Blackboard Sync

Release 0.9.11

Jacob Sánchez

Sep 26, 2023

CONTENTS

BlackboardSync is a multiplatform desktop application written in Python that automatically downloads content from courses in your Blackboard account.

CHAPTER

ONE

FEATURES

- Supported content:
  - Attachments of any type (e.g. .docx, .pptx, .pdf, etc.)
  - Internet links
  - Content descriptions (saved as html)
- Cross-platform
  - Linux, Windows, and macOS ready

CHAPTER

TWO

INSTALLATION

You can find all releases on GitHub. Only MacOS (.dmg) and Windows (.exe) are supported at the moment.

2.1 From PyPI

```
$ python3 -m pip install blackboardsync
$ python3 -m blackboard_sync # notice the underscore
```

CHAPTER

THREE

API DOCUMENTATION

If you are interested about contributing, or if you just want to understand the internals of BlackboardSync more.

3.1 Blackboard API Reference

3.1.1 Blackboard REST API

REST API call parameters must be specified in keyword form. Blackboard API.

an interface to make Blackboard REST API calls on a session basis

class blackboard_sync.blackboard.api.**BlackboardSession**(*base_url: str*, *cookies: RequestsCookieJar*)

Represents a user session in Blackboard.

property base_url: str

API base URL.

download(*response*)

Download the contents of a Content Item.

Parameters

- **course_id** – The course or organization ID.
- **content_id** – The Content ID.
- **attachment_id** –

download_attempt_file_metadata(*response*)

Download the contents of the file for a Student Submission.

Parameters

- **course_id** – The course or organization ID.
- **attempt_id** –
- **attempt_file_id** –

download_webdav(*response*)

Downloads an arbitrary webdav file

fetch_announcements(*response*)

Return a list of System Announcements.

Parameters

announcement_id – Get a System Announcement.

fetch_associated_nodes(*response*)

Obtain a list of nodes to which a given course is directly associated.

Parameters
course_id – The course or organization ID.

fetch_attempt_file_metadata(*response*)

Get the list of file metadata for a Submission associated to the course and attempt.

Get the file metadata for a Student Submission associated to the course and attempt.

Parameters

- course_id – The course or organization ID.
- attempt_id –
- attempt_file_id –

fetch_attendance_data_download_url(*response*)

Generate Download URL for Attendance Data.

Parameters
course_id – The course or organization ID.

fetch_attendance_records_by_meeting_id(*response*)

Return a Course Meeting Attendance information for the given meeting and user Id.

Parameters

- course_id – The course or organization ID.
- meeting_id –
- user_id – The user ID.

fetch_attendance_records_by_user_id(*response*)

Return a list of Course Meeting Attendance for a given user id regardless of courses and meetings.

Parameters

- course_id – The course or organization ID.
- user_id – The user ID.

fetch_avatar(*response*)

Get a user avatar image.

The response is an HTTP redirect rather than image raw data. It is up to the caller of the api to follow the redirect and download the image.

Not yet implemented. :param user_id: The user ID.

fetch_calendar(*response*)

Get the list of calendars. This endpoint will return all calendars viewable by the user.

All users can request a list of calendars viewable to them.

fetch_calendar_items(*response*)

Get a course calendar item.

Parameters

- calendar_item_type – One of (Course, GradebookColumn, Institution, OfficeHours, Personal).

- **calendar_item_id** –

fetch_categories(*response*)

Get categories associated with the provided course.

Parameters
course_id – The course or organization ID.

fetch_category(*response*)

Return a list of categories of the provided type (course or organization).

/ Return the category corresponding the provided type (course or organization) and ID.

Parameters

- **category_type** – One of (Course, Organization).
- **category_id** –

fetch_child_categories(*response*)

Return a list of categories which are children of the category…

corresponding to the provided type (course or organization) and Id

Parameters

- **category_type** – One of (Course, Organization).
- **parent_id** –

fetch_column_attempts(*response*)

Return a list of attempts associated with the specified grade column.

/ Load the grade column attempt for the specified id.

Parameters

- **course_id** – The course or organization ID.
- **column_id** – The grade column ID.
- **attempt_id** –

fetch_column_grade_last_changed(*response*)

Load the grade column grade with the maximum change index.

Parameters

- **course_id** – The course or organization ID.
- **column_id** – The grade column ID.

fetch_column_grades(*response*)

Return a list of grades associated with the specified grade column.

/ Load the grade column grade for a specific user.

