nbexchange Documentation

Release stable

Jun 24, 2022

Documentation

1 API Specification for the NBExchange service 3

2 Exchange API 5

Configuration documentation is at https://github.com/edina/nbexchange/

CHAPTER 1

API Specification for the NBExchange service

All URLs relative to */services/nbexchange*

1.1 Assignments

…/assignments?course_id=$course_code

GET: returns list of assignments

Returns

{"success": True,

"value": [{ "assignment_id": "$assignment_code", "course_id": "$course_code", "student_id": Int "status": Str, "path": path, "notebooks": [

{ "notebook_id": x.name, "has_exchange_feedback": False, "feedback_updated": False, "feedback_timestamp": None, } for x in assignment.notebooks],

"timestamp": action.timestamp.strftime("%Y-%m-%d %H:%M:%S.%f %Z"

),

}, {},..]}

or

{"success": False, "note": $note}

1.2 Assignment

…/assignment?course_id=$course_code&assignment_id=$assignment_code

GET: downloads assignment

Returns binary data or raises Exception (which is returned as a *503* error)

POST: (role=instructor, with file): Add ("release") an assignment returns

> {"success": True, "note": "Released"}

or raises Exception (which is returned as a *503* error)

DELETE: (role=instructor, with file): Remove an assignment.

Marks an asiignment as `active: False`, and forgets any associated notebooks. Returns

> {"success": True, "note": "Assignment '$assignment_code' on course '$course_code' marked as unreleased by user $user"

Takes as *optional* parameter `purge`. This will delete the notebooks, the assignment, and any associated data (`actions`, `feedback`, etc). Returns

> {"success": True, "note": "Assignment '$assignment_code' on course '$course_code' deleted and purged from the database by user $user"}

If there are permission issues, returns

> {"success": False, "note": $note}

1.3 Submission

> .../submission?course_id=$course_code&assignment_id=$assignment_code

POST: stores the submission for that user returns

> {"success": True, "note": "Released"}

or raises Exception (which is returned as a *503* error)

1.4 Collections

> .../collections?course_id=$course_code&assignment_id=$assignment_code

GET: gets a list of submitted items Return: same as *Assignments <#assignments>*

1.5 Collection

> .../collections?course_id=$course_code&assignment_id=$assignment_code&path=$url_encoded_path

GET: downloads submitted assignment Return: same as *Assignment <#assignment>*

CHAPTER 2

Exchange API

This is an overview of how *nbgrader* interacts with it's exchange service... *HOW* that exchenage service works internally is a seperate question

2.1 A simplistic overview

`Assignments` are `created`, `generated`, `released`, `fetched`, `submitted`, `collected`, `graded`. Then `feedback` can be `generated`, `released`, and `fetched`.

The exchange is responsible for recieving *released* assignments, allowing those assignments to be *fetched*, accepting *submissions*, and allowing those submissions to be *collected*. It also allows *feedback* to be transferred.

In doing this, the exchange is the authoritative place to get a list of what's what.

2.1.1 Defined directories

`CourseDirectory` defines the following directories (and their defaults):

- `source_directory` - Where new assignments that are created by instructors are put (`source`)
- `release_directory` - Where assignments that have been processed for release are copied to (`release`)
- `submitted_directory` - Where student submissions are copied to, when an instructor *collects* (`submitted`)
- `autograded_directory` - Where student submissions are copied to, having been *autograded* (`autograded`)
- `feedback_directory` - Where feedback is copied to, when Instructors *generate feedback* (`feedback`)

Also, taken from the nbgrader help:

```
The nbgrader application is a system for assigning and grading notebooks.
Each subcommand of this program corresponds to a different step in the
grading process. In order to facilitate the grading pipeline, nbgrader
places some constraints on how the assignments must be structured. By
default, the directory structure for the assignments must look like this:

    {nbgrader_step}/{student_id}/{assignment_id}/{notebook_id}.ipynb

where 'nbgrader_step' is the step in the nbgrader pipeline, 'student_id'
is the ID of the student, 'assignment_id' is the name of the assignment,
and 'notebook_id' is the name of the notebook (excluding the extension).
```

2.1.2 Calling exchange classes

Exchange functions are called three ways:

1. From the command line - eg: `nbgrader release_assignment assignment1`.
2. From **formgrader** server_extension, which generally calls the methods defined in `nbgrader/apps/{foo}app.py`.
3. From the **assignment_list** server_extension, which generally calls the methods directly.

