

The form data.

Type dict

`files`

The files data.

Type dict

`property collection`(*self*) → dict

Returns the collection information from the form.

Returns collection information from the form

Return type dict

`property form_json`(*self*) → dict

Returns the form where the values have been JSON-encoded.

Returns the form where the values have been JSON-encoded

Return type dict

`creator_id`(*self*, *user_id: str*) → *pine.client.models.CollectionBuilder*
Sets the creator_id to the given, and adds to viewers and annotators.

Parameters **`user_id`** (*`str`*) – the user ID to use for the creator_id

Returns self

Return type *models.CollectionBuilder*

`viewer`(*self*, *user_id: str*) → *pine.client.models.CollectionBuilder*
Adds the given user to the list of viewers.

Parameters **`user_id`** (*`str`*) – the user ID to add as a viewer

Returns self

Return type *models.CollectionBuilder*

`annotator`(*self*, *user_id: str*) → *pine.client.models.CollectionBuilder*
Adds the given user to the list of annotators.

Parameters **`user_id`** (*`str`*) – the user ID to add as an annotator

Returns self

Return type *models.CollectionBuilder*

`label`(*self*, *label: str*) → *pine.client.models.CollectionBuilder*
Adds the given label to the collection.

Parameters **`label`** (*`str`*) – label to add

Returns self

Return type *models.CollectionBuilder*

`metadata`(*self*, *key: str*, *value: Any*) → *pine.client.models.CollectionBuilder*
Adds the given metadata key/value to the collection.

Parameters

- **`key`** (*`str`*) – metadata key
- **`value`** – metadata value

Returns self

Return type *models.CollectionBuilder*

`title`(*self*, *title: str*) → *pine.client.models.CollectionBuilder*
Sets the metadata title to the given.

Parameters **`title`** (*`str`*) – collection title

Returns self

Return type *models.CollectionBuilder*

`description`(*self*, *description: str*) → *pine.client.models.CollectionBuilder*
Sets the metadata description to the given.

Parameters **`description`** (*`str`*) – collection description

Returns self

Return type *models.CollectionBuilder*

`configuration`(*self*, *key: str*, *value: Any*) → *pine.client.models.CollectionBuilder*
Adds the given configuration key/value to the collection.

Parameters

- **`key`** (`str`) – configuration key
- **`value`** – configuration value

Returns self

Return type *models.CollectionBuilder*

`allow_overlapping_ner_annotations`(*self*, *allow_overlapping_ner_annotations: bool*)
Sets the configuration value for allow_overlapping_ner_annotations to the given.

Parameters **`allow_overlapping_ner_annotations`** (`bool`) – whether to allow overlapping NER annotations

Returns self

Return type *models.CollectionBuilder*

`classifier`(*self*, *pipelineId: str*, *overlap: float = 0*, *train_every: int = 100*, *classifierParameters: dict = {}*) → *pine.client.models.CollectionBuilder*
Sets classifier information for the created collection.

Parameters

- **`pipelineId`** (`str`) – the ID of the pipeline from which to create the classifier
- **`overlap`** (`float, optional`) – the classifier overlap, defaults to *0*
- **`train_every`** (`int, optional`) – train the model after this many documents are annotated, defaults to *100*
- **`classifierParameters`** (`dict, optional`) – any parameters to pass to the classifier, defaults to `{}`

Returns self

Return type *models.CollectionBuilder*

`document_csv_file`(*self*, *csv_filename: str*, *has_header: bool*, *text_column: int*) → *pine.client.models.CollectionBuilder*
Sets the CSV file used to create documents to the given.

May raise an Exception if there is a problem opening the indicated file.

Parameters

- **`csv_filename`** (`str`) – the filename of the local CSV file
- **`has_header`** (`bool`) – whether the CSV file has a header
- **`text_column`** (`int`) – if the CSV file has headers, the document text can be found in this column index (the others are used for document metadata)

Returns self

Return type *models.CollectionBuilder*

`image_file`(*self*, *image_filename: str*) → *pine.client.models.CollectionBuilder*
Adds the given image file to the collection.