Parameters

- **course_id** – The course or organization ID.
- **column_id** – The grade column ID.
- **user_id** – The user ID.

fetch_content_children(*response*) → list[*BBContentChild*]

List all child content items directly beneath another content item.

This is only valid for content items that are allowed to have children (e.g. Folders).

Parameters

- **course_id** – The course or organization ID.
- **content_id** – The Content ID.

fetch_content_groups(*response*)

Return a list of content group associations for the specified content.

Parameters

- **course_id** – The course or organization ID.
- **content_id** – The Content ID.
- **group_id** – The group ID.

fetch_contents(*response*) → list[*BBCourseContent*]

List top-level content items in a course.

Parameters

- **course_id** – The course or organization ID.
- **content_id** – The Content ID.

fetch_course_announcements(*response*)

Return a list of Course Announcements or Get a Course Announcement.

Parameters

- **course_id** – The course or organization ID.
- **announcement_id** –

fetch_course_children(*response*)

Return a list of course cross-listings.

/ Load a specific course cross-listing.

Parameters

- **course_id** – The course or organization ID.
- **child_course_id** – The course or organization ID.

fetch_course_meeting(*response*)

Return a Course Meeting for the given meeting Id.

Parameters

- **course_id** – The course or organization ID.
- **meeting_id** –

fetch_course_meetings(*response*)

Return a list of course meetings for a given course id.

Parameters

course_id – The course or organization ID.

fetch_course_memberships(*response*)

Return a list of user memberships for the specified course or organization.

/ Load a user membership in the specified course.

Parameters

- **course_id** – The course or organization ID.
- **user_id** – The user ID.

fetch_course_resource_children(*response*)

Return a list of Course Resources that are children of the specified Resource.

Parameters

- **course_id** – The course or organization ID.
- **resource_id** – The xythos resource ID.

fetch_course_resources(*response*)

Return a list of the top-level course resources.

or Load a Course Resource by Id.

Parameters

- **course_id** – The course or organization ID.
- **resource_id** – The xythos resource ID.

fetch_course_roles(*response*)

Return a list of course roles. / Return a single course role.

Parameters
role_id – The course role ID.

fetch_courses(*response*) → *BBCourse* | list[*BBCourse*]

Return a list of courses and organizations.

/ Loads a specific course or organization.

Parameters
course_id – The course or organization ID.

fetch_cross_list_set(*response*)

Return the course cross-listing set for the specified course.

This will return any and all parent/child associations regardless of the specified course being a parent or child course. The result will be empty if the specified course is not cross-listed.

Parameters
course_id – The course or organization ID.

fetch_current_active_user(*response*)

Display active session information for a specific user.

Parameters
user_id – The user ID.

fetch_data_sources(*response*)

Return a list of data sources.

Parameters
data_source_id – The data source ID.

fetch_domain_config(*response*)

Return the list of LTI domain configs.

/ This endpoint returns the LTI domain config with the specified Id.

Parameters
domain_id –

fetch_file_attachments(*response*) → list[*BBAttachment*] | *BBAttachment*

Get the file attachment meta data associated to the Content Item.

or Get the file attachment meta data by an attachment ID.

Parameters

- **course_id** – The course or organization ID.
- **content_id** – The Content ID.
- **attachment_id** –

fetch_grade_columns(*response*)

Return a list of grade columns. / Load a specific grade column.

Parameters

- **course_id** – The course or organization ID.
- **column_id** – The grade column ID.

fetch_grade_notations(*response*)

Return a list of grade notations. / Return a specific grade notation.

Parameters
course_id – The course or organization ID.

Grade_notation_id

fetch_grade_schemas(*response*)

Return a list of grade schemas associated with the specified course.

/ Load the grade schema associated with the specified course and schema Id.

Parameters

- **course_id** – The course or organization ID.
- **schema_id** – The grade schema ID.

fetch_gradebook_categories(*response*)

Return a list of gradebook categories in a particular course.

/ Return the details of a gradebook category.

Parameters

- **course_id** – The course or organization ID.
- **category_id** – the ID of the category to return

fetch_grading_periods(*response*)

Return a list of grading periods. / Return a specific grading period.

Parameters

- **course_id** – The course or organization ID.

- **period_id** –

fetch_group_memberships(*response*)

Return a list of group memberships objects for the specified group.

/ Loads a group membership in the specified group.

Parameters

- **course_id** – The course or organization ID.
- **group_id** – The group ID.
- **user_id** – The user ID.

fetch_group_set_children(*response*)

Return a list of all groups within a groupset.