2.2 The classes

The nbgrader exchange uses the followng classes:

```
Exchange
ExchangeError
ExchangeCollect
ExchangeFetch
ExchangeFetchAssignment
ExchangeFetchFeedback
ExchangeList
ExchangeRelease
ExchangeReleaseAssignment
ExchangeReleaseFeedback
ExchangeSubmit
```

2.2.1 `Exchange`

Base class. Contains some required configuration parameters and elements - the prominant ones include `path_includes_course` and `coursedir`.

This class defines the following methods which are expeceted to be subclassed:

`init_src()` Define the location files are copied *from*

`init_dest()` Define the location files are copied *to*

`copy_files()` Actually copy the files.

The class also defines a convenience method, which may be subclassed:

```
def start(self):
    if sys.platform == 'win32':
        self.fail(``Sorry, the exchange is not available on Windows.``)
    if not self.coursedir.groupshared:
        # This just makes sure that directory is o+rwx.  In group shared
        # case, it is up to admins to ensure that instructors can write
        # there.
        self.ensure_root()
    self.set_timestamp()
    self.init_src()
    self.init_dest()
    self.copy_files()
```

You may want to subclass this, as `self.root` as a directory only makes sense in a file-based exchange.

2.2.2 ExchangeError

Does nothing in the default exchange, but available for use

2.2.3 ExchangeCollect

Fetches [all] submissions for a specified assignment from the exchange and puts them in the [instructors] home space.

The exchange is called thus:

```
self.coursedir.assignment_id = assignment_id
collect = ExchangeCollect(
    coursedir=self.coursedir,
    authenticator=self.authenticator,
    parent=self)
try:
    collect.start()
except ExchangeError:
    self.fail("nbgrader collect failed")
```

returns.... *nothing*

Expected behaviours

- The expected *destination* for collected files is `{self.coursedir.submitted_directory}/{student_id}/{self.coursedir.assignment_id}`
- `collect.update` is a flag to indicate whether collected files sould be replaced is a later submission is available. There is an assumption this defaults to `True`

2.2.4 ExchangeFetch

(Depreciated, use `ExchangeFetchAssignment`)

2.2.5 ExchangeFetchAssignment

Gets the named assignment & puts the files in the users home space.

The nbgrader server_extension calls it thus:

```
with self.get_assignment_dir_config() as config:
    try:
        config = self.load_config()
        config.CourseDirectory.course_id = course_id
        config.CourseDirectory.assignment_id = assignment_id

        coursedir = CourseDirectory(config=config)
        authenticator = Authenticator(config=config)
        fetch = ExchangeFetchAssignment(
            coursedir=coursedir,
            authenticator=authenticator,
            config=config)
        fetch.start()
    .....
```

Returns.... *nothing*

Expected behaviours

The expected *destination* for files is `{self.assignment_dir}/{self.coursedir.assignment_id}` however if `self.path_includes_course` is `True`, then the location should be `{self.assignment_dir}/{self.coursedir.course_id}/{self.coursedir.assignment_id}`

`self.coursedir.ignore` is described as a:

```
List of file names or file globs.
Upon copying directories recursively, matching files and
directories will be ignored with a debug message.
```

This should be honoured.

In the default exchange, existing files are *not* replaced.

2.2.6 ExchangeFetchFeedback

This copies feedback from the exchange into the students home space.