May raise an Exception if there is a problem opening the indicated file.

Parameters **`image_filename`** (*`str`*) – the filename of the local image file

Returns self

Return type *models.CollectionBuilder*

`is_valid`(*self, error_callback: Callable[[str], None] = None*)
Checks whether the currently set values are valid or not.

See *`is_valid_collection()`*.

Parameters **`error_callback`** (*`function, optional`*) – optional callback that is called with any error messages, defaults to `None`

Returns whether the currently set values are valid or not

Return type bool

`pine.pipelines`

Subpackages

`pine.pipelines.app`

Subpackages

`pine.pipelines.app.listener`

Submodules

`pine.pipelines.app.listener.main`

Module Contents

Functions

`backoff_hdlr`(details)	
`setup_logging`()	
`run`()	

`pine.pipelines.app.listener.main.`**`backoff_hdlr`**(*details*)

`pine.pipelines.app.listener.main.`**`setup_logging`**()

`pine.pipelines.app.listener.main.`**`run`**()

pine.pipelines.app.listener.service_listener

Module Contents

Classes

ServiceRegistration	
ServiceListener	**type services** list[ServiceRegistration]

pine.pipelines.app.listener.service_listener.**config**

pine.pipelines.app.listener.service_listener.**logger**

class pine.pipelines.app.listener.service_listener.**ServiceRegistration**(*name*, *version*, *channel*, *framework*, *framework_types*)

Bases: object

classmethod from_registration_format(*cls*, ***kwargs*)

to_registration_format(*self*)

class pine.pipelines.app.listener.service_listener.**ServiceListener**(*services=None*)

Bases: object

r_pool

r_conn

registration_poll

listener_poll

processor_poll

processing_limit

processing_queue_key

processing_queue_key_timeout

results_queue_key

results_queue_key_timeout

running_jobs_key

classifiers_training_key

processing_lock_key

processing_lock_key_timeout

preprocessing_lock_key

preprocessing_lock_key_timeout

preprocessing_worker_lock_key

preprocessing_worker_lock_key_timeout

registration_channel

start_workers(*self*)

stop_workers(*self*)

pre_process_message(*self*, *message_channel*, *message_data*)

Return type bool | dict

static do_with_redis(*callback: Callable[[redis.StrictRedis], typing.Any]*)

static push_results(*job_id: str*, *response*)

static wait_until_classifier_isnt_training(*classifier_id: str*, *job_id: str*)

static classifier_is_done_training(*classifier_id: str*)

static process_message(*job_id: str*, *job_details*)

_start_registration_task(*self*)

_start_channel_task(*self*)

_start_listener_task(*self*)

_start_queue_processor_task(*self*)

pine.pipelines.shared

Submodules

pine.pipelines.shared.config

Module Contents

Classes

BaseConfig
TestConfig
ConfigBuilder

pine.pipelines.shared.config.**LOGGER**

class pine.pipelines.shared.config.**BaseConfig**(*root_dir=None*)
Bases: object

ROOT_DIR

BASE_DIR

BASE_CFG_FILE = config.yaml

BASE_ENV_PREFIX = AL_

DEBUG = True

TESTING = False

LOGGER_NAME

LOGGER_DIR = logs

LOGGER_FILE = debug.log

LOGGER_LEVEL

PIPELINE = opennlp

EVE_HOST = localhost

EVE_PORT = 5001

REDIS_HOST = localhost

REDIS_PORT = 6379

REDIS_USR

REDIS_PWD

REDIS_DBNUM = 0

REDIS_PREFIX = AL:

REDIS_EXPIRE = 3600

REDIS_MAX_PROCESSES = 10

SCHEDULER_REGISTRATION_TIMEOUT

SCHEDULER_HANDLER_TIMEOUT

SCHEDULER_QUEUE_TIMEOUT

SERVICE_REGISTRATION_CHANNEL = registration

SERVICE_REGISTRATION_FREQUENCY = 60

SERVICE_LISTENING_FREQUENCY = 1

SERVICE_HANDLER_TIMEOUT = 60

SERVICE_LIST

MODELS_DIR

classmethod _get_config_var_paths(*cls*, *root_dict=None*)

classmethod _process_paths(*cls*, *alt_path=None*)

classmethod _process_file_cfg(*cls*)

classmethod _process_env_vars(*cls*)

classmethod as_dict(*cls*)