Parameters

- **course_id** – The course or organization ID.
- **group_id** – The group ID.

fetch_group_sets(*response*)

Return a list of all groupsets / Load a groupset in the specified course.

Parameters

- **course_id** – The course or organization ID.
- **group_id** – The group ID.

fetch_groups(*response*)

Return a list of all top-level groups in the specified course.

/ Load a group in the specified course.

Parameters

- **course_id** – The course or organization ID.
- **group_id** – The group ID.

fetch_institution_roles(*response*)

Return a list of institution roles. / Load a specific institution role.

Parameters
role_id – The institution role ID.

fetch_memberships(*response*)

Get courses associated with the provided category.

Parameters

- **category_type** – One of (Course, Organization).
- **category_id** –

fetch_node_children(*response*)

Return the children of the institutional hierarchy node corresponding…

to the provided ID.

Parameters
node_id – The node ID.

fetch_node_course_associations(*response*)

Return a list of node-course relationships for the specified node.

Parameters
node_id – The node ID.

fetch_nodes(*response*)

Return the Top-level institutional hierarchy nodes.

/ Return the institutional hierarchy node corresponding the provided ID.

Parameters
node_id – The node ID.

fetch_observees(*response*)

Return a list of users being observed by a given user.

Parameters
user_id – The user ID.

fetch_observers(*response*)

Return a list of users observing a given user.

Parameters
user_id – The user ID.

fetch_performance_review_status(*response*)

List the content review statuses for all the users enrolled in a course.

Parameters
course_id – The course or organization ID.

fetch_placements(*response*)

Return a list of LTI placements.

/ Returns the LTI placement with the specified Id.

Parameters
placement_id –

fetch_policies(*response*)

Return the links to the Blackboard and Institution privacy policies.

fetch_proctoring_services(*response*)

Return a list of proctoring service.

/ Return the proctoring service with the specified Id.

Parameters
service_id –

fetch_questions(*response*)

Get the list of questions for an Ultra Assessment or Get a question by Id from it.

Parameters

- **course_id** – The course or organization ID.
- **assessment_id** –
- **question_id** –

fetch_review_status(*response*)

Obtain the review status for a content item.

This endpoint will only fetch the reviewStatus if the corresponding content was previously marked as reviewable.

Parameters

- **course_id** – The course or organization ID.
- **content_id** – The Content ID.
- **user_id** – The user ID.

fetch_sessions(*response*)

List active user sessions in Learn.

fetch_sis_logs(*response*)

Return a list of SIS Integration logs.

Parameters
id – dataSetUid of the integration

fetch_system_roles(*response*)

Return a list of system roles. / Get a specific system role by roleId.

Parameters
role_id – The System Role ID.

fetch_system_task(*response*)

Get the background task by the given task Id.

Parameters
task_id –

fetch_task(*response*)

Check the status of a queued task associated with a Course.

Returns 200 unless task is complete.

Parameters

- **course_id** – The course or organization ID.
- **task_id** –

fetch_terms(*response*)

Return a list of terms. / Load a term.

Parameters
term_id – The term ID.

fetch_user_grades(*response*)

Load the course grades for a specific user.

Parameters

- **course_id** – The course or organization ID.
- **user_id** – The user ID.

fetch_user_memberships(*response*) → list[*BBMembership*]

Return a list of course and organization memberships for the specified user.

Parameters
user_id – The user ID.

fetch_users(*response*)

Return a list of users. / Load a user.

Properties returned will depend on the caller's entitlements.

Parameters
user_id – The user ID.

fetch_version(*response*)

Get the current Learn server version.

get(*version: int = 1*, *use_api: bool = True*, *json: bool = True*, ***g_kwargs*)

Return a decorator (needed to use fancy notation).

Parameters

- **endpoint** (*string*) – Endpoint to make API call to, including placeholders
- **version** (*int*) – Version of the BB API used at endpoint (used as part of url)
- **json** (*bool*) – If false, returns raw requests response, otherwise returns JSON Object
- **g_kwargs** (*dict*) – Any argument in this parameter will be passed on to the requests call

property logger

Logger for Blackboard API, set at level DEBUG.

property timeout: int

Request timeout.

property username: str

Username field used for API requests.

class blackboard_sync.blackboard.api.**SafeFormat**

Custom dictionary object.