The nbgrader server_extension calls it thus:

```
with self.get_assignment_dir_config() as config:
    try:
        config = self.load_config()
        config.CourseDirectory.course_id = course_id
        config.CourseDirectory.assignment_id = assignment_id

        coursedir = CourseDirectory(config=config)
        authenticator = Authenticator(config=config)
        fetch = ExchangeFetchFeedback(
            coursedir=coursedir,
            authenticator=authenticator,
```

(continues on next page)

(continued from previous page)

```
            config=config)
        fetch.start()
    .....
```

returns.... *nothing*

Expected behaviours

- Files should be copied into a `feedback` directory in whichever directory `ExchangeFetchAssignment` deposited files.
- Each submission should be copied into a `feedback/{timestamp}` directory, where `timestamp` is the timestamp from the `timestamp.txt` file generated during the submission.

When writing your own Exchange

- You to need to consider stopping students from seeing each others submissions

2.2.7 `ExchangeList`

This class is responsible for determining what assignments are available to the user.

It has three flags to define various modes of operation:

`self.remove=True` If this flag is set, the assignment files (as defined below) are removed from the exchange.

`self.inbound=True` or `self.cached=True` These both refer to *submitted* assignments. The `assignment_list` plugin sets `config.ExchangeList.cached = True` when it queries for submitted notebooks.

neither This is *released* (and thus *fetched*) assignments.

Note that `CourseDirectory` and `Authenticator` are defined when the server_sextension *assignment_list* calls the lister:

```
with self.get_assignment_dir_config() as config:
    try:
        if course_id:
            config.CourseDirectory.course_id = course_id

        coursedir = CourseDirectory(config=config)
        authenticator = Authenticator(config=config)
        lister = ExchangeList(
            coursedir=coursedir,
            authenticator=authenticator,
            config=config)
        assignments = lister.start()
    ....
```

returns a List of Dicts - eg:

```
[
    {'course_id': 'course_2', 'assignment_id': 'car c2', 'status': 'released', .....},
```

(continues on next page)

(continued from previous page)

```
    {'course_id': 'course_2', 'assignment_id': 'tree c2', 'status': 'released', .....}
↪,
]
```

The format and structure of this data is discussed in *ExchangeList Date Return structure* below.

Note

This gets called **TWICE** by the `assignment_list` server_extension - once for *released* assignments, and again for *submitted* assignments.

2.2.8 ExchangeRelease

(Depreciated, use `ExchangeReleaseAssignment`)

2.2.9 ExchangeReleaseAssignment

This should copy the assignment from the *release* location (normally `{self.coursedir.release_directory}/{self.coursedir.assignment_id}`) and copies it into the exchange service.

The class should check for the assignment existing (look in `{self.coursedir.release_directory}/{self.coursedir.assignment_id}`) before actually copying

The exchange is called thus:

```
release = ExchangeReleaseAssignment(
    coursedir=self.coursedir,
    authenticator=self.authenticator,
    parent=self)
try:
    release.start()
except ExchangeError:
    self.fail(``nbgrader release_assignment failed``)
```

returns.... *nothing*

2.2.10 ExchangeReleaseFeedback

This should copy all the feedback for the current assignment to the exchange.

Feedback is generated by the Instructor. From `GenerateFeedbackApp`:

```
Create HTML feedback for students after all the grading is finished.
This takes a single parameter, which is the assignment ID, and then (by
default) looks at the following directory structure:

    autograded/*/{assignment_id}/*.ipynb

from which it generates feedback the the corresponding directories
according to:

    feedback/{student_id}/{assignment_id}/{notebook_id}.html
```

The exchange is called thus:

```
release_feedback = ExchangeReleaseFeedback(
    coursedir=self.coursedir,
    authenticator=self.authenticator,
    parent=self)
try:
    release_feedback.start()
except ExchangeError:
    self.fail("nbgrader release_feedback failed")
```

returns. . . .. nothing

2.2.11 ExchangeSubmit

This should copy the assignment from the user's work space, and make it available for instructors to *collect*.