Return type dict

classmethod as_attr_dict(*cls*)

Return type munch.Munch | dict

static _try_cast(*value*, *_type*, *_default=None*)

static _str2bool(*_str*, *_default=None*)

class pine.pipelines.shared.config.**TestConfig**(*root_dir=None*)

Bases: *pine.pipelines.shared.config.BaseConfig*

class pine.pipelines.shared.config.**ConfigBuilder**

Bases: object

__env_cfg_variable = BUILDER_CFG_PROFILE

__current_config_instance

__current_config_instance_name

__current_config_instance_print = False

__arg_parser

static __get_configs()

:rtype list[Callable[. . . , BaseConfig]]

static get_config_names()

Return type list[str]

classmethod get_arg_parser(*cls*)

Return type ArgumentParser

classmethod init_config(*cls*, *config_name=None*, *config_base=None*, *enable_terminal=True*, *as_attr_dict=True*)

classmethod get_config(*cls*, *config_name=None*, *config_base=None*, *enable_terminal=True*, *as_attr_dict=True*)

Return type *BaseConfig*

classmethod set_config(*cls*, *config_name=None*, *config_base=None*, *enable_terminal=True*, *as_attr_dict=True*)

classmethod __parse_terminal_config(*cls*)

Return type argparse.Namespace

pine.pipelines.shared.transform

Module Contents

Functions

transform_module_by_config(module_ref, config_ref, config_prefix=None)	Transforms a given module's properties based on ConfigBuilder Values.

`pine.pipelines.shared.transform.`**`transform_module_by_config`**(*module_ref*, *config_ref*, *config_prefix=None*)

Transforms a given module's properties based on ConfigBuilder Values. The prefix can be used to avoid blindy changing values and target a subset of matching values in config_ref. :type module_ref: ModuleType :type config_ref: dict :type config_prefix: str

Submodules

`pine.pipelines.EveClient`

Module Contents

Classes

`EveClient`

`pine.pipelines.EveClient.`**`logger`**

`pine.pipelines.EveClient.`**`config`**

`class` `pine.pipelines.EveClient.`**`EveClient`**(*entry_point='{}:{}'.format(config.EVE_HOST, config.EVE_PORT)*)

Bases: `object`

`eve_headers`

`add`(*self*, *resource*, *add_object*)

`get_obj`(*self*, *resource*, *id*)

`get_all_items`(*self*, *resource*)

`get_all_ids`(*self*, *resource*)

`get_items`(*self*, *resource*, *params={}*)

`_get_documents_map`(*self*, *params: dict = {}*)

`get_documents`(*self*, *collection_id: str*) → Dict[str, str]

Returns a document map where the document overlap is 0.

Parameters **`collection_id`** – str: the ID of the collection

Returns a mapping from document ID to document text for non-overlap documents

Return type dict

`get_documents_by_id`(*self*, *document_ids: List[str]*)

`get_docs_with_annotations`(*self*, *collection_id: str*, *doc_map: Dict[str, str]*) → Tuple[typing.List[str], typing.List[str], typing.List[str], typing.List[str]]

Gets document and annotation data. Only non-overlapping documents are returned.