Needed to safely format endpoint strings with placeholders without having all parameters (those not present will be left blank)

3.1.2 Blackboard Data Classes

Blackboard Model Classes

class blackboard_sync.blackboard.blackboard.**BBAttachment**(***, *id: str | None = None*, *fileName: str | None = None*, *mimeType: str | None = None*)

Blackboard File Attachment.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

fileName: str | None

id: str | None

mimeType: str | None

model_config: ClassVar[ConfigDict] = {'frozen': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'fileName': FieldInfo(annotation=Union[str, NoneType], required=False), 'id': FieldInfo(annotation=Union[str, NoneType], required=False), 'mimeType': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class blackboard_sync.blackboard.blackboard.**BBAvailability**(*, *available: bool | None = None*, *allowGuests: bool = False*, *adaptiveRelease: dict = {}*, *duration:* BBDuration | *None = None*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

adaptiveRelease: dict

allowGuests: bool

available: bool | None

duration: *BBDuration* | None

model_config: ClassVar[ConfigDict] = {'frozen': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adaptiveRelease': FieldInfo(annotation=dict, required=False, default={}), 'allowGuests': FieldInfo(annotation=bool, required=False, default=False), 'available': FieldInfo(annotation=Union[bool, NoneType], required=False), 'duration': FieldInfo(annotation=Union[BBDuration, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class blackboard_sync.blackboard.blackboard.**BBContentChild**(*, *id: str | None = None*, *title: str | None = None*, *body: str | None = None*, *created: datetime | None = None*, *modified: datetime | None = None*, *position: int = 0*, *hasChildren: bool = False*, *launchInNewWindow: bool = False*, *reviewable: bool = False*, *availability: BBAvailability | None = None*, *contentHandler: BBContentHandler | None = None*, *links: list[BBLink] = []*, *hasGradebookColumns: bool = False*, *hasAssociatedGroups: bool = False*, *parentId: str | None = None*)

Blackboard Content Child.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

body: `str | None`

model_config: `ClassVar[ConfigDict] = {'frozen': True}`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: `ClassVar[dict[str, FieldInfo]] = {'availability': FieldInfo(annotation=Union[BBAvailability, NoneType], required=False), 'body': FieldInfo(annotation=Union[str, NoneType], required=False), 'contentHandler': FieldInfo(annotation=Union[BBContentHandler, NoneType], required=False), 'created': FieldInfo(annotation=Union[datetime, NoneType], required=False), 'hasAssociatedGroups': FieldInfo(annotation=bool, required=False, default=False), 'hasChildren': FieldInfo(annotation=bool, required=False, default=False), 'hasGradebookColumns': FieldInfo(annotation=bool, required=False, default=False), 'id': FieldInfo(annotation=Union[str, NoneType], required=False), 'launchInNewWindow': FieldInfo(annotation=bool, required=False, default=False), 'links': FieldInfo(annotation=list[BBLink], required=False, default=[]), 'modified': FieldInfo(annotation=Union[datetime, NoneType], required=False), 'parentId': FieldInfo(annotation=Union[str, NoneType], required=False), 'position': FieldInfo(annotation=int, required=False, default=0), 'reviewable': FieldInfo(annotation=bool, required=False, default=False), 'title': FieldInfo(annotation=Union[str, NoneType], required=False)}`

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

parentId: `str | None`

class blackboard_sync.blackboard.blackboard.**BBContentHandler**(*, *id:* BBResourceType | *str* | *None* = *None*, *url:* *str* | *None* = *None*, *file:* BBFile | *None* = *None*, *gradeColumnId:* *str* | *None* = *None*, *groupContent:* *bool* | *None* = *None*, *targetId:* *str* | *None* = *None*, *targetType:* *str* | *None* = *None*, *placementHandle:* *str* | *None* = *None*, *assessmentId:* *str* | *None* = *None*, *proctoring:* BBProctoring | *None* = *None*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

assessmentId: `str | None`

file: *BBFile* `| None`

gradeColumnId: `str | None`

groupContent: `bool | None`

id: *BBResourceType* `| str | None`

property is_not_handled: `bool`

Return true if resource should not be handled.

model_config: `ClassVar[ConfigDict] = {'frozen': True}`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: `ClassVar[dict[str, FieldInfo]] = {'assessmentId': FieldInfo(annotation=Union[str, NoneType], required=False), 'file': FieldInfo(annotation=Union[BBFile, NoneType], required=False), 'gradeColumnId': FieldInfo(annotation=Union[str, NoneType], required=False), 'groupContent': FieldInfo(annotation=Union[bool, NoneType], required=False), 'id': FieldInfo(annotation=Union[BBResourceType, str, NoneType], required=False), 'placementHandle': FieldInfo(annotation=Union[str, NoneType], required=False), 'proctoring': FieldInfo(annotation=Union[BBProctoring, NoneType], required=False), 'targetId': FieldInfo(annotation=Union[str, NoneType], required=False), 'targetType': FieldInfo(annotation=Union[str, NoneType], required=False), 'url': FieldInfo(annotation=Union[str, NoneType], required=False)}`