The exchange is called thus:

```
with self.get_assignment_dir_config() as config:
    try:
        config = self.load_config()
        config.CourseDirectory.course_id = course_id
        config.CourseDirectory.assignment_id = assignment_id
        coursedir = CourseDirectory(config=config)
        authenticator = Authenticator(config=config)
        submit = ExchangeSubmit(
            coursedir=coursedir,
            authenticator=authenticator,
            config=config)
        submit.start()
    .....
```

The *source* for files to be submitted needs to match that in `ExchangeFetchAssignment`.

returns. . . . *nothing*

When writing your own Exchange

- You to need to consider stopping students from seeing each others submissions
- nbgrader functionality requires a file called `timestamp.txt` to be in the submission, containing the timestamp of that submission. The creation of this file is the responsibility of this class.
- Whilst nothing is done *as yet*, the default exchange checks the names of submitted notebooks, and logs differences.
- Submissions need to record `student_id`, as well as `course_id` & `assignment_id`
- The default exchange copies files to both an `inbound` and `cache` store. This may be significant considering `ExchangeList`

2.3 ExchangeList Date Return structure

As mentioned in the *ExchangeList* class documentation above, this data is returned as a List of Dicts.

The format of the Dicts vary depending on the type of assignments being listed.

2.3.1 Removed

Returns a list of assignments formatted as below (whether they are `released` or `submitted`), but with the status set to `removed`

2.3.2 Released & Submitted

1. The first step is to loop through a list of assignments (lets call each one a `path`) and get some basic data:

`released`

```
{course_id: xxxx, assignment_id: yyyy, timestamp: ISO 8601}
```

`submitted`

```
{course_id: xxxx, assignment_id: yyyy, student_id: aaaa, timestamp: ISO 8601}
```

2. We then add `status` and `path` information:

```
if self.inbound or self.cached:
    info['status'] = 'submitted'
    info['path'] = path  # ie, where it is in the exchange
elif os.path.exists(assignment_dir):
    info['status'] = 'fetched'
    info['path'] = os.path.abspath(assignment_dir)  # ie, where it in on the students␣
↪home space.
else:
    info['status'] = 'released'
    info['path'] = path # again, where it is in the exchange

if self.remove:
    info['status'] = 'removed'
    # Note, no path - it's been deleted.
```

(`assignment_dir` is the directory in the students home space, so needs to take into account `self.path_includes_course`)

Note that the API does include `collected`, `feedback_released` and `feedback_fetched` items, however NBgrader doesn't use them, so they are essentially un-processed records - `feedback_released` is per-notebook (with a path) and `collected` & `feedback_fetched` is all notebooks (and sans path)

3. Next loop through all the *notebooks* in the `path`, and get some basic data:

```
nb_info = {'notebook_id': /name, less extension/, 'path': /path_to_file/}
```

4. **If the notebook is `info['status'] != 'submitted'`:** that's all the data we have:

```
info['notebooks'].append(nb_info)
```

else, add *feedback* details for *this* notebook:

```
nb_info['has_local_feedback'] = _has_local_feedback()
nb_info['has_exchange_feedback'] = _has_exchange_feedback()
if nb_info['has_local_feedback']:
    nb_info['local_feedback_path'] = _local_feedback_path()
if nb_info['has_local_feedback'] and nb_info['has_exchange_feedback']:
```

(continues on next page)

(continued from previous page)

```
    nb_info['feedback_updated'] = _exchange_feedback_checksum() != _local_
↪feedback_checksum()
info['notebooks'].append(nb_info)
```

5. **Having looped through all notebooks** If `info['status'] == 'submitted'`, add feedback notes to the top-level assignment record:

```
info['has_local_feedback'] = _any_local_feedback()
info['has_exchange_feedback'] = _any_exchange_feedback()
info['feedback_updated'] = _any_feedback_updated()
if info['has_local_feedback']:
    info['local_feedback_path'] = os.path.join(
        assignment_dir, 'feedback', info['timestamp'])
else:
    info['local_feedback_path'] = None
```