Parameters

- **`collection_id`** – str: the ID of the collection

- **doc_map** – dict[str, str]: map of document IDs to document text

Returns (documents, labels, doc_ids, ann_ids) where documents is a list of the texts, labels is a list of the annotations, doc_ids is a list of the document IDs, and ann_ids is a list of the annotation IDs

Return type tuple

update(*self*, *resource*, *id*, *etag*, *update_obj*)

pine.pipelines.NERWrapper

Module Contents

Classes

NERWrapper

class pine.pipelines.NERWrapper.**NERWrapper**(*classifier_type*, *model_dir='models'*, *entry_point='localhost:5000'*)

eve_headers

load_classifier(*self*, *classifier_id*)

predict(*self*, *classifier_id*, *documents*, *document_ids*)

update_model(*self*, *classifier_id*)

update(*self*, *resource*, *id*, *etag*, *update_obj*)

get_obj(*self*, *resource*, *id*)

get_all_items(*self*, *resource*)

get_all_ids(*self*, *resource*)

get_items(*self*, *resource*)

pine.pipelines.NERWrapper.**parser**

pine.pipelines.NER_API

Module Contents

Classes

ner_api

pine.pipelines.NER_API.**logger**

pine.pipelines.NER_API.**config**

class pine.pipelines.NER_API.**ner_api**

Bases: `object`

`status`(*self*, *classifier_id: str*, *pipeline_name: str*) → dict

`perform_fold`(*self*, *model: pine.pipelines.pmap_ner.NER*, *train_data*, *test_data*, ***pipeline_parameters*)

`perform_five_fold`(*self*, *model: pine.pipelines.pmap_ner.NER*, *documents*, *annotations*, *doc_ids*, ***pipeline_parameters*)

`get_document_ranking`(*self*, *model: pine.pipelines.pmap_ner.NER*, *doc_map: Dict[str, str]*, *doc_ids: List[str]*) → List[str]

Calculates document rankings and returns document IDs sorted by ranking.

The ranking should be which documents should be evaluated first. This probably corresponds in some ways to the documents which the model is least confident about.

Parameters

- **`model`** – NER model
- **`doc_map`** – dict: mapping of document IDs to document text where overlap is 0
- **`doc_ids`** – list: IDs of documents where ???

Returns sorted document IDs

Return type list

`get_classifier_pipeline_metrics_objs`(*self*, *classifier_id*)

`train_model`(*self*, *custom_filename*, *classifier_id*, *pipeline_name*)

`predict`(*self*, *classifier_id: str*, *pipeline_name: str*, *document_ids: List[str]*, *texts: List[str]*)

`pine.pipelines.RankingFunctions`

Module Contents

Functions

`rank`(document_ids: List[str], results: List[DocumentPredictionProbabilities], metric: str) → List[Tuple[str, float]]	if metric == ‘lc’: return least_confidence(results)
`least_confidence`(document_ids: List[str], results: List[DocumentPredictionProbabilities]) → List[Tuple[str, float]]	
`least_confidence_squared`(document_ids: List[str], results: List[DocumentPredictionProbabilities]) → List[Tuple[str, float]]	
`least_confidence_squared_by_entity`(document_ids: List[str], results: List[DocumentPredictionProbabilities]) → List[Tuple[str, float]]	
`largest_margin`(document_ids: List[str], results: List[DocumentPredictionProbabilities]) → List[Tuple[str, float]]	

continues on next page

Table 55 – continued from previous page

`entropy_rank`(document_ids: List[str], results: List[DocumentPredictionProbabilities], N=None) → List[Tuple[str, float]]	
`random_rank`(document_ids: List[str], results: List[DocumentPredictionProbabilities]) → List[Tuple[str, float]]	
`most_of_least_popular`(document_ids: List[str], results: List[DocumentPredictionProbabilities]) → List[Tuple[str, float]]	

`pine.pipelines.RankingFunctions.`**`logger`**

`pine.pipelines.RankingFunctions.`**`rank`**(*document_ids: List[str], results: List[DocumentPredictionProbabilities], metric: str*) → List[Tuple[str, float]]

if metric == 'lc': return least_confidence(results) if metric == 'ma': return largest_margin(results) if metric == 'en': return entropy_rank(results) if metric == 'lcs': return least_confidence_squared(results) if metric == 'lce': return least_confidence_squared_by_entity(results) if metric == 'ra': return random_rank(results) if metric == 'mlp': return most_of_least_popular(results) return -1