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

placementHandle: `str | None`

proctoring: *BBProctoring* `| None`

classmethod resource_parser(*v:* BBResourceType | *str*)

Validate and parse an id resource type.

targetId: str | None

targetType: str | None

url: str | None

class blackboard_sync.blackboard.blackboard.**BBCourse**(*, *id: str | None = None*, *courseId: str | None = None*, *name: str | None = None*, *description: str | None = None*, *modified: datetime | None = None*, *organization: bool = False*, *ultraStatus: str | None = None*, *closedComplete: bool = False*, *availability:* BBAvailability *| None = None*, *enrollment:* BBEnrollment *| None = None*, *locale:* BBLocale *| None = None*, *externalAccessUrl: str | None = None*)

BlackboardCourse. Represents an academic course.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

availability: *BBAvailability* | None

closedComplete: bool

property code: str | None

Parse course code.

courseId: str | None

description: str | None

enrollment: *BBEnrollment* | None

externalAccessUrl: str | None

id: str | None

locale: *BBLocale* | None

model_config: ClassVar[ConfigDict] = {'frozen': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'availability': FieldInfo(annotation=Union[BBAvailability, NoneType], required=False), 'closedComplete': FieldInfo(annotation=bool, required=False, default=False), 'courseId': FieldInfo(annotation=Union[str, NoneType], required=False), 'description': FieldInfo(annotation=Union[str, NoneType], required=False), 'enrollment': FieldInfo(annotation=Union[BBEnrollment, NoneType], required=False), 'externalAccessUrl': FieldInfo(annotation=Union[str, NoneType], required=False), 'id': FieldInfo(annotation=Union[str, NoneType], required=False), 'locale': FieldInfo(annotation=Union[BBLocale, NoneType], required=False), 'modified': FieldInfo(annotation=Union[datetime, NoneType], required=False), 'name': FieldInfo(annotation=Union[str, NoneType], required=False), 'organization': FieldInfo(annotation=bool, required=False, default=False), 'ultraStatus': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

model_post_init(*__context: Any*) → None

This function is meant to behave like a BaseModel method to initialise private attributes.

It takes context as an argument since that's what pydantic-core passes when calling it.

Args:
self: The BaseModel instance. __context: The context.

modified: datetime | None

name: str | None

organization: bool

property title: str | None

Parse course title.

ultraStatus: str | None

class blackboard_sync.blackboard.blackboard.**BBCourseContent**(*, *id: str | None = None*, *title: str | None = None*, *body: str | None = None*, *created: datetime | None = None*, *modified: datetime | None = None*, *position: int = 0*, *hasChildren: bool = False*, *launchInNewWindow: bool = False*, *reviewable: bool = False*, *availability: BBAvailability | None = None*, *contentHandler: BBContentHandler | None = None*, *links: list[BBLink] = []*, *hasGradebookColumns: bool = False*, *hasAssociatedGroups: bool = False*)

Blackboard Content.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

availability: *BBAvailability* | None

body: str | None

contentHandler: *BBContentHandler* | None

created: datetime | None

hasAssociatedGroups: bool

hasChildren: bool

hasGradebookColumns: bool

id: str | None

launchInNewWindow: bool

links: list[*BBLink*]

model_config: ClassVar[ConfigDict] = {'frozen': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'availability': FieldInfo(annotation=Union[BBAvailability, NoneType], required=False), 'body': FieldInfo(annotation=Union[str, NoneType], required=False), 'contentHandler': FieldInfo(annotation=Union[BBContentHandler, NoneType], required=False), 'created': FieldInfo(annotation=Union[datetime, NoneType], required=False), 'hasAssociatedGroups': FieldInfo(annotation=bool, required=False, default=False), 'hasChildren': FieldInfo(annotation=bool, required=False, default=False), 'hasGradebookColumns': FieldInfo(annotation=bool, required=False, default=False), 'id': FieldInfo(annotation=Union[str, NoneType], required=False), 'launchInNewWindow': FieldInfo(annotation=bool, required=False, default=False), 'links': FieldInfo(annotation=list[BBLink], required=False, default=[]), 'modified': FieldInfo(annotation=Union[datetime, NoneType], required=False), 'position': FieldInfo(annotation=int, required=False, default=0), 'reviewable': FieldInfo(annotation=bool, required=False, default=False), 'title': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

modified: datetime | None

position: int

reviewable: bool

title: str | None

property title_path_safe: str

Return a path safe version of the title.

class blackboard_sync.blackboard.blackboard.**BBDuration**(*, *type: str | None = None*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

model_config: ClassVar[ConfigDict] = {'frozen': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'type': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

type: str | None

class blackboard_sync.blackboard.blackboard.**BBDurationType**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Blackboard Course Duration Type.

Continuous: str = 'Continuous'

DateRange: str = 'DateRange'

FixedNumDays: str = 'FixedNumDays'

Term: str = 'Term'

class blackboard_sync.blackboard.blackboard.**BBEnrollment**(***, *type: str | None = None*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

model_config: ClassVar[ConfigDict] = {'frozen': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'type': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

type: str | None

class blackboard_sync.blackboard.blackboard.**BBFile**(***, *fileName: str | None = None*)

Blackboard File.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

fileName: str | None

model_config: ClassVar[ConfigDict] = {'frozen': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'fileName': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class blackboard_sync.blackboard.blackboard.**BBLink**(*, *href: str | None = None*, *rel: str | None = None*, *title: str | None = None*, *type: str | None = None*)

Blackboard Link.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

href: str | None

model_config: ClassVar[ConfigDict] = {'frozen': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'href': FieldInfo(annotation=Union[str, NoneType], required=False), 'rel': FieldInfo(annotation=Union[str, NoneType], required=False), 'title': FieldInfo(annotation=Union[str, NoneType], required=False), 'type': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

rel: str | None

title: str | None

type: str | None

class blackboard_sync.blackboard.blackboard.**BBLocale**(*, *force: bool = False*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

force: bool

model_config: ClassVar[ConfigDict] = {'frozen': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'force': FieldInfo(annotation=bool, required=False, default=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class blackboard_sync.blackboard.blackboard.**BBMembership**(*, *id: str | None = None*, *userId: str | None = None*, *courseId: str | None = None*, *dataSourceId: str | None = None*, *created: datetime | None = None*, *modified: datetime | None = None*, *availability: BBAvailability | None = None*, *courseRoleId: str | None = None*, *lastAccessed: datetime | None = None*, *childCourseId: str | None = None*)

Blackboard Membership. Represents relation between student and course.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

availability: *BBAvailability* | None

childCourseId: str | None

courseId: str | None

courseRoleId: str | None

created: datetime | None

dataSourceId: str | None

id: str | None

lastAccessed: datetime | None

model_config: ClassVar[ConfigDict] = {'frozen': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'availability': FieldInfo(annotation=Union[BBAvailability, NoneType], required=False), 'childCourseId': FieldInfo(annotation=Union[str, NoneType], required=False), 'courseId': FieldInfo(annotation=Union[str, NoneType], required=False), 'courseRoleId': FieldInfo(annotation=Union[str, NoneType], required=False), 'created': FieldInfo(annotation=Union[datetime, NoneType], required=False), 'dataSourceId': FieldInfo(annotation=Union[str, NoneType], required=False), 'id': FieldInfo(annotation=Union[str, NoneType], required=False), 'lastAccessed': FieldInfo(annotation=Union[datetime, NoneType], required=False), 'modified': FieldInfo(annotation=Union[datetime, NoneType], required=False), 'userId': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

modified: datetime | None

userId: str | None

class blackboard_sync.blackboard.blackboard.**BBProctoring**(*, *secureBrowserRequiredToTake: bool = False*, *secureBrowserRequiredToReview: bool = False*, *webcamRequired: bool = False*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

model_config: ClassVar[ConfigDict] = {'frozen': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'secureBrowserRequiredToReview': FieldInfo(annotation=bool, required=False, default=False), 'secureBrowserRequiredToTake': FieldInfo(annotation=bool, required=False, default=False), 'webcamRequired': FieldInfo(annotation=bool, required=False, default=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

secureBrowserRequiredToReview: bool

secureBrowserRequiredToTake: bool

webcamRequired: bool

class blackboard_sync.blackboard.blackboard.**BBResourceType**(*value*, *names=None*, *, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Different resource types on Blackboard.

asmt_test_link = 'x-bb-asmt-test-link'

assignment = 'x-bb-assignment'

blankpage = 'x-bb-blankpage'

bltiplacement_portal = 'x-bb-bltiplacement-Portal'

courselink = 'x-bb-courselink'

document = 'x-bb-document'

externallink = 'x-bb-externallink'

file = 'x-bb-file'

folder = 'x-bb-folder'

syllabus = 'x-bb-syllabus'

toollink = 'x-bb-toollink'

turnitin_assignment = 'x-turnitin-assignment'

class blackboard_sync.blackboard.blackboard.**ImmutableModel**

Model with const attributes.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

model_config: ClassVar[ConfigDict] = {'frozen': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

3.2 Sync API Reference

3.2.1 BlackboardSync

BlackboardSync.

Automatically sync content from Blackboard

class blackboard_sync.sync.**BlackboardSync**

Represents an instance of the BlackboardSync application.

Create an instance of the program.

_sync_task() → None

Constantly check if the data is outdated and if so start a download job.

Method run by Sync thread.

auth(*cookie_jar: RequestsCookieJar*) → bool

Create a new Blackboard session with the given credentials.

Will start syncing automatically if login successful.

Parameters

persistence (*bool*) – If true, login will be saved in the OS designated keyring.

change_download_location(*new_dir: Path*, *redownload: bool = False*) → None

Set new sync location.

Parameters

- **dir** (*Path*) – The path of the sync dir.
- **redownload** (*bool*) – If true, ALL content will be re-downloaded to the new location.

property download_location: Path

Location to where all Blackboard content will be downloaded.

force_sync() → None

Force Sync thread to start download job ASAP.

property is_active: bool

Indicate the state of the sync thread.

property is_logged_in: bool

Indicate if a user session is currently active.

property is_syncing: bool

Flag raised everytime a download job is running.

property last_sync_time: datetime | None

Datetime right before last download job started.

log_out() → None

Stop syncing and forget user session.

property logger: Logger

Logger for BlackboardSync, set at level WARN.

property min_year: int

property next_sync: datetime

Time when last sync will be outdated.

property outdated: bool

Return true if last download job is outdated.

setup(*university_index: int*, *download_location: Path*, *min_year: int | None = None*) → None

Setup the university information.

start_sync() → None

Stars Sync thread.

stop_sync() → None

Stop Sync thread.

property sync_interval: int

Time to wait between download jobs.

property university_index

property username: str | None

3.2.2 BlackboardDownload

BlackboardDownload, mass download all user content from Blackboard

class blackboard_sync.download.**BlackboardDownload**(*sess: BlackboardSession*, *download_location: Path*, *last_downloaded: datetime | None = None*, *data_sources: list[str] = []*, *min_year: int | None = None*)

Blackboard download job.

BlackboardDownload constructor

Download all files in blackboard recursively to download_location, only if they have been altered since specified datetime

Keyword arguments:

Parameters

- **sess** (BlackboardSession) – UCLan BB user session
- **download_location** ((*str / Path*)) – Where files will be stored
- **last_downloaded** (*str*) – Files modified before this will not be downloaded
- **data_sources** – List of valid data sources
- **min_year** – Only courses created on or after this year will be downloaded

_create_desktop_link(*path: Path*, *url: str*, *comment: str = ''*) → None

Creates a platform-aware internet shortcut

_handle_file(*content: BBCourseContent*, *parent_path: Path*, *course_id: str*, *depth: int = 0*) → None

Download BBContent recursively, depending on filetype

cancel() → None

Cancel the download job.

property data_sources: list[str]

Filter for courses.

download() → datetime | None

Retrieve the user's courses, and start download of all contents

Returns
Datetime when method was called.

property download_location: Path

The location where files will be downloaded to.

property files_processed: int

Number of files that have been downloaded.

property logger: Logger

Logger for BlackboardDownload, set at level DEBUG.

property user_id: str

User ID used for API calls.

3.3 PyQt UI Reference

3.3.1 Qt UI Elements

BlackboardSync Qt GUI.

class blackboard_sync.qt.qt_elements.**Assets**

Helper class to get the path of app assets.

classmethod icon() → QIcon

QIcon of application logo.

classmethod load_ui(*qt_obj*)

Load a UI file for a *QObject*.

classmethod watermark() → QPixmap

QPixmap of application watermark.

class blackboard_sync.qt.qt_elements.**LoginWebView**(*start_url: str*, *target_url: str*)

Blackboard login widget.

Create instance of LoginWebView.

clear_cookie_store() → None

Clear the HTTP cache and cookies.

property cookie_jar: RequestsCookieJar

Contains session cookies of the current session.

property login_complete_signal

Fire when the login flow has completed.

restore() → None

property url: str

URL of current website.

class blackboard_sync.qt.qt_elements.**LoginWindow**

Deprecated widget previously used to login.

class blackboard_sync.qt.qt_elements.**OSUtils**

static add_to_startup() → None

Add the app to start up on macOS.

static open_dir_in_file_browser(*dir_to_open: Path*) → None

Start a subprocess to open the default file explorer at the given location.

class blackboard_sync.qt.qt_elements.**PersistenceWarning**

QDialog shown if user chooses to store their login details on their device.

Create instance of PersistenceWarning Dialog.

class blackboard_sync.qt.qt_elements.**RedownloadDialog**

QMessageBox shown after a change in download location.

It consults the user about whether files should be redownloaded to the new location or not.

Create a *RedownloadDialog*.

property redownload: bool

Indicate if files have to be redownloaded.

class blackboard_sync.qt.qt_elements.**SettingsWindow**

Settings windown UI element.

Create instance of SettingsWindow.

property download_location: Path

Path of download location.

property log_out_signal

Fire when user chooses to log out.

property save_signal

Fire when settings are saved.

property setup_wiz_signal

Fire when user wants to redo initial setup.

property sync_frequency: int

Seconds to wait between each sync job.

property username: str

Username of current session.

class blackboard_sync.qt.qt_elements.**SetupWizard**(*institutions: list[str]*)

Initial setup wizard.

Create a *SetupWizard*.

Parameters
institutions (*list[str]*) – List of institution names

class Pages(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Pages contained in the wizard.

DOWNLOAD_LOCATION = 2

DOWNLOAD_SINCE = 3

INSTITUTION = 1

INTRO = 0

LAST = 3

property download_location: Path

Sync location path selected by user.

initializePage(*self*, *id: int*)

property institution: str

Text of item selected in institution combo box.

property institution_index: int

Index of item selected in institution combo box.

property min_year: int | None

Courses from this year onward will be downloaded.

validateCurrentPage() → bool

Override QWizard method to validate pages.

class blackboard_sync.qt.qt_elements.**SyncPeriod**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Enum containing all valid Sync intervals for this UI.

HALF_HOUR = 1800

ONE_HOUR = 3600

SIX_HOURS = 21600

class blackboard_sync.qt.qt_elements.**SyncTrayIcon**

BlackboardSync system tray icon.

Create a *QSystemTrayIcon*.

property login_signal

Fire once user is authenticated.

property open_dir_signal

Fire once user wants to open download directory.

property quit_signal

Fire once user decides to quit app.

property reset_setup_signal

Fire when the user wants to reset the initial setup.

set_logged_in(*logged: bool*) → None

Set logged-in status in menu.

property settings_signal

Fire when the settings menu is opened.

property show_menu_signal

Fire when menu is about to be shown.

show_msg(*title: str*, *msg: str*, *severity: int = 1*, *duration: int = 10*) → None

Show the user a message through the tray icon.

property sync_signal

Fire if user forces sync.

toggle_currently_syncing(*syncing: bool*) → None

Toggle currently syncing indicator in menu.

update_last_synced(*last: str*) → None

Update last sync time in menu.

class blackboard_sync.qt.qt_elements.**SyncTrayMenu**(*logged_in: bool = False*, *last_synced: str = ''*)

QMenu associated with app system tray icon.

Create the menu for a *SyncTrayIcon*.

Parameters

- **logged_in** (*bool*) – Whether user is currently logged in.
- **last_synced** (*str*) – Last sync time shown.

set_logged_in(*logged: bool*) → None

Set the UI to reflect logged-in status.

toggle_currently_syncing(*syncing: bool*) → None

Toggle the currently syncing indicator.

update_last_synced(*last: str*) → None

Update the time of last download shown to user.

class blackboard_sync.qt.qt_elements.**UniNotSupportedDialog**(*help_url: str*)

QDialog about unsupported Blackboard partners.

Create instance of dialog.

Parameters
help_url (*str*) – URL to help website

class blackboard_sync.qt.qt_elements.**UpdateFoundDialog**

QMessageBox shown after a more recent version was found.

Create a *UpdateFoundDialog*.

property should_update: bool

Indicate if BBSync should be updated.

PYTHON MODULE INDEX

b

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W