#Dictionary method is inefficient as it runs every method before returning one

`pine.pipelines.RankingFunctions.`**`least_confidence`**(*document_ids: List[str], results: List[DocumentPredictionProbabilities]*) → List[Tuple[str, float]]

`pine.pipelines.RankingFunctions.`**`least_confidence_squared`**(*document_ids: List[str], results: List[DocumentPredictionProbabilities]*) → List[Tuple[str, float]]

`pine.pipelines.RankingFunctions.`**`least_confidence_squared_by_entity`**(*document_ids: List[str], results: List[DocumentPredictionProbabili* → List[Tuple[str, float]]

`pine.pipelines.RankingFunctions.`**`largest_margin`**(*document_ids: List[str], results: List[DocumentPredictionProbabilities]*) → List[Tuple[str, float]]

`pine.pipelines.RankingFunctions.`**`entropy_rank`**(*document_ids: List[str], results: List[DocumentPredictionProbabilities], N=None*) → List[Tuple[str, float]]

`pine.pipelines.RankingFunctions.`**`random_rank`**(*document_ids: List[str], results: List[DocumentPredictionProbabilities]*) → List[Tuple[str, float]]

`pine.pipelines.RankingFunctions.`**`most_of_least_popular`**(*document_ids: List[str], results: List[DocumentPredictionProbabilities]*) → List[Tuple[str, float]]

pine.pipelines.corenlp_NER_pipeline

Module Contents

Classes

corenlp_NER

pine.pipelines.corenlp_NER_pipeline.**config**

pine.pipelines.corenlp_NER_pipeline.**logger**

class pine.pipelines.corenlp_NER_pipeline.**corenlp_NER**(*java_dir=None*, *ner_path=None*, *load_model=None*, *tmp_dir=None*)

Bases: *pine.pipelines.pipeline.Pipeline*

__jar =

__jdk_dir =

__train_file =

__test_file =

__model =

__temp_dir

__crf

__props

__is_setup = False

__id

__default_fit_params

classmethod setup(*cls*, *java_dir=None*, *ner_path=None*)

status(*self*) → dict

fit(*self*, *X*, *y*, ***params*)

predict(*self*, *X: Iterable[str]*) → List[DocumentPredictions]

predict_proba(*self*, *X: Iterable[str]*, *get_all_labels=False*, *include_other=False*, ***kwargs*) → List[DocumentPredictionProbabilities]

next_example(*self*, *X*, *Xid*)

get_id(*self*)

format_data(*self*, *X*, *y*)

save_model(*self*, *model_name*)

load_model(*self*, *model_name*)

tokenize(*self*, *input_text*)

evaluate(*self*, *X*, *y*, *Xid*, *verbose=False*)

evaluate_orig (*self, X, y, Xid*)

pine.pipelines.opennlp_NER_pipeline

Module Contents

Classes

opennlp_NER

pine.pipelines.opennlp_NER_pipeline.**config**

pine.pipelines.opennlp_NER_pipeline.**logger**

class pine.pipelines.opennlp_NER_pipeline.**opennlp_NER** (*java_dir=None, ner_path=None, tmp_dir=None*)

Bases: *pine.pipelines.pipeline.Pipeline*

__ner_path =

__jdk_dir =

__temp_dir

__train_file =

__test_file =

__model

__nameFinder

__id

__is_setup = False

classmethod setup (*cls, java_dir=None, ner_path=None*)

status (*self*) → dict

fit (*self, X, y, **params*)

predict (*self, X: Iterable[str]*) → List[DocumentPredictions]

predict_proba (*self, X: Iterable[str], **kwargs*) → List[DocumentPredictionProbabilities]

next_example (*self, X, Xid*)

save_model (*self, model_name*)

load_model (*self, model_name*)

find_all_init (*self*)

find_all (*self, tokens*)

get_id (*self*)

format_data (*self, X, y*)

convert_ann_collection_to_per_token (*self*, *annotations: List[Union[NerPrediction, typing.Tuple[int, int, str]]]*, *tokens*)

evaluate (*self*, *X*, *y*, *Xid*)

evaluate_orig (*self*, *X*, *y*, *Xid*)

`pine.pipelines.pipeline`

Module Contents

Classes

NerPrediction
DocumentPredictions
NerPredictionProbabilities
DocumentPredictionProbabilities
Pipeline

class pine.pipelines.pipeline.**NerPrediction** (*offset_start: int*, *offset_end: int*, *label: str*)

Bases: `object`

serialize (*self*) → Tuple[int, int, str]

class pine.pipelines.pipeline.**DocumentPredictions** (*ner: List[NerPrediction]*, *doc: List[str]*, *extra_data: Any = None*)

Bases: `object`

serialize (*self*) → dict

class pine.pipelines.pipeline.**NerPredictionProbabilities** (*offset_start: int*, *offset_end: int*, *predictions: List[Tuple[str, float]]*)

Bases: `object`

get_highest_prediction (*self*) → Tuple[str, float]

get_predictions_from_highest_to_lowest (*self*) → List[Tuple[str, float]]

serialize (*self*) → Tuple[int, int, typing.List[typing.Tuple[str, float]]]

class pine.pipelines.pipeline.**DocumentPredictionProbabilities** (*ner: List[NerPredictionProbabilities]*, *doc: List[Tuple[str, float]]*)

Bases: `object`

serialize (*self*) → dict

class pine.pipelines.pipeline.**Pipeline**

Bases: `object`

abstract status(*self*) → dict

abstract fit(*self*, *X*, *y*, ***params*)

abstract predict(*self*, *X: Iterable[str]*) → List[DocumentPredictions]

abstract predict_proba(*self*, *X: Iterable[str]*, ***kwargs*) → List[DocumentPredictionProbabilities]

abstract next_example(*self*, *X*, *Xid*)

abstract save_model(*self*, *model_name*)

abstract load_model(*self*, *model_name*)

pine.pipelines.pmap_ner

Module Contents

Classes

NER

pine.pipelines.pmap_ner.**logger**

class pine.pipelines.pmap_ner.**NER**(*lib=None*, ***kwargs*)

Bases: *pine.pipelines.pipeline.Pipeline*

__lib =

pipeline

__SUPPORTED_PIPELINES = ['spacy', 'corenlp', 'opennlp']

pipe_init(*self*, *x*, ***kwargs*)

property pipeline_class(*self*)

status(*self*) → dict

fit(*self*, *X*, *y*, ***kwargs*)

predict(*self*, *X: Iterable[str]*) → List[DocumentPredictions]

predict_proba(*self*, *X: Iterable[str]*, ***kwargs*) → List[DocumentPredictionProbabilities]

evaluate(*self*, *X*, *y*, *Xid*, ***kwargs*)

next_example(*self*, *X*, *Xid*)

save_model(*self*, *model_name*)

load_model(*self*, *model_name*)

pine.pipelines.run_service

pine.pipelines.spacy_NER_pipeline

For more details, see the documentation: * Training: https://spacy.io/usage/training * NER: https://spacy.io/usage/linguistic-features#named-entities

Compatible with: spaCy v2.0.0+

Module Contents

Classes

spacy_NER

pine.pipelines.spacy_NER_pipeline.**logger**

class pine.pipelines.spacy_NER_pipeline.**spacy_NER**(*model_path=None*)

Bases: *pine.pipelines.pipeline.Pipeline*

__model

__nlp = []

__ner = []

__optimizer = []

__default_fit_params

_load_model(*self*, *model_path=None*)

status(*self*) → dict

fit(*self*, *X*, *y*, ***params*)

evaluate(*self*, *X*, *y*, *Xid*)

predict(*self*, *X: Iterable[str]*) → List[DocumentPredictions]

predict_proba(*self*, *X: Iterable[str]*, ***kwargs*) → List[DocumentPredictionProbabilities]

next_example(*self*, *X*, *Xid*)

format_data(*self*, *X*, *y*)

add_label(*self*, *entity*)

save_model(*self*, *model_name*)

load_model(*self*, *model_name*)

CHAPTER TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols