## CONTENTS

1 Architecture/Infrastructure  .......................................................... 3  
   1.1 Development Environments  ...................................................... 3  
   1.2 QA/Production Deployments .................................................. 3  

2 Development Environment Setup  .................................................... 5  
   2.1 Prerequisites  ............................................................................ 5  
   2.2 Local Development Environment Setup .................................... 5  
   2.3 Cron Container ................................................................. 6  

3 Authentication/Authorization  ........................................................... 7  
   3.1 Target Token Instructions ...................................................... 7  

4 User Notes  ..................................................................................... 21  
   4.1 Transfer Details ................................................................. 21  

5 Developer Notes  ............................................................................ 23  
   5.1 Testing  ................................................................................... 23  
   5.2 Docker Commands ............................................................ 23  
   5.3 Updating Documentation ..................................................... 24  
   5.4 GitHub Differences ............................................................ 24  

6 Target Integration  ........................................................................... 25  
   6.1 Target Endpoints ................................................................. 25  
   6.2 Resource Endpoints ............................................................ 27  
   6.3 Resource Download Endpoint ............................................. 31  
   6.4 Resource Upload Endpoint .................................................. 33  
   6.5 Resource Transfer Endpoint ................................................ 36  
   6.6 Keyword Enhancement Endpoint ........................................ 36  
   6.7 Error Handling ................................................................. 38  

7 API Endpoints  ................................................................................. 39  
   7.1 Authentication ................................................................. 39  
   7.2 Duplicate File Handling ....................................................... 39  
   7.3 Searching Resource Collections .......................................... 39  
   7.4 Paging Resource Collections ............................................... 39  
   7.5 Target Endpoints ................................................................. 40  
   7.6 Resource Endpoints ............................................................ 43  
   7.7 Resource Download Endpoints .......................................... 47  
   7.8 Resource Upload Endpoints .................................................. 50  
   7.9 Resource Transfer Endpoints .............................................. 54  
   7.10 Keyword Enhancement Endpoints ....................................... 60
8 Web Services
  8.1 Fixity ................................................................. 65
  8.2 File Transfer Service (FTS) Metadata ........................................ 67
  8.3 Keyword Assignment .................................................. 73
  8.4 Preservation Quality .................................................. 75

9 Services ................................................................. 77
  9.1 EaaSI (Emulation-as-a-Service Infrastructure) Service ................. 77
  9.2 FAIR Evaluator Service ................................................ 77
  9.3 FAIRshake Assessment Service .......................................... 77

10 Service Endpoints ..................................................... 79
  10.1 Service Endpoints .................................................... 79
  10.2 Keyword Enhancement ................................................ 80
  10.3 EaaSI Endpoints ...................................................... 81
  10.4 FAIRshare Endpoints ................................................ 83
  10.5 FAIRshake Endpoints ................................................ 85

11 Resources ............................................................... 89
  11.1 Links ................................................................. 89
  11.2 Example BagIts ...................................................... 89

12 QA Testing ............................................................... 91
  12.1 Resources ............................................................ 92
  12.2 BagIt Tool ............................................................ 92
  12.3 Getting Authorization Tokens From Partner Sites ....................... 92
  12.4 Test Files ............................................................ 92
  12.5 Known Bugs And Issues .............................................. 93
  12.6 Testing Instructions .................................................. 93
  12.7 Verifying Fixity ...................................................... 105
  12.8 Verifying Keyword Enhancement ..................................... 106
  12.9 Services .............................................................. 107

13 Other Integrations ..................................................... 113
  13.1 Whole Tale Integration Proposal .................................... 113
  13.2 Whole Tale Integration Implementation ................................ 114

14 Under Development ................................................... 119

15 Indices ................................................................. 121

HTTP Routing Table ...................................................... 123
PresQT (Preservation Quality Tool) is an open-source tool with RESTful services to improve Preservation and Re-use of Research Data and Software.

Note: Development is underway by the development team in the Center for Research Computing at Notre Dame. This documentation will grow/be changed throughout the life of the PresQT project. Readers should expect that pages will often be incomplete and/or move as features are actively being developed and implemented. Please send any feedback to the content presented here to Noel Recla nrecla@nd.edu

More information can be found here https://presqt.crc.nd.edu/

Current Target Integrations:

<table>
<thead>
<tr>
<th>Target Collection</th>
<th>Details</th>
<th>Download</th>
<th>Upload</th>
<th>Transfer In [Targets]</th>
<th>Transfer Out [Targets]</th>
<th>Hash Algorithms</th>
<th>Keyword Get</th>
<th>Keywords Upload</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSF</td>
<td></td>
<td></td>
<td></td>
<td>[Github, CurateND, Zenodo, GitLab, FigShare]</td>
<td>[Github, Zenodo, GitLab, FigShare]</td>
<td>[sha256, md5]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CurateND</td>
<td></td>
<td></td>
<td></td>
<td>[OSF, CurateND, Zenodo, GitLab, FigShare]</td>
<td>[OSF, Github, Zenodo, GitLab, FigShare]</td>
<td>[md5]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Github</td>
<td></td>
<td></td>
<td></td>
<td>[OSF, CurateND, Zenodo, GitLab, FigShare]</td>
<td>[OSF, Github, Zenodo, GitLab, FigShare]</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zenodo</td>
<td></td>
<td></td>
<td></td>
<td>[OSF, Github, CurateND, GitLab, FigShare]</td>
<td>[OSF, Github, CurateND, GitLab, FigShare]</td>
<td>[md5]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GitLab</td>
<td></td>
<td></td>
<td></td>
<td>[OSF, Github, CurateND, Zenodo, FigShare]</td>
<td>[OSF, Github, Zenodo, FigShare]</td>
<td>[sha256]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FigShare</td>
<td></td>
<td></td>
<td></td>
<td>[OSF, Github, CurateND, Zenodo, Gitlab]</td>
<td>[OSF, Github, Gitlab, Zenodo]</td>
<td>[md5]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Current Service Integrations:

<table>
<thead>
<tr>
<th>Service</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>EaaSI</td>
<td>Send resources from a PresQT server to EaaSI to generate an emulation proposal</td>
</tr>
<tr>
<td>Keyword Enhancement</td>
<td>Suggest/add keywords to existing keywords</td>
</tr>
<tr>
<td>FAIRshare Evaluator</td>
<td>Evaluate the FAIR-ness of a resource</td>
</tr>
<tr>
<td>FAIRshake Assessment</td>
<td>Manually assess the FAIRness of a resource</td>
</tr>
</tbody>
</table>
1.1 Development Environments

PresQT uses Docker throughout its pipeline to make it as easy as possible for newcomers to the project to get up and running with PresQT services. This was accomplished by creating an easy to use development environment using Docker compose.

The diagram below illustrates how we are using Docker Compose to create a constellation of 2 containers on developer machines representing the two essential components of PresQT:

- Nginx
  - Serves as a security layer for incoming requests to PresQT
  - Can also serve as a load balancer in the future
- Django/Gunicorn
  - After passing through the Nginx layer, this container processes API requests from users and then takes the necessary actions to fulfill the user’s requests by communicating with partner services via the their own APIs.

1.2 QA/Production Deployments

Unsurprisingly, there is a significant overlap between the developer setup and the QA/Production deployment architecture. The following is how they will vary:

- One machine (the “Web Server”) will be the host for the Nginx container and one or more identical Django containers that will respond to APIs requests from client researchers in a load balanced manner.
CHAPTER TWO

DEVELOPMENT ENVIRONMENT SETUP

2.1 Prerequisites

- Local installation of Docker for Mac/Windows/Linux
- Knowledge of Git procedures
- Knowledge of setting environment variables
- Knowledge of `docker-compose` utility.

2.2 Local Development Environment Setup

1. Clone the repo to your local machine in the desired folder location:

   $ git clone https://github.com/ndlib/presqt.git

2. Export required `ENV_VARS`:

   - `ENVIRONMENT`: Should be either `production` or `development`.
   - `SECRET_KEY`: A Django “secret key” value.

   # Example Exportation
   $ export ENVIRONMENT=development
   $ export SECRET_KEY=y4xgryt7ex9g+4mcs4=^sg5afp31z#94eb6=6o6161o=a3ly_h

3. Export optional `ENV_VARS` for testing:

   - `CURATE_ND_TEST_TOKEN`: The test token for Curate’s API.
   - `GITHUB_TEST_USER_TOKEN`: The test token for GitHub’s API.
   - `OSF_TEST_USER_TOKEN`: The test token for OSF’s API.
   - `OSF_PRIVATE_USER_TOKEN`: The private test token for OSF’s API.
   - `OSF_UPLOAD_TEST_USER_TOKEN`: The upload test token for OSF’s API.
   - `OSF_PRESQT_FORK_TOKEN`: The PresQT fork user test token for OSF’s API.
   - `ZENODO_TEST_USER_TOKEN`: The test token for Zenodo’s API.
   - `GITLAB_TEST_USER_TOKEN`: The test token for GitLab’s API.
   - `FIGSHARE_TEST_USER_TOKEN`: The test token for FigShare’s API.
4. Execute `docker-compose up` within the repo’s base folder.

```bash
$ docker-compose up --build
```


## 2.3 Cron Container

There is now a third docker container that is responsible for running clean up tasks at specified times. It has been implemented in development to run the `delete_outdated_mediafiles` command every 15 minutes. The command has also been altered slightly to delete any mediafiles held in these directories when you are in a development environment. The command is set to run daily at 4:30am for our other servers.
CHAPTER THREE

AUTHENTICATION/AUTHORIZATION

PresQT will not have the ability to create a ‘session’ for the user based on authentication. It will be expecting tokens to be passed through the header of the request. When retrieving items it expects ‘presqt-source-token’ to be in the header. When depositing an item it expects ‘presqt-destination-token’ to be in the header.

3.1 Target Token Instructions

3.1.1 Open Science Framework

1. Navigate to and login to your account.
2. Upon logging in, click on your username in the top right corner and then click on Settings.
3. Once in Settings, click on Personal Access Tokens in the left hand menu.
4. Click on Create token.
5. Create a token name and select all scope options. Then press Create token.
6. Make sure you copy this token somewhere securely, this will be the only time it is shown to you.

3.1.2 CurateND

1. Navigate to and login to your account.
2. In the top right corner, select Manage and then click on API Access Tokens.
3. Click on Create New Token.
4. Make sure you copy this token somewhere securely.

3.1.3 GitHub

1. Navigate to and login to your account.
2. In the top right corner, select your profile picture and then click on Settings.
3. In the bottom left of your settings, select Developer Settings.
4. On the left hand side of this screen, select Personal Access Tokens.
5. Click on Generate New Token.
6. Add a note about what the token will be used for, and select all scopes. Then select Generate Token.
Sign in with your OSF account to continue

- Sign in through institution
- OR
- Email
- Password

SIGN IN

Stay signed in
Forgot your password?

Create an OSF account  Back to OSF

My Quick Files  My Projects  Search  Support  Donate

My Profile  OSF Support  Settings  Log out
3.1. Target Token Instructions
Chapter 3. Authentication/Authorization

Personal access tokens

Token **Demo** created:

This token will never expire. This token should never be shared with others. If it is accidentally revealed publicly, it should be deactivated immediately.

Sign in

- **Please enter a username**
- **Please enter a password**

Remember me

Sign In

Need help signing in?

**API Access Tokens**

- My Works
- Group Administration
- My Account
- Temporary Access Grants
- Log Out
API Access Token List

Create New Token

API Access Token

<table>
<thead>
<tr>
<th>API Access Token</th>
<th>Issued By</th>
<th>For User</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Token]</td>
<td>[Issuer]</td>
<td>[User]</td>
</tr>
</tbody>
</table>

Sign in to GitHub

Username or email address
Password
Forgot password?

Sign in

New to GitHub? Create an account.
7. Make sure you copy this token somewhere securely, this will be the only time it is shown to you.

### 3.1.4 Zenodo

1. Navigate to and login to your account.
2. In the top right corner, select your username and then click on Applications.
3. In the Personal access tokens section, click on New token.
4. Give the token a name and select all scopes, then click Create.
5. Make sure you copy this token somewhere securely, this will be the only time it is shown to you.

### 3.1.5 GitLab

1. Navigate to and login to your account.
2. In the top right corner, select your username and then click on Settings.
3. In the left hand menu, select Access Tokens.
4. Give the token a name and select all scopes, then click Create personal access token.
5. Make sure you copy this token somewhere securely, this will be the only time it is shown to you.

### 3.1.6 FigShare

1. Navigate to and login to your account.
2. In the top right corner, select your username and then click on Applications.
3.1. Target Token Instructions

Personal settings

Profile
- Account
- Security
- Security log
- Emails
- Notifications
- Billing
- SSH and GPG keys
- Blocked users
- Repositories
- Organizations
- Saved replies
- Applications

Developer settings

GitHub Apps

OAuth Apps

Personal access tokens

Personal access tokens

Generate new token
Revoke all
Note

Demo

What’s this token for?

Select scopes
Scopes define the access for personal tokens. Read more about OAuth scopes.

<table>
<thead>
<tr>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repo</td>
<td>Full control of private repositories</td>
</tr>
<tr>
<td>repo:status</td>
<td>Access commit status</td>
</tr>
<tr>
<td>repo_deployment</td>
<td>Access deployment status</td>
</tr>
<tr>
<td>public_repo</td>
<td>Access public repositories</td>
</tr>
<tr>
<td>repo:invite</td>
<td>Access repository invitations</td>
</tr>
<tr>
<td>write:packages</td>
<td>Upload packages to github package registry</td>
</tr>
<tr>
<td>read:packages</td>
<td>Download packages from github package registry</td>
</tr>
<tr>
<td>delete:packages</td>
<td>Delete packages from github package registry</td>
</tr>
<tr>
<td>admin:org</td>
<td>Full control of orgs and teams, read and write org projects</td>
</tr>
<tr>
<td>write:org</td>
<td>Read and write org and team membership, read and write org projects</td>
</tr>
<tr>
<td>read:org</td>
<td>Read org and team membership, read org projects</td>
</tr>
<tr>
<td>admin:public_key</td>
<td>Full control of user public keys</td>
</tr>
<tr>
<td>write:public_key</td>
<td>Write user public keys</td>
</tr>
<tr>
<td>read:public_key</td>
<td>Read user public keys</td>
</tr>
<tr>
<td>admin:repo_hook</td>
<td>Full control of repository hooks</td>
</tr>
<tr>
<td>write:repo_hook</td>
<td>Write repository hooks</td>
</tr>
<tr>
<td>read:repo_hook</td>
<td>Read repository hooks</td>
</tr>
<tr>
<td>admin:org_hook</td>
<td>Full control of organization hooks</td>
</tr>
<tr>
<td>gist</td>
<td>Create gists</td>
</tr>
<tr>
<td>notifications</td>
<td>Access notifications</td>
</tr>
</tbody>
</table>

Make sure to copy your new personal access token now. You won’t be able to see it again!
Log in to account

- Log in with GitHub
- Log in with ORCID

--- OR ---

Enter your Email Address

Enter your Password

Log in

New to Zenodo? Sign Up

Personal access tokens

Following are personal tokens used to access the Zenodo API:
New personal access token

Name

Name of personal access token.

Scopes

- **deposit:actions**
  Allow publishing of uploads.

- **deposit:write**
  Allow upload (but not publishing).

- **user:email**
  Allow access to email address (read-only).

Scopes assign permissions to your personal access token. A personal access token works just like a normal OAuth access token for authentication against the API.

[Cancel] [Create]

Personal access token / Demo

Access token

Please copy the personal access token now. You won't see it again!

Do not share this personal access token. It gives full access to your account.

Username or email

Password

[Remember me] [Forgot your password?]

Sign in
3.1. Target Token Instructions

Team Prometheus
@Prometheus-Upload

Set status
Profile
Start a Gold trial🚀
Settings
Sign out

User Settings

Profile
Account
Billing
Applications
Chat
Access Tokens
Name
Demo

Expires at
YYYY-MM-DD

Scopes

- **api**
  Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.

- **read_user**
  Grants read-only access to the authenticated user’s profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.

- **read_api**
  Grants read access to the API, including all groups and projects, the container registry, and the package registry.

- **read_repository**
  Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.

- **write_repository**
  Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).

- **read_registry**
  Grants read-only access to container registry images on private projects.

- **write_registry**
  Write Registry

Create personal access token

Your New Personal Access Token

NB

Make sure you save it - you won't be able to access it again.
3. Scroll down to the bottom of the screen, and click **Create Personal Token**.

4. Give the token a description (name), then click **Save**.

5. Make sure you copy this token somewhere securely, this will be the only time it is shown to you.
Create a new personal token

Personal tokens allow you to access our API without going through the 3 legged oauth process. Tokens can be used for many applications, including our desktop uploader. Add a description to easily recognize associations made when you return.

Description *

Cancel Save

Token created

Please copy and save this token as you won't have a chance to see it again later.
## 4.1 Transfer Details

<table>
<thead>
<tr>
<th>Target</th>
<th>As Source</th>
<th>As Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSF</td>
<td>Only provides checksums for OSF Storage files</td>
<td>Writes PRESQT_FTS_METADATA.json file</td>
</tr>
<tr>
<td></td>
<td>Keywords written to ‘Tags’ attribute</td>
<td>Keywords written to ‘Tags’ attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stores all resources in OSF Storage</td>
</tr>
<tr>
<td>curateND</td>
<td>Provides checksums for all files</td>
<td>N/A</td>
</tr>
<tr>
<td>Github</td>
<td>Does not provide checksums for files</td>
<td>Writes PRESQT_FTS_METADATA.json file</td>
</tr>
<tr>
<td></td>
<td>Keywords written to ‘Topics’ attribute</td>
<td>Does not provide checksums for files</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Keywords written to ‘Topics’ attribute</td>
</tr>
<tr>
<td>Zenodo</td>
<td>Provides checksums for all files</td>
<td>Writes PRESQT_FTS_METADATA.json file</td>
</tr>
<tr>
<td></td>
<td>Keywords written to ‘Keywords’ attribute</td>
<td>Resources will be written in BagIt format as a ZIP file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Keywords written to ‘Keywords’ attribute</td>
</tr>
<tr>
<td>GitLab</td>
<td>Provides checksums for all files</td>
<td>Writes PRESQT_FTS_METADATA.json file</td>
</tr>
<tr>
<td></td>
<td>Keywords written to ‘Tag List’ attribute</td>
<td>Keywords written to ‘Tag List’ attribute</td>
</tr>
<tr>
<td>Figshare</td>
<td>Provides checksums for all files</td>
<td>Writes PRESQT_FTS_METADATA.json file</td>
</tr>
<tr>
<td></td>
<td>Keywords written to ‘Tags’ attribute</td>
<td>Resources will be written in BagIt format as a ZIP file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Keywords written to ‘Tags’ attribute.</td>
</tr>
</tbody>
</table>
5.1 Testing

A high code coverage percentage has been maintained with unit and integration tests for all code using a package called Coverage (https://coverage.readthedocs.io/en/v4.5.x/) to track code coverage.

To run unit tests without using Coverage:

```
$ python manage.py test
```

To run unit tests using Coverage with comprehensive code coverage report generated into an HTML file:

```
coverage run manage.py test && coverage combine && coverage html
```

**Note:** This command will generate a directory that is ignored by Git via our .gitignore file. To see the code coverage open the file /coverage_html/index.html in a browser.

**Note:** Coverage options are specified in a configuration file called .coveragerc. This is where you would add files/directories you want to omit from the Coverage report.

**Note:** ‘coverage combine’ will take the coverage files created for multiprocesses (located in the base directory) and will combine them with the main coverage files. If a test using multiprocessing fails these coverage files will remain and must be deleted manually.

We also tried to split unit and integration tests up between core PresQT code and Target code. Tests that cover core code reside in presqt/api_v1/tests/ while target tests that cover target functions reside in presqt/targets/{target_name}/tests/.

**Attention:** All tests require their corresponding target tokens to be stored as environment variables since these tokens can not be stored publicly. Contact an administrator for access to these.

5.2 Docker Commands

To rebuild the docker container after a new package has been added to the requirements files:
Run the following command for an interactive `-i terminal -t` for this container:

```bash
$ docker-compose up --build
```

### 5.3 Updating Documentation

As the project grows we encourage developers to add documentation. PresQT documentation is built using Sphinx and ReadtheDocs.

When documentation is added you should just need to run while in the `/docs` directory:

```bash
$ make clean
$ make html
```


### 5.4 GitHub Differences

There is a slight difference in how we have implemented GitHub as opposed to other partners. Due to the way GitHub’s API handles files, there is no way for us to hit an endpoint with a given id. The only way to navigate to a files endpoint is to know the associated GitHub username and repository title. We have decided to make our own unique id’s for these items by combining the repo_id and the path to the file/dir.

Example of an id:

```
21387123:path2/Fto%2Ffile%2Ejpg
```

We then translate this into two get requests. The first one will be to the repo id.

Example:

```
https://api.github.com/repositories/21387123
```

From this, we can get a contents url that we can translate into a request.

Example:

```
```

Using this custom generated id, we can hit this endpoint on PresQT to get file details.

```
https://presqt-qa.crc.nd.edu/api_v1/targets/github/resources/21387123:path%252Fto%252Ffile%252Ejpg
```
The goal of PresQT is to make it as simple as possible for a new target to integrate itself with the PresQT services. Below are lists of code actions to take when integrating a target.

### 6.1 Target Endpoints

‘Targets’ are providers the PresQT API will connect to such as OSF, CurateND, HubZero, etc. Since PresQT doesn’t have a database, the Targets’ information will be held in a JSON file located in `/presqt/specs/targets.json`. You must add data to this file to integrate with PresQT.

#### 6.1.1 Target Collection/Details

1. Add your target dictionary to the file `presqt/specs/targets.json`

   Target JSON Details:
### Key

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>str</td>
<td>Name of the Target. This will be used as path parameters in the URL.</td>
</tr>
<tr>
<td>readable_name</td>
<td>str</td>
<td>Human readable name of the Target for the front end</td>
</tr>
<tr>
<td>status_url</td>
<td>str</td>
<td>Url which is 200 OK if the API works.</td>
</tr>
<tr>
<td>token_url</td>
<td>str</td>
<td>Url where users can create their API tokens.</td>
</tr>
<tr>
<td>supported_actions</td>
<td>array</td>
<td>Actions the target supports. Only make actions true when action is working</td>
</tr>
<tr>
<td>resource_collection</td>
<td>bool</td>
<td>Get all top level resources for the user in this target</td>
</tr>
<tr>
<td>resource_detail</td>
<td>bool</td>
<td>Get an individual resource’s details</td>
</tr>
<tr>
<td>resource_download</td>
<td>bool</td>
<td>Download a resource</td>
</tr>
<tr>
<td>resource_upload</td>
<td>bool</td>
<td>Upload a resource</td>
</tr>
<tr>
<td>resource_transfer_in</td>
<td>bool</td>
<td>Transfer a resource in to the target</td>
</tr>
<tr>
<td>resource_transfer_out</td>
<td>bool</td>
<td>Transfer a resource out of the target</td>
</tr>
<tr>
<td>supported_transfer_partners</td>
<td>dict</td>
<td>Targets this target can transfer in and out of</td>
</tr>
<tr>
<td>transfer_in</td>
<td>array</td>
<td>Targets this target can accept transfers from</td>
</tr>
<tr>
<td>transfer_out</td>
<td>array</td>
<td>Targets this target can transfer to</td>
</tr>
<tr>
<td>supported_hash_algorithms</td>
<td>array</td>
<td>The hash algorithms supported by the target</td>
</tr>
<tr>
<td>infinite_depth</td>
<td>bool</td>
<td>Does the target support an infinite depth hierarchy?</td>
</tr>
<tr>
<td>search_parameters</td>
<td>array</td>
<td>Which search parameters does the target support? options: [general, title, id, author]</td>
</tr>
<tr>
<td>keywords</td>
<td>bool</td>
<td>Fetch keywords</td>
</tr>
<tr>
<td>keywords_upload</td>
<td>bool</td>
<td>Upload keywords to the target specific keyword attribute.</td>
</tr>
</tbody>
</table>

**Target JSON Example:**

```json
{
  "name": "osf",
  "readable_name": "OSF",
  "status_url": "https://api.osf.io/v2/nodes/",
  "token_url": "https://osf.io/settings/tokens",
  "supported_actions": {
    "resource_collection": true,
    "resource_detail": true,
    "resource_download": true,
    "resource_upload": true,
    "resource_transfer_in": true,
    "resource_transfer_out": true,
    "keywords": true,
    "keywords_upload": true
  },
  "supported_transfer_partners": {
    "transfer_in": ["github", "curate_nd"],
    "transfer_out": ["github"]
  }
}
```

(continues on next page)
"supported_hash_algorithms": ["sha256", "md5"],
"infinite_depth": true,
"search_parameters": ["title", "id", "general", "author"]
}

There is a management command that will validate targets.json that can be run after you add your target. It can be run manually with:

```bash
$ python manage.py validate_target_json
```

Otherwise the same management command is run when docker-compose up runs. If the validation fails then it does not allow the docker containers to be spun up.

2. Add your target directory inside presqt/targets/
   • Your target integration functionality will exist here.

### 6.2 Resource Endpoints

#### 6.2.1 Resource Collection

Targets that integrate with the Resources Collection API Endpoint must have a function that returns a specifically structured dataset.

1. Update your target in presqt/specs/targets.json by setting supported_actions.resource_collection to true.

2. Add a function to return the resource collection inside of your target directory.

   • If you would like to keep your file/function names consistent with what already exists add this function at presqt/targets/<target_name>/functions/fetch/<target_name>_fetch_resources()

   • The function must have the following parameters in this order:

     | Parameter       | Type  |
     |-----------------|-------|
     | token           | str   |
     | query_parameter | str   |

   • The function must return the following in this order:

     | Parameter       | Type                  |
     |-----------------|-----------------------|
     | resources       | list                  |
     | pages           | dict                  |

**Resource dictionary details:**
<table>
<thead>
<tr>
<th>kind</th>
<th>str</th>
<th>Type of Resource Options: [container, item]</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind_name</td>
<td>str</td>
<td>Target specific name for that kind For example OSF kind_names are: [project, folder, file]</td>
</tr>
<tr>
<td>container</td>
<td>str</td>
<td>ID of the container for the resource For example if the resource is a file in a folder then the container value would be the ID of the folder Can be None if the resource has no container</td>
</tr>
<tr>
<td>id</td>
<td>str</td>
<td>ID of the resource</td>
</tr>
<tr>
<td>title</td>
<td>str</td>
<td>Title of the resource</td>
</tr>
</tbody>
</table>

Page dictionary details:

<table>
<thead>
<tr>
<th>first_page</th>
<th>str</th>
<th>The first page number</th>
</tr>
</thead>
<tbody>
<tr>
<td>previous_page</td>
<td>str</td>
<td>The previous page number</td>
</tr>
<tr>
<td>next_page</td>
<td>str</td>
<td>The next page number</td>
</tr>
<tr>
<td>last_page</td>
<td>str</td>
<td>The last page number</td>
</tr>
<tr>
<td>total_pages</td>
<td>str</td>
<td>The total amount of pages</td>
</tr>
<tr>
<td>per_page</td>
<td>str</td>
<td>The amount of resources per page</td>
</tr>
</tbody>
</table>

Example Resource Collection Function:

```python
def <your_target_name>_fetch_resources(token, query_parameter):
    # Process to obtain resource collection IF search_parameter goes here.
    # Variables below are defined here to show examples of structure.
    target_resources = get_target_resources()

    resources = []
    for resource in target_resources:
        resource.append({
            'kind': 'container',
            'kind_name': 'Project',
            'id': resource.id,
            'container': None,
            'title': resource.title
        })

    # Process to obtain page numbers goes here
```
(continues on next page)
3. Add the resource collection function to `presqt/api_v1/utilities/utils/function_router.py`
   - Follow the naming conventions laid out in this class’ docstring
   - This will make the function available in core PresQT code

### 6.2.2 Resource Detail

Targets that integrate with the Resources Detail API Endpoint must have a function that returns a specifically structured dataset that represents the resource.

1. Update your target in `presqt/specs/targets.json` by setting `supported_actions.resource_detail` to `true`.

2. Add a function to return the resource details inside of your target directory.
   - If you would like to keep your file/function names consistent with what already exists add this function at `presqt/targets/<target_name>/functions/fetch/<target_name>_fetch_resource()`
   - The function must have the following parameters in this order:

     | Parameter   | Type  | Description                             |
     |-------------|-------|-----------------------------------------|
     | token       | str   | User’s token for the target             |
     | resource_id | str   | ID for the resource we want to fetch    |

   - The function must return the following in this order:

     | Parameter  | Type  | Description                              |
     |------------|-------|------------------------------------------|
     | resource   | object| Python object representing the resource requested |

**Resource dictionary details:**
kind | str | Type of Resource Options: [container, item]

kind_name | str | Target specific name for that kind For example OSF kind_names are: [node, folder, file]

id | str | ID of the resource

title | str | Title of the resource
date_created | str | Date the resource was created
date_modified | str | Date the resource was last modified

hashes | dict | Hashes of the resource in the target Key must be the hash algorithm used value must be the hash itself Can be an empty dict if no hashes exist

extra | dict | Any extra target specific data. Can be an empty dict

children | list | A list of children resources, each child in the list must be a dictionary that follows the structure of the resource_collection dictionaries listed above. Example: [{‘kind’: ‘’,
‘kind_name’: ‘’,
‘id’: ‘’,
‘container’: ‘’,
‘title’: ‘’}]

Example Resource Collection Function:

```python
def <your_target_name>_fetch_resource(token, resource_id):
    # Process to obtain resource details goes here.
    # Variables below are defined here to show examples of structure.
    # Variables below are defined here to show examples of structure.
    resource = {
        "kind": "item",
        "kind_name": "file",
        "id": "12345",
        "title": "o_o.jpg",
        "date_created": "2019-05-13T14:54:17.129170Z",
        "date_modified": "2019-05-13T14:54:17.129170Z",
        "hashes": {
            "md5": "abca7ef057dcb7cb8d79c36243823e4",
            "sha256": "ea94ce55261720c56aabb508c66dcdf1f481c30c09b7f2f5ab0b79e3199b7e2b55"
        },
        "extra": {
            "category": "project",
            "fork": false,
            "current_user_is_contributor": true,
            "preprint": false,
            "current_user_permissions": [
                "read",
                "write",
                "admin"
            ],
        },
        "children": []
    }
    return resource
```

3. Add the resource detail function to `presqt/api_v1/utilities/utils/function_router.py`

- Follow the naming conventions laid out in this class’ docstring
• This will make the function available in core PresQT code

6.3 Resource Download Endpoint

1. Update your target in presqt/specs/targets.json by setting supported_actions.resource_download to true.

2. Add a function to perform the resource download inside of your target directory.
   • If you would like to keep your file/function names consistent with what already exists add this function at presqt/targets/<target_name>/functions/download/<target_name>_download_resource()
   • The function must have the following parameters in this order:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td>str</td>
<td>User’s token for the target</td>
</tr>
<tr>
<td>resource_id</td>
<td>str</td>
<td>ID for the resource we want to download</td>
</tr>
<tr>
<td>process_info_path</td>
<td>str</td>
<td>The path to this download’s process_info_path</td>
</tr>
<tr>
<td>action</td>
<td>str</td>
<td>The type of action occurring</td>
</tr>
</tbody>
</table>

• The function must return a dictionary with the following keys:

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resources</td>
<td>List of dictionaries containing resource data</td>
</tr>
<tr>
<td>empty_containers</td>
<td>List of strings identifying empty container paths. They need to be specified separately because they are written separate from the file data</td>
</tr>
<tr>
<td>action_metadata</td>
<td>Dictionary containing FTS metadata about the action occurring</td>
</tr>
<tr>
<td>extra_metadata</td>
<td>Dictionary containing extra metadata identified by partners</td>
</tr>
</tbody>
</table>

Resource Dictionary Details

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>bytes</td>
<td>The file contents in byte format</td>
</tr>
<tr>
<td>hashes</td>
<td>dict</td>
<td>Hashes of the resource in the target</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key must be the hash algorithm used value must be the hash itself</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Can be an empty dict if no hashes exist</td>
</tr>
<tr>
<td>title</td>
<td>str</td>
<td>Title of the file</td>
</tr>
<tr>
<td>path</td>
<td>str</td>
<td>Path to save the file to at the destination</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Start the path with a /</td>
</tr>
<tr>
<td>source_path</td>
<td>dict</td>
<td>Full path of the file at the source</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Start the path with a /</td>
</tr>
<tr>
<td>extra_metadata</td>
<td>dict</td>
<td>Dictionary containing any extra data to save to FTS metadata</td>
</tr>
</tbody>
</table>

Action Metadata Dictionary Details

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceUsername</td>
<td>str</td>
<td>Username of the user making the request at the source target</td>
</tr>
</tbody>
</table>
Extra Metadata Dictionary Details

<table>
<thead>
<tr>
<th>title</th>
<th>str</th>
<th>The title of the resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>creators</td>
<td>list</td>
<td>List of dictionaries containing creator info</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{&quot;first_name&quot;: &quot;&quot;, &quot;last_name&quot;: &quot;&quot;, &quot;ORCID&quot;: &quot;&quot;}</td>
</tr>
<tr>
<td>publication_date</td>
<td>str</td>
<td>The date the resource was published</td>
</tr>
<tr>
<td>description</td>
<td>str</td>
<td>A brief description of the resource</td>
</tr>
<tr>
<td>keywords</td>
<td>list</td>
<td>A list of associated keywords</td>
</tr>
<tr>
<td>license</td>
<td>str</td>
<td>The resource’s license</td>
</tr>
<tr>
<td>related_identifiers</td>
<td>list</td>
<td>A list of dictionaries containing identifiers {&quot;type&quot;: &quot;doi&quot;, &quot;identifier&quot;: &quot;&quot;}</td>
</tr>
<tr>
<td>references</td>
<td>str</td>
<td>References related to the resource</td>
</tr>
<tr>
<td>notes</td>
<td>str</td>
<td>Notes related to the resource</td>
</tr>
</tbody>
</table>

- If you want to keep track of the progress of the download there are two functions available to do so. `update_process_info()` is for updating the total number of resources in the download and `increment_process_info()` is for updating the number of resources gathered thus far.

Example Resource Download Function:

```python
def <your_target_name>_download_resource(token, resource_id, process_info_path):
    # Process to download resource goes here.
    # Variables below are defined here to show examples of structure.
    resources = [
        
            'file': binary_file_contents, 
            'hashes': { 'md5': 'lab2c3d4e5f6g', 'sha256': 'fh3383h83fh' }, 
            'title': 'file.jpg', 
            'path': '/path/to/file.jpg', 
            'source_path': 'project_name/path/to/file.jpg', 
            'extra_metadata': { 
                'dateSubmitted': '2019-10-22Z', 
                'creator': 'Justin Branco', 
            }, 
        
        'file': binary_file_contents, 
        'hashes': { 'md5': 'zadf23fg3', 'sha256': '9382hash383h' }, 
        'title': 'funnysong.mp3', 
        'path': '/path/to/file/funnysong.mp3' 
        'source_path': 'project_name/path/to/file/funnysong.mp3', 
        'extra_metadata': { 
            'dateSubmitted': '2019-10-22Z', 
            'creator': 'Justin Branco', 
        }, 
    ]
```

(continues on next page)
empty_containers = ['path/to/empty/container/', 'another/empty/']

action_metadata = {"sourceUsername": contributor_name}
extra_metadata = {
    "title": project_info['title'],
    "creators": creators,
    "publication_date": project_info['date_created'],
    "description": project_info['description'],
    "keywords": project_info['tags'],
    "license": license,
    "related_identifiers": identifiers,
    "references": None,
    "notes": None
}

return {
    'resources': files,
    'empty_containers': empty_containers,
    'action_metadata': action_metadata,
    'extra_metadata': extra_metadata
}

3. Add the resource download function to presqt/api_v1/utilities/utils/function_router.py
   - Follow the naming conventions laid out in this class’ docstring
   - This will make the function available in core PresQT code

6.4 Resource Upload Endpoint

1. Update your target in presqt/specs/targets.json by setting supported_actions.resource_upload to true.

2. Add a function to perform the resource upload inside of your target directory.
   - If you would like to keep your file/function names consistent with what already exists add this function at presqt/targets/<target_name>/functions/upload/<target_name>_upload_resource()
   - The function must have the following parameters in this order:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td>str</td>
<td>User’s token for the target</td>
</tr>
<tr>
<td>resource_id</td>
<td>str</td>
<td>ID of the resource requested</td>
</tr>
<tr>
<td>resource_main_dir</td>
<td>str</td>
<td>Path to the main directory on the server for the resources to be uploaded</td>
</tr>
<tr>
<td>hash_algorithm</td>
<td>str</td>
<td>Hash algorithm we are using to check for fixity</td>
</tr>
<tr>
<td>file_duplicate_action</td>
<td>str</td>
<td>The action to take when a duplicate file is found</td>
</tr>
<tr>
<td>process_info_path</td>
<td>str</td>
<td>The path to this download’s process_info_path</td>
</tr>
<tr>
<td>action</td>
<td>str</td>
<td>The type of action occurring</td>
</tr>
</tbody>
</table>

   - The function must return a dictionary with the following keys:
re/sources_ignored

Array of string paths of files that were ignored when uploading the resource
Path should have the same base as resource_main_dir

resources_updated

Array of string paths of files that were updated when uploading the resource
Path should have the same base as resource_main_dir

file_metadata_list

List of dictionaries that contains FTS metadata and hash info for each file

action_metadata

Dictionary containing FTS metadata about the action occurring

project_id

str
ID of the parent project for this upload. Needed for metadata upload

project_link

str
The link to either the resource or the home page of the user if not available through API

Metadata Dictionary Details

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionRootPath</td>
<td>str</td>
<td>Original path of the file on the server before upload. This is used to connect this metadata with download metadata if the action is a transfer.</td>
</tr>
<tr>
<td>destinationHash</td>
<td>dict</td>
<td>Hash of the resource in the target that was calculated using the hash_algorithm given as a function parameter. Key must be the hash algorithm used, value must be the hash itself. Can be an empty dict if no hashes exist.</td>
</tr>
<tr>
<td>destinationPath</td>
<td>str</td>
<td>Full path of the file at the destination. Start the path with a /</td>
</tr>
<tr>
<td>title</td>
<td>str</td>
<td>Title of the file</td>
</tr>
</tbody>
</table>

Action Metadata Dictionary Details

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destinationUsername</td>
<td>str</td>
<td>Username of the user making the request at the destination target</td>
</tr>
</tbody>
</table>

Example Resource Upload Function:

```python
def <your_target_name>_upload_resource(token, resource_id, resource_main_dir, hash_algorithm, file_duplicate_action):
    # Process to upload resource goes here.
    # Variables below are defined here to show examples of structure.
    file_metadata_list = [
        {
            "actionRootPath": resource_main_dir/path/to/updated/file.jpg,
        }
    ]
```

(continues on next page)
3. Add a function to upload FTS metadata to the correct location within the resource’s parent project.

- If you would like to keep your file/function names consistent with what already exists add this function at `presqt/targets/<target_name>/functions/upload_metadata/<target_name>_upload_metadata()`

- The function must have the following parameters in this order:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td>str</td>
<td>User’s token for the target</td>
</tr>
<tr>
<td>project_id</td>
<td>str</td>
<td>The id of the parent project for the resource uploaded</td>
</tr>
<tr>
<td>metadata_dict</td>
<td>dict</td>
<td>The FTS metadata dictionary to upload</td>
</tr>
</tbody>
</table>

- The function doesn’t return anything

**Example Resource Upload Function:**

```python
def <your_target_name>_upload_metadata(token, project_id, metadata_dict):
    # Process to upload metadata goes here.

    # If you want to upload the extra metadata to fields supported by your API
    # you will have to add that functionality as well. The extra values are stored
    # in metadata_dict['extra_metadata']. IE:
    # update_project_with_metadata(url, metadata_dict['extra_metadata'])
```

3. Add the resource upload and upload metadata functions to `presqt/api_v1/utilities/utils/function_router.py`

- Follow the naming conventions laid out in this class’ docstring
- This will make the function available in core PresQT code
6.5 Resource Transfer Endpoint

1. Update your target in `presqt/specs/targets.json` by setting `supported_actions.resource_transfer_in`, `supported_actions.resource_transfer_out`, `supported_actions.supported_transfer_partners.transfer_in`, and `supported_actions.supported_transfer_partners.transfer_out` appropriately.

The resource transfer endpoint utilizes the Download and Upload functions. If these two functions are in place then transfer is available.

2. To support Keyword Enhancement during the transfer process, add keyword functions as outlined below in the Keyword Enhancement Endpoint section.

6.6 Keyword Enhancement Endpoint

Targets that want the ability to suggest or enhance new keywords must provide keyword functions.

6.6.1 Suggest Keywords

To support the suggestion of keywords, a keyword fetch function must be written that will fetch keywords from the target.

1. Update your target in `presqt/specs/targets.json` by setting `keywords` to `true`.

2. Add a function to return a dictionary of keywords found in the target.

   - If you would like to keep your file/function names consistent with what already exists add this function at `presqt/targets/<target_name>/functions/keywords/<target_name>_fetch_keywords()`.
   - The function must have the following parameters in this order:

     - `token` str: User’s token for the target
     - `resource_id` str: ID for the resource we want to get keywords from

   - The function must return a dictionary with the following keys:

     - `keywords` array: Array of keywords found in the target
     - `<attribute_name>` array: Array of keywords found for this attribute

     Name the key whatever the attribute name is. See example for more details.

Example Keyword Fetch Function:

```python
def <your_target_name>_fetch_keywords(token, resource_id):
    # Process to fetch keywords goes here.
    # Variables below are defined here to show examples of structures.
    # This target has keywords in two attributes, 'topics' and 'tags'.
    keyword_dictionary = {
```

(continues on next page)
3. Add the keyword fetch function to `presqt/api_v/utilities/utils/function_router.py`
   - Follow the naming conventions laid out in this class’ docstring
   - This will make the function available in core PresQT code

### 6.6.2 Enhance Keywords

To support the enhancement of keywords, a keyword upload function must be written that will upload new enhanced keywords to the target.

1. Update your target in `presqt/specs/targets.json` by setting `keywords_upload` to `true`.
2. Add a function to upload give keywords to the target.
   - If you would like to keep your file/function names consistent with what already exists add this function to `presqt/targets/<target_name>/functions/keywords/<target_name>_upload_keywords()
   - The function must have the following parameters **in this order**:
     - `token` (`str`) - User’s token for the target
     - `resource_id` (`str`) - ID for the resource we want to upload keywords to
     - `keywords` (`list`) - List of new keywords to upload
   - The function must return a dictionary with the following keys:
     - `updated_keywords` (`list`) - List of the final keyword list at the target
     - `project_id` (`str`) - The ID of the project containing this resource

**Example Keyword Upload Function:**

```python
def <your_target_name>_upload_keywords(token, resource_id, keywords):
    updated_keywords = ['cat', 'food', 'feline', 'grub']
    project_id = '1234'
    return {'updated_keywords': updated_keywords, 'project_id': project_id}
```

3. Add the keyword upload function to `presqt/api_v/utilities/utils/function_router.py`
   - Follow the naming conventions laid out in this class’ docstring
   - This will make the function available in core PresQT code
6.7 Error Handling

When any of these target functions are called within PresQT core code they are wrapped inside of a Try-Except clause which looks for the exception PresQTResponseException. The definition of this exception can be found at presqt.utilities.exceptions.exceptions.PresQTResponseException.
7.1 Authentication

Refer to the authentication details here.

7.2 Duplicate File Handling

When uploading or transferring resources a header, presqt-file-duplicate-action, must be included. The options are ignore or update. This header tells the target uploading the resource what to do when a file being uploaded already exists in the source target.

Ignore will not update the duplicate file, even if the contents of the files don’t match.

Update will only update the duplicate file if the contents of the files don’t match.

7.3 Searching Resource Collections

Search results are ordered by date modified unless the target does not support it.

Only a single search filter can be used at a time.

7.3.1 Search Filters

General search across all available target search parameters: resources/?general=search_value

Search by project ‘title’: resources/?title=Project+Title

Search by project ‘id’: resources/?id=123456

Search by project ‘author’: resources/?author=bfox6

Search by project ‘keywords’: resources/?keywords=cat

7.4 Paginating Resource Collections

Pagination has been added at the collection level to improve load times. Targets now return Pagination information for users resources, as well as searched resources.
7.4.1 Page Parameter

Pagination across all available targets: resources/?page=page_number

7.5 Target Endpoints

7.5.1 Target Collection

GET /api_v1/targets/

Retrieve details of all Targets.

Example request:

```
GET /api_v1/targets/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[

{

"name": "osf",
"readable_name": "OSF",
"status_url": "https://api.osf.io/v2/nodes/",
"token_url": "https://osf.io/settings/tokens",
"supported_actions": {

"resource_collection": true,
"resource_detail": true,
"resource_download": true,
"resource_upload": true,
"resource_transfer_in": true,
"resource_transfer_out": true,
"keywords": true,
"keywords_upload": true,
},

"supported_transfer_partners": {

"transfer_in": [github,
curate_nd
],

"transfer_out": [github

],

"supported_hash_algorithms": [sha256,
md5
],

"infinite_depth": true

"links": ["name": "Detail",

```
```
7.5.2 Target Details

GET /api_v1/targets/(str: target_name)/
Retrieve details of a single Target.

Example request:

```
GET /api_v1/targets/OSF/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Status Codes

- 200 OK – Targets successfully retrieved
Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "name": "osf",
   "readable_name": "OSF",
   "status_url": "https://api.osf.io/v2/nodes/",
   "token_url": "https://osf.io/settings/tokens",
   "supported_actions": {
      "resource_collection": true,
      "resource_detail": true,
      "resource_download": true,
      "resource_upload": true,
      "resource_transfer_in": true,
      "resource_transfer_out": true,
      "keywords": true,
      "keywords_upload": true,
   },
   "supported_transfer_partners": {
      "transfer_in": [
         "github",
         "curate_nd"
      ],
      "transfer_out": [
         "github"
      ]
   },
   "supported_hash_algorithms": [
      "sha256",
      "md5"
   ],
   "infinite_depth": true
   "links": [
      {
         "name": "Collection",
         "link": "https://presqt-prod.crc.nd.edu/api_v1/targets/osf/resources/
         "method": "GET"
      },
      {
         "name": "Upload",
         "link": "https://presqt-prod.crc.nd.edu/api_v1/targets/osf/resources/
         "method": "POST"
      },
      {
         "name": "Transfer",
         "link": "https://presqt-prod.crc.nd.edu/api_v1/targets/osf/resources/
         "method": "POST"
      }
   ]
}
```

Status Codes

- **200 OK** – Target successfully retrieved
7.6 Resource Endpoints

7.6.1 Resource Collection

**GET /api_v1/targets/(str: target_name)/resources/**
Retrieve details of all top level resources for a given Target and User Token

Example request:
```
GET /api_v1/targets/OSF/resources/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Example response:
```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "resources": [
    {
      "kind": "container",
      "kind_name": "project",
      "id": "cmn5z",
      "container": null,
      "title": "Test Project",
      "links": [
        {
          "name": "Detail",
          "link": "https://presqt-prod.crc.nd.edu/api_v1/targets/osf/resources/cmn5z/",
          "method": "GET"
        }
      ]
    },
    {
      "kind": "container",
      "kind_name": "project",
      "id": "12345",
      "container": null,
      "title": "Egg Project",
      "links": [
        {
          "name": "Detail",
          "link": "https://presqt-prod.crc.nd.edu/api_v1/targets/osf/resources/12345/",
          "method": "GET"
        }
      ]
    }
  ],
  "pages": {
    "first_page": "https://presqt-prod.crc.nd.edu/api_v1/targets/osf/resources?page=1",
    ...
  }
}
```
Example request w/ search parameter:

```plaintext
GET /api_v1/targets/OSF/resources?title=egg/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Example request w/ search parameter and page parameter:

```plaintext
GET /api_v1/targets/OSF/resources?title=egg&page=3/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Search filtering rules can be found here.

Request Headers

- `presqt-source-token` – User’s token for the source target

Status Codes

- 200 OK – Resources successfully retrieved
- 400 Bad Request – The Target does not support the action `resource_collection`
- 400 Bad Request – `presqt-source-token` missing in the request headers
- 400 Bad Request – The search query is not formatted correctly.
- 401 Unauthorized – Token is invalid
- 404 Not Found – Invalid Target name

### 7.6.2 Resource Detail

**GET /api_v1/targets/(str: target_name)/resources/(str: resource_id).json/**

Retrieve details of a Resource in JSON format

Example request:

```plaintext
GET /api_v1/targets/OSF/resources/1234.json/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Example response:
HTTP/1.1 200 OK
Content-Type: application/json

{
    "kind": "container",
    "kind_name": "project",
    "id": "cmn5z",
    "title": "Test Project",
    "date_created": "2019-05-13T15:06:34.521000Z",
    "date_modified": "2019-05-13T15:06:34.521000Z",
    "hashes": {
        "md5": null,
        "sha256": null
    },
    "extra": {
        "last_touched": "2019-11-07T17:00:51.680957",
        "materialized_path": "/Test Project",
        "current_version": 1,
        "provider": "googledrive",
        "path": "/Test Project",
        "current_user_can_comment": true,
        "guid": "byz93",
        "checkout": null,
        "tags": [],
        "size": null
    },
    "children": [
        {
            "kind": "container",
            "kind_name": "storage",
            "id": "cmn5z:osfstorage",
            "container": "cmn5z",
            "title": "osfstorage",
            "links": [
                {
                    "name": "Detail",
                    "link": "https://presqt-prod.crc.nd.edu/api_v1/targets/osf/resources/cmn5z:osfstorage/",
                    "method": "GET"
                }
            ]
        },
        {
            "kind": "container",
            "kind_name": "folder",
            "id": "5cd9832cf244ec0021e5f245",
            "container": "cmn5z:osfstorage",
            "title": "Images",
            "links": [
                {
                    "name": "Detail",
                    "link": "https://presqt-prod.crc.nd.edu/api_v1/targets/osf/resources/5cd9832cf244ec0021e5f245/",
                    "method": "GET"
                }
            ]
        }
    ],
}

(continues on next page)
Request Headers

- **presqt-source-token** – User’s token for the source target

Status Codes

- **200 OK** – Resource successfully retrieved
- **400 Bad Request** – The Target does not support the action resource_detail
- **400 Bad Request** – presqt-source-token missing in the request headers
- **400 Bad Request** – Invalid format given. Must be json
- **401 Unauthorized** – Token is invalid
- **403 Forbidden** – User does not have access to this Resource
- **404 Not Found** – Invalid Target name
• 404 Not Found – Resource with this ID not found for this user
• 410 Gone – Resource no longer available

7.7 Resource Download Endpoints

7.7.1 Download Resource

GET /api_v1/targets/(str: target_name)/resources/(str: resource_id).zip/  
Retrieve a Resource as a ZIP file. This endpoint begins the download process but does not return the zip file. Rather, it returns a link which can be used to hit the Job Status endpoint to check in on the process.

Example request:

GET /api_v1/targets/OSF/resources/1234.zip/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json

Example response:

HTTP/1.1 202 Accepted
Content-Type: application/json

{  
"message": "The server is processing the request.",  
"download_job_zip": "https://presqt-prod.crc.nd.edu/api_v1/job_status/download.zip/",  
"download_job_json": "https://presqt-prod.crc.nd.edu/api_v1/job_status/download.json/"
}

Request Headers

• presqt-source-token – User’s token for the source target

Status Codes

• 202 Accepted – Resource has begun downloading
• 400 Bad Request – The Target does not support the action resource_download
• 400 Bad Request – User currently has processes in progress.
• 400 Bad Request – presqt-source-token missing in the request headers
• 400 Bad Request – presqt-email-opt-in missing in the request headers
• 400 Bad Request – Invalid format given. Must be zip
• 404 Not Found – Invalid Target name

7.7.2 Resource Download Job Status

GET /api_v1/job_status/download.json/  
Use the Job Status endpoint to check in on the Download Process. Provide the presqt-source-token in the headers.

Example request
GET /api_v1/job_status/download/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json

Example response if the download request is still in progress:

HTTP/1.1 202 Accepted
Content-Type: application/json

{
   "job_percentage": 27,
   "status": "in_progress",
   "status_code": null,
   "message": "Downloading files from OSF..."
}

Example response if the download request finished successfully:

HTTP/1.1 200 OK
Content-Type: application/json

{
   "status_code": "200",
   "message": "Download successful. See PRESQT_FTS_METADATA.json for more details."
   "zip_name": "osf_download_cmn5z.zip",
   "failed_fixity": [
      "Test Project/googledrive/PresQT Swimlane Activity Diagram 03_21_19 (2).pdf",
      "Test Project/googledrive/module_responses.csv",
      "Test Project/googledrive/Google Images/IMG_4740.jpg",
      "Test Project/googledrive/Character Sheet - Alternative - Print Version. pdf"
   ],
   "job_percentage": 100,
   "status": "finished"
}

Example response if the download failed:

HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
   "job_percentage": 0,
   "status": "failed",
   "status_code": 404,
   "message": "Resource with id 'bad_id' not found for this user."
}

Request Headers

- presqt-source-token – User's Token for the source target

Status Codes

- 200 OK – Download has finished successfully
- 202 Accepted – Download is being processed on the server
• 400 Bad Request – presqt-source-token missing in the request headers
• 400 Bad Request – Invalid format given. Must be json or zip.
• 404 Not Found – Invalid Ticket Number
• 500 Internal Server Error – Download failed on the server

GET /api_v1/job_status/download.zip/
Check on the Download Process for the given user. If download has failed or is in progress this endpoint will return a JSON payload detailing this. If download has completed this endpoint will return the zip file of the resource originally requested.

Example request:

```
GET /api_v1/job_status/download.zip/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Example response if the download request is still in progress:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

{
  "job_percentage": 27,
  "status": "in_progress",
  "status_code": null,
  "message": "Downloading files from OSF..."
}
```

Example response if download finished successfully:

```
HTTP/1.1 200 OK
Content-Type: application/zip

Payload is ZIP file
```

Example response if the download failed:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
  "job_percentage": 0,
  "status": "failed",
  "status_code": 404,
  "message": "Resource with id 'bad_id' not found for this user."
}
```

Request Headers

• presqt-source-token – User’s Token for the source target

Status Codes

• 200 OK – Download has finished successfully
• 202 Accepted – Download is being processed on the server
• 400 Bad Request – presqt-source-token missing in the request headers
• 400 Bad Request – Invalid format given. Must be json or zip.
• 404 Not Found – Invalid Ticket Number
• 500 Internal Server Error – Download failed on the server

PATCH /api_v1/job_status/upload/
Cancel the Download Process for the given user.

If the download has finished before it can be cancelled it will return the finished info from process_info.json.

If the download was successfully cancelled then it will return the cancelled info from process_info.json.

Example request:

```plaintext
PATCH /api_v1/job_status/download/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Example response if download cancelled successfully:

```plaintext
HTTP/1.1 200 OK
Content-Type: application/json

{  
  "status_code": "499",
  "message": "Download was cancelled by the user"
}
```

Example response if download finished before endpoint was able to cancel:

```plaintext
HTTP/1.1 406 OK
Content-Type: application/json

{  
  "status_code": "200",
  "message": "Download successful."
}
```

Request Headers
• presqt-source-token – User’s Token for the source target

Status Codes
• 200 OK – Download cancelled
• 406 Not Acceptable – Download finished before cancellation
• 400 Bad Request – presqt-source-token missing in the request headers
• 404 Not Found – Invalid Ticket Number

7.8 Resource Upload Endpoints

7.8.1 Upload New Top Level Resource

**POST /api_v1/targets/(str: target_name)/resources/**

Upload a new top level resource, for instance a Project. This endpoint begins the Upload process. It returns a
link which can be used to the hit the Job Status endpoint to check in on the process.

**Example request:**

```plaintext
POST /api_v1/targets/OSF/resources/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json

```

**Example response:**

```plaintext
HTTP/1.1 202 Accepted
Content-Type: application/json

{   "message": "The server is processing the request.",
   "upload_job": "https://presqt-prod.crc.nd.edu/api_v1/job_status/upload/"
}
```

**Request Headers**

- `presqt-destination-token` – User’s Token for the destination target
- `presqt-file-duplicate-action` – Action to be taken if a duplicate file is found (Either `update` or `ignore`)

**Form Parameters**

- `presqt-file` – The Resource to Upload. Must be a BagIt file in ZIP format.

**Status Codes**

- 202 Accepted – Resource has begun uploading
- 400 Bad Request – The Target does not support the action `resource_upload`
- 400 Bad Request – `presqt-destination-token` missing in the request headers
- 400 Bad Request – The file, `presqt-file`, is not found in the body of the request
- 400 Bad Request – The file provided is not a zip file
- 400 Bad Request – The file provided is not in BagIt format
- 400 Bad Request – Checksums failed to validate
- 400 Bad Request – `presqt-file-duplicate-action` missing in the request headers
- 400 Bad Request – `presqt-email-opt-in` missing in the request headers
- 400 Bad Request – Invalid `file_duplicate_action` header given. The options are `ignore` or `update`
- 400 Bad Request – Repository is not formatted correctly. Multiple directories exist at the top level
- 400 Bad Request – Repository is not formatted correctly. Files exist at the top level
- 400 Bad Request – User currently has processes in progress.
- 401 Unauthorized – Token is invalid
- 404 Not Found – Invalid Target name
### 7.8.2 Upload To Existing Resource

**POST /api_v1/targets/(str: target_name)/resources/(str: resource_id)/**

Upload a resource to an existing container. This endpoint begins the upload process. It returns a link which can be used to check the Job Status endpoint to check in on the process.

**Example request:**

```plaintext
POST /api_v1/targets/OSF/resources/1234/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

**Example response:**

```plaintext
HTTP/1.1 202 Accepted
Content-Type: application/json

{
    "message": "The server is processing the request."
}
```

#### Request Headers

- `presqt-destination-token` – User’s Token for the destination target
- `presqt-file-duplicate-action` – Action to be taken if a duplicate file is found (Either update or ignore)

#### Form Parameters

- `presqt-file` – The Resource to Upload. Must be a BagIt file in ZIP format.

#### Status Codes

- 202 Accepted – Resource has begun uploading
- 400 Bad Request – The Target does not support the action resource_upload
- 400 Bad Request – `presqt-destination-token` missing in the request headers
- 400 Bad Request – `presqt-email-opt-in` missing in the request headers
- 400 Bad Request – The file, `presqt-file`, is not found in the body of the request
- 400 Bad Request – The file provided is not a zip file
- 400 Bad Request – The file provided is not in BagIt format
- 400 Bad Request – Checksums failed to validate
- 400 Bad Request – `presqt-file-duplicate-action` missing in the request headers
- 400 Bad Request – Invalid `file_duplicate_action` header given. The options are ignore or update
- 400 Bad Request – User currently has processes in progress.
- 401 Unauthorized – Token is invalid
- 403 Forbidden – User does not have access to this Resource
- 404 Not Found – Invalid Target name
7.8.3 Resource Upload Job Status

GET /api_v1/job_status/upload/
Check on the Upload Process for the given user.

Example request:

```
GET /api_v1/job_status/upload/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Example response if the upload is in progress:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status_code": null,
    "message": "Uploading files to OSF...",
    "status": "in_progress",
    "job_percentage": 0
}
```

Example response if upload finished successfully:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status_code": "200",
    "message": "Upload successful.",
    "status": "finished",
    "failed_fixity": [],
    "resources_ignored": [],
    "resources_updated": [],
    "job_percentage": 99
}
```

Example response if upload failed:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "job_percentage": 0,
    "status": "failed",
    "status_code": 404,
    "message": "Resource with id 'bad_id' not found for this user."
}
```

Request Headers

- presqt-destination-token – User’s Token for the destination target

Status Codes

• 410 Gone – Resource no longer available
• 200 OK – Upload has finished successfully
• 202 Accepted – Upload is being processed on the server
• 400 Bad Request – presqt-destination-token missing in the request headers
• 404 Not Found – Invalid Ticket Number
• 500 Internal Server Error – Upload failed on the server

PATCH /api_v1/job_status/upload/
Cancel the Upload Process for the given user. If the upload has finished before it can be cancelled it will return the finished info from process_info.json. If the upload was successfully cancelled then it will return the cancelled info from process_info.json.

Example request:

PATCH /api_v1/job_status/upload/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json

Example response if upload cancelled successfully:

HTTP/1.1 200 OK
Content-Type: application/json

{   "status_code": "499",
    "message": "Upload was cancelled by the user"
}

Example response if upload finished before endpoint was able to cancel:

HTTP/1.1 406 OK
Content-Type: application/json

{   "status_code": "200",
    "message": "Upload successful."
}

Request Headers
• presqt-destination-token – User’s Token for the destination target

Status Codes
• 200 OK – Upload cancelled
• 406 Not Acceptable – Upload finished before cancellation
• 400 Bad Request – presqt-destination-token missing in the request headers
• 404 Not Found – Invalid Ticket Number

7.9 Resource Transfer Endpoints

Note: The Upload and Transfer endpoints are the same POST endpoints except the specification of where the source resource is coming from.
For **Uploads** the resource will be a file provided as form-data
For **Transfers** the location of resource (source_target and resource_id) will be specified in the body as JSON

### 7.9.1 Transfer New Top Level Resource

**POST /api_v1/targets/(str: target_name)/resources/**

Transfer a resource from a source target to a destination target. Make the resource a new top level resource, for instance a Project. This endpoint begins the Transfer process. It returns a link which can be used to hit the Job Status endpoint to check in on the process.

Example request:

```
POST /api_v1/targets/OSF/resources/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json

Example body json:
{
   "source_target_name" : "github",
   "source_resource_id" : "209372336",
   "keywords" : [ "keywords", "to", "add" ]
}
```

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

{
   "message": "The server is processing the request.",
   "transfer_job": "https://presqt-prod.crc.nd.edu/api_v1/job_status/transfer/
```}

**Request Headers**

- `presqt-destination-token` – User’s Token for the destination target
- `presqt-source-token` – User’s Token for the source target
- `presqt-file-duplicate-action` – Action to be taken if a duplicate file is found (Either update or ignore)
- `presqt-keyword-action` – Type of keyword action to perform (Either automatic, manual or none)

**JSON Parameters**

- `source_target_name (string)` – The Source Target where the Resource being Transferred exists
- `source_resource_id (string)` – The ID of the Resource to Transfer

**Status Codes**

- **202 Accepted** – Resource has begun transferring
- **400 Bad Request** – The Source Target does not support the action `resource_transfer_out`
• 400 Bad Request – The Destination Target does not support the action resource_transfer_in
• 400 Bad Request – presqt-source-token missing in the request headers
• 400 Bad Request – presqt-destination-token missing in the request headers
• 400 Bad Request – presqt-file-duplicate-action missing in the request headers
• 400 Bad Request – presqt-email-opt-in missing in the request headers
• 400 Bad Request – Invalid file-duplicate-action header given. The options are ignore or update
• 400 Bad Request – source_resource_id can’t be none or blank
• 400 Bad Request – source_resource_id was not found in the request body
• 400 Bad Request – source_target_name was not found in the request body
• 400 Bad Request – keywords was not found in the request body.
• 400 Bad Request – keywords must be in list format.
• 400 Bad Request – Source target does not allow transfer to the destination target
• 400 Bad Request – Destination target does not allow transfer to the source target
• 400 Bad Request – Invalid presqt-keyword-action header given. The options are automatic, manual, or none
• 400 Bad Request – presqt-keyword-action missing in the request headers
• 400 Bad Request – User currently has processes in progress.
• 401 Unauthorized – Source Token is invalid
• 401 Unauthorized – Destination Token is invalid
• 403 Forbidden – User does not have access to the Resource to transfer
• 404 Not Found – Invalid Source Target name
• 404 Not Found – Invalid Destination Target name
• 410 Gone – Resource to transfer is no longer available

7.9.2 Transfer To Existing Resource

POST /api_v1/targets/(str: target_name)/resources/(str: resource_id)/
Transfer a resource from a source target to a destination target. Transfer to an existing resource. This endpoint begins the Transfer process. It returns a link which can be used to the hit the Job Status endpoint to check in on the process.

Example request:

```
POST /api_v1/targets/OSF/resources/1234/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json

Example body json:
{
    "source_target_name":"github",
```
Example response:

```json
HTTP/1.1 202 Accepted
Content-Type: application/json

{
  "message": "The server is processing the request.",
  "transfer_job": "https://presqt-prod.crc.nd.edu/api_v1/job_status/transfer/"
}
```

**Request Headers**

- `presqt-destination-token` – User’s Token for the destination target
- `presqt-source-token` – User’s Token for the source target
- `presqt-file-duplicate-action` – Action to be taken if a duplicate file is found (Either `update` or `ignore`)
- `presqt-keyword-action` – Type of keyword action to perform (Either `automatic`, `manual`, or `none`)

**JSON Parameters**

- `source_target_name` *(string)* – The Source Target where the Resource being Transferred exists
- `source_resource_id` *(string)* – The ID of the Resource to Transfer

**Status Codes**

- **202 Accepted** – Resource has begun transferring
- **400 Bad Request** – The Source Target does not support the action `resource_transfer_out`
- **400 Bad Request** – The Destination Target does not support the action `resource_transfer_in`
- **400 Bad Request** – `presqt-source-token` missing in the request headers
- **400 Bad Request** – `presqt-destination-token` missing in the request headers
- **400 Bad Request** – `presqt-file-duplicate-action` missing in the request headers
- **400 Bad Request** – `presqt-email-opt-in` missing in the request headers
- **400 Bad Request** – Invalid `file_duplicate_action` header give. The options are `ignore` or `update`
- **400 Bad Request** – `source_resource_id` can’t be none or blank
- **400 Bad Request** – `source_resource_id` was not found in the request body
- **400 Bad Request** – `source_target_name` was not found in the request body
- **400 Bad Request** – `keywords` was not found in the request body.
- **400 Bad Request** – `keywords` must be in list format.

7.9. Resource Transfer Endpoints
• 400 Bad Request – Source target does not allow transfer to the destination target
• 400 Bad Request – Destination target does not allow transfer to the source target
• 400 Bad Request – Invalid presqt-keyword-action header given. The options are automatic, manual or none
• 400 Bad Request – presqt-keyword-action missing in the request headers
• 400 Bad Request – User currently has processes in progress.
• 401 Unauthorized – Source Token is invalid
• 401 Unauthorized – Destination Token is invalid
• 403 Forbidden – User does not have access to the Resource to transfer
• 403 Forbidden – User does not have access to the Resource to transfer to
• 404 Not Found – Invalid Source Target name
• 404 Not Found – Invalid Destination Target name
• 410 Gone – Resource to transfer is no longer available
• 410 Gone – Resource to transfer to is longer available

7.9.3 Resource Transfer Job Status

GET /api_v1/job_status/transfer/
Check on the Transfer Process for the given user.

Example request:

```
GET /api_v1/job_status/transfer/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Example response if transfer is in progress:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

{
    "status_code": null,
    "message": "Creating PRESQT_FTS_METADATA...",
    "job_percentage": 50
}
```

Example response if transfer finished successfully:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status_code": "200",
    "message": "Transfer successful.",
    "job_percentage": 99,
    "failed_fixity": [
        "/PrivateProject/README.md"
    ],
}
```

(continues on next page)
"resources_ignored": [],
"resources_updated": [],
"enhanced_keywords": [
    "EGG",
    "DISORDERED SOLVENT",
    "Electrostatic Gravity Gradiometer",
    "animal house",
    "aqua",
    "Wasser",
],
"initial_keywords": [
    "animals",
    "eggs",
    "water"
],
"source_resource_id": "209372336",
"destination_resource_id": "qadt3"
}

Example response if transfer failed:

HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "status_code": 404,
    "message": "The resource with id, 20938989898989872336, does not exist for this user."
    "job_percentage": 0,
    "status": "failed"
}

Request Headers

- presqt-destination-token – User’s Token for the destination target
- presqt-source-token – User’s Token for the source target

Status Codes

- 200 OK – Transfer has finished successfully
- 202 Accepted – Transfer is being processed on the server
- 400 Bad Request – presqt-destination-token missing in the request headers
- 400 Bad Request – presqt-source-token missing in the request headers
- 404 Not Found – Invalid Ticket Number
- 500 Internal Server Error – Transfer failed on the server

PATCH /api_v1/job_status/transfer/
Cancel the Transfer Process for the given user. If the transfer has finished before it can be cancelled it will return the finished info from process_info.json. If the transfer was successfully cancelled then it will return the cancelled info from process_info.json.

Example request:

7.9. Resource Transfer Endpoints
Example response if transfer cancelled successfully:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status_code": "499",
    "message": "Transfer was cancelled by the user"
}
```

Example response if transfer finished before endpoint was able to cancel:

```
HTTP/1.1 406 OK
Content-Type: application/json

{
    "status_code": "200",
    "message": "Transfer successful."
}
```

Request Headers

- `presqt-destination-token` – User’s Token for the destination target
- `presqt-source-token` – User’s Token for the source target

Status Codes

- 200 OK – Transfer cancelled
- 406 Not Acceptable – Transfer finished before cancellation
- 400 Bad Request – `presqt-destination-token` missing in the request headers
- 400 Bad Request – `presqt-source-token` missing in the request headers
- 404 Not Found – Invalid Ticket Number

### 7.10 Keyword Enhancement Endpoints

#### 7.10.1 Get a Resource’s Keywords And Keyword Enhancements

**GET /api_v1/targets/(str: target_name)/resources/(str: resource_id)/keywords/**

Retrieve a resource’s keywords that are both stored in the target and in the PresQT Metadata File (if one exists). Send the keywords to a keyword enhancer. Return both the Target Keywords and Enhanced Keywords in the payload.

Example request:

```
GET /api_v1/targets/OSF/resources/1234/keywords/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Example response:
HTTP/1.1 200 OK
Content-Type: application/json

{
    "keywords": [
        "eggs",
        "animal",
        "water"
    ],
    "enhanced_keywords": [
        "animals",
        "Animals",
        "EGG",
        "Electrostatic Gravity Gradiometer",
        "water",
        "Water",
        "DISORDERED SOLVENT",
        "aqua",
        "Wasser",
        "dihydrogen oxide",
        "OXYGEN ATOM",
        "oxidane"
    ],
    "all_keywords": [
        "animals",
        "Animals",
        "EGG",
        "Electrostatic Gravity Gradiometer",
        "water",
        "Water",
        "DISORDERED SOLVENT",
        "aqua",
        "Wasser",
        "dihydrogen oxide",
        "OXYGEN ATOM",
        "oxidane",
        "eggs",
        "animal",
        "water"
    ]
}

Request Headers

- presqt-source-token – User’s Token for the source target

Status Codes

- 200 OK – Keywords successfully fetched
- 400 Bad Request – The Source Target does not support the action keywords
- 400 Bad Request – The resource type does not support keywords
- 401 Unauthorized – Source Token is invalid

7.10. Keyword Enhancement Endpoints
7.10.2 Upload Keywords to a Resource

**POST /api_v1/targets/(str: target_name)/resources/(str: resource_id)/keywords/**

Take a list of keywords and add them to the Resource’s keywords both in the target and in the PresQT FTS Metadata file. The returned payload will contain both the new keywords added and the final full list of keywords in the target.

**Example request:**

```
POST /api_v1/targets/OSF/resources/1234/keywords/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json

Example body json:
{
   "keywords": ["cat", "water"]
}
```

**Example response:**

```
HTTP/1.1 202 Accepted
Content-Type: application/json

{
   "keywords_added": [
      "feline",
      "aqua",
      "dihydrogen oxide",
      "DISORDERED SOLVENT",
      "EGG",
      "Electrostatic Gravity Gradiometer",
      "oxidane",
      "OXYGEN ATOM",
      "Wasser",
      "Water"
   ],
   "final_keywords": [
      "feline",
      "aqua",
      "dihydrogen oxide",
      "DISORDERED SOLVENT",
      "EGG",
      "eggs",
      "Electrostatic Gravity Gradiometer",
      "oxidane",
      "OXYGEN ATOM",
      "Wasser",
      "water",
      "Water"
   ]
}
```

**Request Headers**

- `presqt-source-token` – User’s Token for the source target

**JSON Parameters**

- `keywords (array)` – An array of the keywords to upload
Status Codes

- **202 Accepted** – Keywords successfully uploaded
- **400 Bad Request** – The Source Target does not support the action keywords
- **400 Bad Request** – The Source Target does not support the action keywords_upload
- **400 Bad Request** – The resource type does not support keywords
- **400 Bad Request** – keywords is missing from the request body
- **400 Bad Request** – keywords must be in list format
- **401 Unauthorized** – Source Token is invalid
8.1 Fixity

8.1.1 Tools

- Python Hashlib Library
- BagIt Python Validation

8.1.2 PresQT Supported Hash Algorithms

The following is a master list of hash algorithms that are both supported by a target and supported by Python’s HashLib library:

- sha256
- md5

Each individual target’s supported hash algorithms can be found in presqt/specs/targets.json

8.1.3 Resource Download Fixity

Fixity is checked during Resource Download by comparing the file hashes provided by the source target with hashes that are generated after files are downloaded on to the server. If the provided hash and the calculated hash match then fixity passes!

The download function will try and find a matching hash algorithm between the source target supported algorithms and algorithms supported by the Python Hashlib library to use when generating hashes for files downloaded to the server. If no hash algorithms match or if the source target does not provide file hashes then md5 is uses as a default. It also counts this situation as fixity passing since we didn’t know what the original hash was.

Valid Hashes Provided + Fixity Passes Example:

```json
{
    "sha256": "343e249fdb0818a58edcc64663e1eb116843b4e1c4e74790ff331628593c02be",
    "md5": "a4536efb47b26eaf509edfdaca44203c"
}
```

will yield
Valid Hashes Provided + Fixity Fails Example:

```json
{
  "hash_algorithm": "sha256",
  "given_hash": "343e249fdb0818a58edcc64663e1eb116843b4e1c4e74790ff331628593c02be",
  "calculated_hash": "343e249fdb0818a58edcc64663e1eb116843b4e1c4e74790ff331628593c02be",
  "fixity": true
}
```

will yield

```json
{
  "hash_algorithm": "sha256",
  "given_hash": "343e249fdb0818a58edcc64663e1eb116843b4e1c4e74790ff331628593c02be",
  "calculated_hash": "12345678",
  "fixity": false
}
```

Blank Hashes Provided Example:

```json
{
  "sha256": null,
  "md5": null
}
```

will yield

```json
{
  "hash_algorithm": "md5",
  "given_hash": null,
  "calculated_hash": "343e249fdb0818a58edcc64663e1eb116843b4e1c4e74790ff331628593c02be",
  "fixity": true
}
```

Unknown Hashes Provided Example:

```json
{
  "unknown_hasher": "12345",
  "special_hasher": "1234567"
}
```

will yield

```json
{
  "hash_algorithm": "md5",
  "given_hash": null,
  "calculated_hash": "343e249fdb0818a58edcc64663e1eb116843b4e1c4e74790ff331628593c02be",
  "fixity": true
}
```
8.1.4 Resource Upload Fixity

During the resource upload process, fixity is checked in two locations. First, when files are saved to the disk from the request. Second, after files are uploaded to the target.

Fixity Check 1

Resources must be included in the POST request in BagIt format as a zip file. After unzipping the file and saving it to the server we validate the bag using BagIt’s built-in validator. If any files saved don’t match the manifest originally given then the fixity has failed and the server will return an error.

Generate New Hashes If Necessary

We now know that the currently saved files are the same as what the user sent forward. Before uploading resources to the target we will make sure that there is a dictionary of hashes available generated by a hash algorithm supported by the target. If the target supports a hash algorithm provided by the resource’s ‘bag’ then we will simply use those. If not, then we need to generate new hashes based on a target supported hash algorithm.

Fixity Check 2

After resources are uploaded to the target, we compare the resources’ hashes brought back from the target to the hashes we captured before. If any hashes don’t match then fixity fails. Since the resources have already been uploaded we simply capture which resources’ fixity fails and pass that along the response payload along with the message, ‘Upload successful but fixity failed’.

8.1.5 Resource Transfer Fixity

Since the Transfer endpoint takes advantage of the Download and Upload endpoints, fixity is checked using all methods already existing in those endpoints.

8.2 File Transfer Service (FTS) Metadata

PresQT keeps track of file history of resources being updated by PresQT by passing along an FTS Metadata file with each PresQT action. The file is titled PRESQT_FTS_METADATA.json. Every time PresQT takes action on a resource, the source details about the files moved are written to the metadata file.
Definition of PresQT FTS Metadata fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allKeywords</td>
<td>array</td>
<td>All Keywords added to this resource via PresQT.</td>
</tr>
<tr>
<td>actions</td>
<td>array</td>
<td>Array of PresQT actions that have taken place on the this project</td>
</tr>
<tr>
<td>id</td>
<td>string</td>
<td>ID of the PresQT action (uuid4). Created at the time metadata is written</td>
</tr>
<tr>
<td>actionDateTime</td>
<td>string</td>
<td>Date and time that the action took place</td>
</tr>
<tr>
<td>actionType</td>
<td>string</td>
<td>Type of action (Download, Upload, Transfer)</td>
</tr>
<tr>
<td>sourceTargetName</td>
<td>string</td>
<td>Name of the source target the action is taking place on</td>
</tr>
<tr>
<td>sourceUsername</td>
<td>string</td>
<td>Requesting user’s source target username</td>
</tr>
<tr>
<td>destinationTargetName</td>
<td>string</td>
<td>Name of the destination target the action is taking place on</td>
</tr>
<tr>
<td>destinationUsername</td>
<td>string</td>
<td>Requesting user’s destination target username</td>
</tr>
<tr>
<td>keywords</td>
<td>dict</td>
<td>Keyword enhancements that took place during this action</td>
</tr>
<tr>
<td>sourceKeywords-[]</td>
<td>array</td>
<td>The source keywords added during this action</td>
</tr>
<tr>
<td>sourceKeywords-[]</td>
<td>array</td>
<td>This includes keywords in the target keywords found in FTS metadata file</td>
</tr>
<tr>
<td>sourceKeywords-[]</td>
<td>array</td>
<td>The new keyword enhancements added to the target</td>
</tr>
<tr>
<td>ontologies</td>
<td>array</td>
<td>Ontologies connected to the enhanced keywords added.</td>
</tr>
<tr>
<td>enhancer*</td>
<td>string</td>
<td>The enhancement service used to enhance the keywords</td>
</tr>
<tr>
<td>files</td>
<td>array</td>
<td>Array of files that were involved in the PresQT action</td>
</tr>
<tr>
<td>sourcePath</td>
<td>string</td>
<td>Path of the file at the source target</td>
</tr>
<tr>
<td>sourceHashes</td>
<td>dict</td>
<td>Object that contains the file hashes at the source target</td>
</tr>
<tr>
<td>title</td>
<td>string</td>
<td>Title of the file at the source target</td>
</tr>
<tr>
<td>extra</td>
<td>dict</td>
<td>Object that contains all extra metadata we can retrieve from the source target</td>
</tr>
<tr>
<td>failedFixityInfo</td>
<td>array</td>
<td>Array containing dictionaries of info on files that failed fixity check</td>
</tr>
<tr>
<td>newGenerated-Hash**</td>
<td>string</td>
<td>PresQT generated hash of the file</td>
</tr>
<tr>
<td>algorithmUsed**</td>
<td>string</td>
<td>Hash Algorithm used for the newGeneratedHash</td>
</tr>
<tr>
<td>reasonFixity-Failed**</td>
<td>string</td>
<td>Reason fixity failed for the file</td>
</tr>
<tr>
<td>destinationPath</td>
<td>string</td>
<td>Path of the file at the destination target</td>
</tr>
<tr>
<td>destinationHashes</td>
<td>dict</td>
<td>Object that contains the file hashes at the destination target</td>
</tr>
</tbody>
</table>

Example of PresQT FTS Metadata generated by a transfer of a project from GitHub to OSF:

```json
{
    "allKeywords": ["cat", "dog", "feline", "doggo", "pupper"],
    "actions": [
        {
            "id": "bc5a48dc-d1f9-46bd-9137-48fe4843df77",
            "actionDateTime": "2019-11-12 15:45:45.309566+00:00",
            "actionType": "resource_transfer_in",
            "sourceTargetName": "github",
            "sourceUsername": "github_username",
            "destinationTargetName": "osf",
            "destinationUsername": "osf_username",
```
"keywords": {
  "sourceKeywordsAdded": ["cat", "dog"],
  "sourceKeywordsEnhanced": ["feline", "doggo", "pupper"],
  "ontologies": [
    {
      "keywords": [
        "doggo",
        "pupper"
      ],
      "ontology": "http://purl.obolibrary.org/obo/CHEBI_153377",
      "ontology_id": "CHEBI_153377",
      "categories": [
        "canine"
      ]
    },
    {
      "keywords": [
        "feline"
      ],
      "ontology": "http://purl.obolibrary.org/obo/CHEBI_153377",
      "ontology_id": "CHEBI_153377",
      "categories": [
        "felines"
      ]
    }
  ],
  "enhancer": "scigraph"
},
"files": {
  "created": [
    {
      "destinationPath": "NewProject/osfstorage/funnyfunnyimages/Screen_Shot.png",
      "destinationHashes": {
        "md5": "3505a89c3cbb82873a107ae41f3997c3"
      },
      "failedFixityInfo": [
        {
          "NewGeneratedHash": "3505a89c3cbb82873a107ae41f3997c3",
          "algorithmUsed": "md5",
          "reasonFixityFailed": "Either a Source Hash was not provided or the source hash algorithm is not supported."
        }
      ],
      "title": "Screen_Shot.png",
      "sourceHashes": {},
      "sourcePath": "/NewProject/funnyfunnyimages/Screen_Shot.png",
      "extra": {
        "commit_hash": "211ef8db83612802aee15a0e04badfe287bcb9",
        "size": 731202,
        "html_url": "https://github.com/presqt-test-user/NewProject/blob/master/funnyfunnyimages/Screen_Shot.png",
        "git_url": "https://api.github.com/repos/presqt-test-user/NewProject/git/blobs/211ef8db83612802aee15a0e04badfe287bcb9"
      }
    }
  ]
}
(continues on next page)
Now if we download from OSF the same project that was just transferred, then PresQT FTS Metadata would be:

```json
{
  "allKeywords": ["cat", "dog", "feline", "doggo", "pupper"],
  "actions": [
    {
      "id": "bc5a48dc-d1f9-46bd-9137-48fe4843df77",
      "actionDateTime": "2019-11-12 15:45:45.309566+00:00",
      "actionType": "resource_transfer_in",
      "sourceTargetName": "github",
      "sourceUsername": "github_username",
      "destinationTargetName": "osf",
      "destinationUsername": "osf_username",
      "keywords": {
        "sourceKeywordsAdded": ["cat", "dog"],
        "sourceKeywordsEnhanced": ["feline", "doggo"],
        "ontologies": [
          {
            "keywords": ["doggo", "pupper"],
            "ontology": "http://purl.obolibrary.org/obo/CHEBI_153377",
            "ontology_id": "CHEBI_153377",
            "categories": ["canine"
          ]
        ]
      }
    },
    {
      "keywords": ["feline"],
      "ontology": "http://purl.obolibrary.org/obo/CHEBI_153377",
      "ontology_id": "CHEBI_153377"
    }
  ]
}```
"categories": [
  "felines"
],
"enhancer": "scigraph",
"files": {
  "created": [
    "destinationPath": "NewProject/osfstorage/funnyfunnyimages/Screen_Shot.png",
    "destinationHashes": {
      "md5": "3505a89c3cbb82873a107ae41f3997c3"
    },
    "failedFixityInfo": [
      "NewGeneratedHash": "3505a89c3cbb82873a107ae41f3997c3",
      "algorithmUsed": "md5",
      "reasonFixityFailed": "Either a Source Hash was not provided or the source hash algorithm is not supported."
    ]
  ]
},
"title": "Screen_Shot.png",
"sourceHashes": {},
"sourcePath": "/NewProject/funnyfunnyimages/Screen_Shot",
"extra": {
  "commit_hash": "211ef8db83612802aeaa151a0e04badfe287bcb9",
  "size": 731202,
  "html_url": "https://github.com/presqt-test-user/NewProject/blob/master/funnyfunnyimages/Screen_Shot.png",
  "git_url": "https://api.github.com/repos/presqt-test-user/NewProject/git/blobs/211ef8db83612802aeaa151a0e04badfe287bcb9",
  "download_url": "https://raw.githubusercontent.com/presqt-test-user/NewProject/master/funnyfunnyimages/Screen_Shot.png",
  "type": "file",
  "_links": {
    "git": "https://api.github.com/repos/presqt-test-user/NewProject/git/blobs/211ef8db83612802aeaa151a0e04badfe287bcb9",
    "html": "https://github.com/presqt-test-user/NewProject/blob/master/funnyfunnyimages/Screen_Shot.png"
  }
},
"updated": [],
"ignored": []
}
"id": "bc5a48dc-d1f9-46bd-9137-48fe4843df77",
"actionDateTime": "2019-11-12 15:45:45.309566+00:00"}
8.2.1 Metadata Location When Downloading

The PresQT FTS Metadata file will be written to the highest level possible of the resource being downloaded.
8.2.2 Metadata Location When Uploading or Transferring

The PresQT FTS Metadata file will be written to the highest level possible of the destination project. Since this possible level may vary for any target, we leave it up to the target to handle this when they integrate with Upload.

8.2.3 Existing Metadata

If a valid PresQT FTS Metadata file is found at the top level of the resource being affected by the action then we will add a new action to this existing metadata file.

If an invalid PresQT FTS Metadata file is found at the top level of the resource being affected by the action then we will rename the invalid metadata file to INVALID_PRESQT_FTS_METADATA.json and then we will create a new valid metadata file with the current actions metadata.

8.3 Keyword Assignment

8.3.1 Keyword Enhancers

- SciGraph http://ec-scigraph.sdsc.edu:9000/scigraph/docs/

8.3.2 Keyword Difference Between Targets

Each target holds keywords in different attributes. Some may have keywords in multiple attributes. The following table outlines the keyword attributes for each target.

<table>
<thead>
<tr>
<th>Targets</th>
<th>Keyword Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSF</td>
<td>[Tags]</td>
</tr>
<tr>
<td>Github</td>
<td>[Topics]</td>
</tr>
<tr>
<td>Gitlab</td>
<td>[Tag List]</td>
</tr>
<tr>
<td>CurateND</td>
<td>[Subjects]</td>
</tr>
<tr>
<td>Zenodo</td>
<td>[Keywords]</td>
</tr>
<tr>
<td>FigShare</td>
<td>[Tags]</td>
</tr>
</tbody>
</table>

8.3.3 Keyword Assignment During Transfer

When transferring a resource you have the option of either manual or automatic keyword enhancement. Manual enhancement will only add source keywords and the keywords provided in the request body. Automatic will add all enhancements including any provided in the request body. These can be set by setting presqt-keyword-action in the headers to either manual or automatic.

Manual Keywords

If presqt-keyword-action is manual then PresQT will only add keywords found in the source target and keywords given in the body of the request. This means you need to get the possible enhancements before initiating a transfer.
Automatic Keywords

If `presqt-keyword-action` is automatic then PresQT will add keywords found in the source, keywords given in the request body, and any keyword enhancements found during the transfer process. The following steps occur during the transfer in this case:

1. Fetch all source keywords both in the target and in the FTS metadata file for the transferred resource.
2. Get enhancements with the given enhancer (Defaults to SciGraph for now).
3. Upload keyword enhancements to the Source Target and Destination Target.
4. Add the keyword enhancements to the FTS Metadata file that gets written to the Destination Target during the transfer.
5. Add the keyword enhancements to the FTS Metadata file that gets written to the Source Target during the transfer.

![Fig. 2: Image 2: Lifecycle of Keyword Enhancement during a transfer](image)

![Fig. 3: Image 3: Practical Example of Keyword Enhancement during a transfer](image)
**8.3.4 Keyword Assignment Service Endpoint**

Keyword Enhancement can be done without transferring.

1. Use the **Keyword Enhancement GET** endpoint to fetch the keywords from the resource.
2. Pass the keywords you want to enhance to the **Keyword Enhancement POST** endpoint.
3. Enhanced keywords will get uploaded to the target and a new action will get written to the FTS metadata file.

![Lifecycle of a Keyword Enhancement Service](image)

**Fig. 4: Lifecycle of a Keyword Enhancement Service**

---

**8.4 Preservation Quality**

IN PROGRESS
Fig. 5: Image 5: Practical Example of a Keywords Enhancement Service
A service is a unique target integration that is not accessible via the normal API endpoints. Typically, the point of a service is to take action on a resource (emulate, annotate, etc.) rather than simply storing it.

9.1 EaaS (Emulation-as-a-Service Infrastructure) Service

PresQT takes advantage of EaaS’s ability to interpret resources and suggest a relevant emulation environment. Our PresQT API calls use EaaS’s Proposal API to send resources to EaaS.

9.1.1 Step 1: Download the Resource

PresQT is able to use the existing download endpoint to fetch a Target’s resource to a PresQT server.

9.1.2 Step 2: Start a proposal task on an EaaS server

Then, using the ticket number created from the PresQT download task, a POST request can be made to PresQT to send EaaS a url where the downloaded resource can be fetched. During this POST request we write a one time use token to the downloaded resource’s process_info.json file. The URL we send to EaaS to fetch the PresQT resource has a query parameter with this token. This EaaS download endpoint is for EaaS use only.

9.1.3 Step 3: Get proposal status from EaaS

We then have a GET endpoint that makes a request to EaaS to find the progress of the Proposal Task. If the task is complete then we return the url for the suggested emulation environment. Otherwise, we return a 202 status and let the user know the proposal task is still in progress.

9.2 FAIR Evaluator Service

PresQT takes advantage of FAIRshare’s prebuilt maturity indicator tests. Our PresQT API calls use an approved collection of tests identified by the PI’s and community.

9.3 FAIRshake Assessment Service

PresQT takes advantage of FAIRshake’s manual assessment functionality to allow users to assess the FAIRness of their research projects.
Fig. 1: Image 1: Workflow of getting an EaaSI Emulation Environment of a given resource
10.1 Service Endpoints

10.1.1 Service Collection

GET /api_v1/services/
Retrieve details of all Services.

Example request:
```
GET /api_v1/services/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Example response:
```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "name": "eaasi",
    "readable_name": "EaaSI",
    "links": [
      {
        "name": "Detail",
        "link": "https://presqt-prod.crc.nd.edu/api_v1/services/eaasi/",
        "method": "GET"
      }
    ]
  }
]
```

Status Codes
- 200 OK – Services successfully retrieved

10.1.2 Service Details

GET /api_v1/services/(str: service_name)/
Retrieve details of a single Service.

Example request:
GET /api_v1/services/eaasi/ HTTP/1.1  
Host: presqt-prod.crc.nd.edu  
Accept: application/json

Example response:

HTTP/1.1 200 OK  
Content-Type: application/json

{
   "name": "eaasi",
   "readable_name": "EaaSI",
   "links": [
      {
         "name": "Proposals",
         "link": "https://presqt-prod.crc.nd.edu/api_v1/services/eaasi/proposals/",
         "method": "POST"
      }
   ]
}

Status Codes

- 200 OK – Service successfully retrieved
- 404 Not Found – Invalid Service name

10.2 Keyword Enhancement

10.2.1 Get Keyword Enhancements From A List Of Keywords

GET /api_v1/services/presqt/keyword_enhancement/

Take a list of keywords and run them through the keyword enhancement service. The returned payload will contain both the new keywords added and the final full list of keywords.

There are separate endpoints for keyword enhancements through Targets. See the API Endpoint documentation to learn more.

Example request:

POST /api_v1/services/presqt/keyword_enhancement/ HTTP/1.1  
Host: presqt-prod.crc.nd.edu  
Accept: application/json

Example body json:

{
   "keywords": ["cat", "water"]
}

Example response:

HTTP/1.1 202 Accepted  
Content-Type: application/json

(continues on next page)
```
{
    "keywords_added": [
        "feline",
        "aqua",
        "dihydrogen oxide",
        "DISORDERED SOLVENT",
        "EGG",
        "Electrostatic Gravity Gradiometer",
        "oxidane",
        "OXYGEN ATOM",
        "Wasser",
        "Water"
    ],
    "final_keywords": [
        "feline",
        "aqua",
        "dihydrogen oxide",
        "DISORDERED SOLVENT",
        "EGG",
        "eggs",
        "Electrostatic Gravity Gradiometer",
        "oxidane",
        "OXYGEN ATOM",
        "Wasser",
        "water",
        "Water"
    ]
}
```

**JSON Parameters**

- **keywords** *(array)* – An array of the keywords to upload

**Status Codes**

- **202 Accepted** – Keywords successfully uploaded

### 10.3 EaaSI Endpoints

#### 10.3.1 Submit EaaSI Proposal

**POST /api_v1/services/eaasi/proposals/**

Send a file from a PresQT server to start a proposal task on an EaaSI server.

**Example request:**

```
POST /api_v1/services/eaasi/proposals/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json

Example body json:
{
    "ticket_number":"39e56297-04cc-440a-b73e-9788b220f12b"
}
```
Example response:

```json
HTTP/1.1 200 OK
Content-Type: application/json
{
  "id": "19",
  "message": "Proposal task was submitted."
  "proposal_link": "https://presqt-prod.crc.nd.edu/api_v1/services/eaasi/1/
}
```

Status Codes

- **200 OK** – Proposal successfully started.
- **400 Bad Request** – ‘presqt-source-token’ missing in request headers
- **400 Bad Request** – A download does not exist for this user on the server.
- **404 Not Found** – Invalid ticket number
- **404 Not Found** – A resource_download does not exist for this user on the server.

### 10.3.2 Get EaaSI Proposal

**GET /api_v1/services/eaasi/proposals/(str: proposal_id)/**

Check on the state of the EaaSI Proposal Task on the EaaSI server.

**Example request:**

```plaintext
GET /api_v1/services/eaasi/proposals/12/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

**Example response if the proposal task is not finished:**

```json
HTTP/1.1 202 Accepted
Content-Type: application/json
{
  "message": "Proposal task is still in progress."
}
```

**Example response if the proposal task is finished successfully:**

```json
HTTP/1.1 200 OK
Content-Type: application/json
{
  "image_url": "https://eaasi-portal.emulation.cloud:443/blobstore/api/v1/blobs/imagebuilder-outputs/2ca330d6-23f7-4f0a-943a-e3984b29642c?access_token=default",
  "image_type": "cdrom",
  "environments": [],
  "suggested": {}
}
```

**Status Codes**

- **200 OK** – Proposal Task has finished successfully
• 202 Accepted – Proposal Task is being processed on the EaaSI server
• 404 Not Found – Invalid Proposal ID

10.3.3 EaaSI Download

GET /api_v1/services/eaasi/(str: ticket_number)/?eaasi_token=(str: eaasi_token)
EaaSI specific download endpoint that exposes a resource on a PresQT server to download.

Example request:

```
GET /api_v1/services/eaasi/download/39e56297-04cc-440a-b73e/?eaasi=E9luKQU9Ywe5j

HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/zip

Payload is ZIP file
```

Status Codes

• 200 OK – File successfully retrieved.
• 400 Bad Request – eaasi_token not found as query parameter.
• 401 Unauthorized – eaasi_token does not match the ‘eaasi_token’ for this server process.
• 404 Not Found – File unavailable.
• 404 Not Found – Invalid ticket number.
• 404 Not Found – A resource_download does not exist for this user on the server.

10.4 FAIRshare Endpoints

10.4.1 Get FAIRshare Tests

GET /api_v1/services/fairshare/evaluator/
Get a list of tests from FAIRshare that are currently supported by PresQT.

Example request:

```
GET /api_v1/services/fairshare/evaluator/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

(continues on next page)
[  
  {  
    "test_name": "FAIR Metrics Gen2- Unique Identifier ",
    "description": "Metric to test if the metadata resource has a unique identifier. This is done by comparing the GUID to the patterns (by regexp) of known GUID schemas such as URLs and DOIs. Known schema are registered in FAIRSharing (https://fairsharing.org/standards/?q=&selected_facets=type_exact:identifier%20schema)",
    "test_id": 1
  },  
  {  
    "test_name": "FAIR Metrics Gen2 - Identifier Persistence ",
    "description": "Metric to test if the unique identifier of the metadata resource is likely to be persistent. Known schema are registered in FAIRSharing (https://fairsharing.org/standards/?q=&selected_facets=type_exact:identifier%20schema). For URLs that don't follow a schema in FAIRSharing we test known URL persistence schemas (purl, oclc, fdlp, purlz, w3id, ark)",
    "test_id": 2
  }...
]

**Status Codes**

- **200 OK** – Tests returned successfully

10.4.2 POST FAIRshare Evaluator

**POST /api_v1/services/fairshare/evaluator/**

Submit a FAIRshare Evaluation request with a doi and list of test ids.

**Example request:**

```
POST /api_v1/services/fairshare/evaluator/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json

Example body json:
{  
  "resource_id": "10.17605/OSF.IO/EGGS12",
  "tests": [1, 2]
}
```

**Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[  
  {  
    "metric_link": "https://w3id.org/FAIR_Evaluator/metrics/1",
    "test_name": "FAIR Metrics Gen2- Unique Identifier ",
    "description": "Metric to test if the metadata resource has a unique identifier. This is done by comparing the GUID to the patterns (by regexp) of known GUID schemas such as URLs and DOIs. Known schema are registered in FAIRSharing (https://fairsharing.org/standards/?q=&selected_facets=type_exact:identifier%20schema)",
    "test_id": 1
  }
```
"successes": [
  "Found an identifier of type 'doi'",
],
"failures": [],
"warnings": []
],
{
  "metric_link": "https://w3id.org/FAIR_Evaluator/metrics/2",
  "test_name": "FAIR Metrics Gen2 - Identifier Persistence",
  "description": "Metric to test if the unique identifier of the metadata resource is likely to be persistent. Known schema are registered in FAIRSharing (https://fairsharing.org/standards/?q=&selected_facets=type_exact:identifier%20schema). For URLs that don't follow a schema in FAIRSharing we test known URL persistence schemas (purl, oclc, fdlp, purlz, w3id, ark).",
  "successes": [
    "The GUID of the metadata is a doi, which is known to be persistent."
  ],
  "failures": [],
  "warnings": []
}]

Status Codes

- 200 OK – Evaluation completed successfully.
- 400 Bad Request – ‘resource_id’ missing in the request body.
- 400 Bad Request – ‘tests’ missing in the request body.
- 400 Bad Request – ‘tests’ must be in list format.
- 400 Bad Request – At least one test is required. Options are: […]
- 400 Bad Request – ‘eggs’ not a valid test name. Options are: […]
- 503 Service Unavailable – FAIRshare returned a <status_code> error trying to process the request

10.5 FAIRshake Endpoints

10.5.1 Get FAIRshake Rubrics

GET /api_v1/services/fairshake/rubric/{str: rubric_id}/
Get a list of metrics from FAIRshake that are associated with the rubric id.

Example request:

```
GET /api_v1/services/fairshake/rubric/9/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json
```

Example response:
HTTP/1.1 200 OK
Content-Type: application/json

{
  "metrics": {
    "30": "The structure of the repository permits efficient discovery of data and metadata by end users.",
    "31": "The repository uses a standardized protocol to permit access by users.",
    "32": "The repository provides contact information for staff to enable users with questions or suggestions to interact with repository experts.",
    "33": "Tools that can be used to analyze each dataset are listed on the corresponding dataset pages.",
    "34": "The repository maintains licenses to manage data access and use.",
    "35": "The repository hosts data and metadata according to a set of defined criteria to ensure that the resources provided are consistent with the intent of the repository.",
    "36": "The repository provides documentation for each resource to permit its complete and accurate citation.",
    "37": "A description of the methods used to acquire the data is provided.",
    "38": "Version information is provided for each resource, where available.",
  },
  "answer_options": {
    "0.0": "no",
    "0.25": "nobut",
    "0.5": "maybe",
    "0.75": "yesbut",
    "1.0": "yes"
  }
}

Status Codes
- 200 OK – Rubric returned successfully
- 400 Bad Request – ‘egg’ is not a valid rubric id. Choices are: [‘7’, ‘8’, ‘9’]

10.5.2 POST FAIRshake Assessment

POST /api_v1/services/fairshake/rubric/{str: rubric_id}/
Submit a FAIRshake Assessment request for the given rubric.

Example request:
POST /api_v1/services/fairshake/rubric/9/ HTTP/1.1
Host: presqt-prod.crc.nd.edu
Accept: application/json

Example body json:
{
  "project_url": "https://github.com/ndlib/presqt",
  "project_title": "presqt",
  "rubric_answers": {
    "30": "0.0",
    (continues on next page)
Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "digital_object_id": 166055,
  "rubric_responses": [
    {
      "metric": "The structure of the repository permits efficient discovery of data and metadata by end users.",
      "score": "0.0",
      "score_explanation": "no"
    }
  ]
}
```

Status Codes

- **200 OK** – Assessment completed successfully.
- **400 Bad Request** – ‘eggs’ is not a valid rubric id. Options are: [‘7’, ‘8’, ‘9’]
- **400 Bad Request** – ‘project_url’ missing in POST body.
- **400 Bad Request** – ‘project_title’ missing in POST body.
- **400 Bad Request** – ‘rubric_answers’ missing in POST body.
- **400 Bad Request** – ‘rubric_answers’ must be an object with the metric id’s as the keys and answer values as the values.
- **400 Bad Request** – Missing response for metric ‘30’. Required metrics are: [‘30’, ‘31’, ‘32’]
- **400 Bad Request** – ‘egg’ is not a valid answer. Options are: [‘0.0’, ‘0.25’, ‘0.5’, ‘0.75’, ‘1.0’]
- **400 Bad Request** – ‘egg’ is not a valid metric. Required metrics are: [‘30’, ‘31’, ‘32’]
This page contains all relevant resources used during development

### 11.1 Links

- 
- 
- 
- 
- 
- 
- 

### 11.2 Example BagIts

#### 11.2.1 BagIt Zip files

Since the upload endpoint requires a BagIt file in zip format here are some pre-made zip files to test the upload endpoint.

#1 Valid BagIt For Top Level Container w/Folder
#2 Valid BagIt For Top Level Container w/File
#3 Valid BagIt For Existing Container w/Single File
#4 Valid BagIt For Existing Container w/Folders & Files
#5 Invalid BagIt - Bad Manifest
#6 Invalid BagIt - Missing File
#7 Invalid BagIt - Unknown File

#### 11.2.2 Example Workflow

The following are instructions on how the BagIt files above can be used to test the Upload endpoint:

1. Make a POST to https://presqt-prod.crc.nd.edu/api_v1/targets/osf/resources/ with BagIt #2 to see a new top level container created.
2. Get the id of the new container and make a POST to https://presqt-prod.crc.nd.edu/api_v1/targets/osf/resources/{resource_id}/ with BagIt #3 and with the 'presqt-file-duplicate-action' set to ‘ignore’ to see that the duplicate file is found and it’s contents are different but the file is updated.

3. Make the same request as 2 but set the header ‘presqt-file-duplicate-action’ to ‘update’ to see the file updated.

4. With the same container id make a POST request to https://presqt-prod.crc.nd.edu/api_v1/targets/osf/resources/{resource_id}/ with BagIt #4 to see new files and folders added to the top level container.

5. A POST request with BagIts 5-7 should return an error with nothing being uploaded.
Be sure to use the table of contents shown here to help navigate the instructions to go to the testing site.
12.1 Resources

We use the term **resources** for all content such as files, folders, projects, repos, items, etc. It’s a catch all term since different websites name their content differently.

12.2 BagIt Tool

BagIt is a hierarchical filesystem format for storing and transferring digital content. PresQT expects all files Uploaded to be zipped files in BagIt format. All downloads from PresQT come in BagIt format as well. PresQT has a tool that will take a zipped file and return it to you in BagIt format.

12.3 Getting Authorization Tokens From Partner Sites

An **Authorization Token** is a unique identifier for a user requesting access to a service.

You can for instructions on how to get authorization tokens for each target.

12.4 Test Files

Here are some pre-made ZIP files that are in BagIt format that can be downloaded for use with PresQT.

- presqt_Images.zip
- presqt_MediaFiles.zip
- presqt_TextFiles.zip
- presqt_MixedFileTypes.zip
12.5 Known Bugs And Issues

- None as of this writing

12.6 Testing Instructions

12.6.1 Login To Targets From PresQT Demo UI

1. Click on any Target icon under ‘Available Connections’ to pop open a login window.

2. Copy your Authorization Token for the target and press Connect

3. Resources associated with this token will appear on the left side.

4. You can log out of the target and use a different token by pressing the button next to the resources header.

5. To log into a different target simply repeat the process with a different target icon. Once logged in you can switch between targets without having to provide your key.

12.6.2 Navigate and Searching The Resource Collection

to go to the testing site.

1. After logging in you can navigate through your Resource Collection by clicking on the folders and files on the left.

2. Clicking on a resource shows you the Resource Details on the right.
Chapter 12. QA Testing
3. Searching for public resources can be accomplished by selecting a search type and then pressing the search icon. Public resources will be shown in the Resource Collection. 4. You can get back to your resources by pressing the refresh button.

12.6.3 Resource Details And Actions

1. Once you click on a resource you will get its details and buttons for each action available for this resource. If the button is disabled then that action isn’t available for that resource.

12.6.4 Resource Download

to go to the testing site.
1. To download a resource, first select the resource in the resource collection and then click the Download action button in the details section.

2. A modal will pop open providing you with transaction details. Click on the Download button to start the download.

3. Once the download is complete, the modal will provide you with details about how the download process went.

4. All downloads come in BagIt format. After the download is complete, unzip the file, and you will see BagIt specification files. The data you requested to download will reside in the data folder.

12.6.5 Resource Upload

to go to the testing site.

12.6.6 Upload As A New Project

1. To upload to the target as a new project click the Create New Project button above the resource collection.

2. A modal will pop open with an upload stepper. First select the file you’d like to upload. The file must be a zip file who’s contents are in valid BagIt format.

3. Next, the modal will display transaction details. Click Upload File to begin the upload process.

4. Once the upload is completed, the modal will provide you with details about how the upload process went.

5. You should also see the new uploaded resources appear in the resource collection.
12.6. Testing Instructions
12.6. Testing Instructions
12.6.7 Upload To An Existing Resource

1. To upload a resource, first select the resource in the resource collection and then click the Upload action button in the details section.

2. A modal will pop open with an upload stepper. First select the file you’d like to upload. The file must be a zip file who’s contents are in valid BagIt format.

3. Select how you want PresQT to handle any duplicate files it finds existing in the resource already. Ignore will simply ignore the duplicate. Update will update the existing file with the new uploaded file’s contents if they differ.

4. Next, the modal will display transaction details. Click Upload File to begin the upload process.

5. Once the upload is completed, the modal will provide you with details about how the upload process went.

6. You should also see the new uploaded resources appear in the resource collection.

12.6.8 Resource Transfer

to go to the testing site.
12.6. Testing Instructions
1. To transfer a resource to another target, first select the resource in the resource collection and then click the Transfer button in the details section.

2. A modal will pop open with a transfer stepper. First, select the target you want to transfer to and press the Next button.

3. Input your token for the target you selected and press the Next button.

4. Select the resource you want to transfer to. Don’t select any resource if you want to create a new project. Press Next once you have made your selection.

5. Select how you want PresQT to handle any duplicate files it finds existing in the resource already. Ignore will simply ignore the duplicate. Update will update the existing file with the new transferred file’s contents if they differ. Press the Next button once you’ve made your selection. If you are making a new project then just press Next.

6. Next, the modal will display transaction details. Click Transfer File to begin the transfer process.

7. Once the transfer is completed, the modal will provide you with details about how the transfer process went.
12.6. Testing Instructions
8. You should also see the new transferred resources appear in the modal’s resource collection on the right.

### 12.7 Verifying Fixity

**Fixity** means the assurance that a digital file has remained unchanged. We determine file fixity at every step along PresQT actions. More details about how PresQT handles fixity can be found [here](#).

#### 12.7.1 Download

All downloads come with a file with detailed fixity information named `fixity_info.json`. This file has an entry for every file involved in the download including each file’s checksum hash at the Source Target and the hash calculated on the PresQT servers before sent to the browser for download. To verify fixity remains, the user must calculate the files’ hashes on their local machine and compare it to the hashes provided.
12.7.2 Upload

Fixity during upload can be determined by inspecting the `PRESQT_FTS_METADATA.json` file included with every upload. The attribute `failedFixityInfo` in this file will contain the details if the file being uploaded has failed fixity.

12.7.3 Transfer

Fixity during Transfer can be determined the same as Upload by inspecting the `PRESQT_FTS_METADATA.json` file in the destination target.

12.8 Verifying Keyword Enhancement

See Here for Keyword Enhancement details.

12.8.1 Keyword Enhancement As A Service

Keyword Enhancement as a service will write a new entry to the `PRESQT_FTS_METADATA.json` file in the target. The action entry for keyword enhancement will say exactly which keywords were added during this enhancement.

12.8.2 Keyword Enhancement During Transfer

Keyword Enhancement during a transfer will work similarly to Keyword Enhancement As A Service. The difference is, for the destination target, the details of keyword enhancement will be located in the transfer action entry instead of there being a new action entry for keyword enhancement.
12.9 Services

to go to the testing site.

12.9.1 Send a Proposal to EaaSI

1. To send a resource to EaaSI, first select the resource in the resource collection and then click the Services action button in the details section. A drop down menu will appear from where you can select EaaSI.

2. A modal will pop open with an EaaSI stepper. First read the proposal and ensure the information is correct. Once you have verified that this is what you’d like to do, press the Send button.

3. A spinner will keep you informed of where in the process the request is, whether that be on the PresQT server or
on EaaSI’s.

4. Once the upload is completed, the modal will provide you with details about how the process went. There will also be a link for you to download the EaaSI created image.

5. You can now open the image and run it however you please. Note: At this point in time, EaaSI’s server is only returning cd-rom images for us during testing. The environments will be changed to accurately take into account the files contained within the project as development continues.

12.9.2 FAIRshare Evaluator Service

1. To initiate a FAIRshare evaluation, first select the resource in the resource collection and then click the Services action button in the details section. A drop down menu will appear from where you can select FAIRshare.

2. A modal will pop open with a FAIRshare Evaluator Service stepper. First read the information and ensure the information is correct. Once you have verified that this is what you’d like to do, select the tests you would like to run.

3. Once you have selected the tests you’d like to run, you can choose to opt in for email notifications. When you are ready to run the tests, press the Evaluate button.

   4. A spinner will let you know that FAIRshare is processing the request. This may take awhile.

   5. Once the process is complete, the results will be displayed in a drop down format to be reviewed.
## Chapter 12. QA Testing

### FARshare Evaluator Service

**Submit FARshare Evaluator Request**

Submit a FARshare Evaluator request for PresQT Data and Software Preservation Quality Tool Project with doi 10.17605/OSF.IO/DJQJ7. They will return the results of the tests you select to run below.

### FARshare tests supported by PresQT

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FAIR Metrics Gen2 - Unique Identifier</strong></td>
<td>Metric to test if the metadata resource has a unique identifier. This is done by comparing the GUID to the pattern (by regex) of known GUID schemas such as UMI and DOI. Known schemas are registered in FAIRsharing (<a href="https://fairsharing.org/standards/">https://fairsharing.org/standards/</a>). Only selected, facets-type, exact/identifier%20/schema).</td>
</tr>
<tr>
<td><strong>FAIR Metrics Gen2 - Identifier Persistence</strong></td>
<td>Metric to test if the unique identifier of the metadata resource is likely to be persistent. Known schemas are registered in FAIRsharing (<a href="https://fairsharing.org/standards/">https://fairsharing.org/standards/</a>). For URIs that don’t follow a schema in FAIRsharing we test known URI persistence schemas (pub, oai, rdfs, pid, w3id, ar).</td>
</tr>
<tr>
<td><strong>FAIR Metrics Gen2 - Structured Metadata</strong></td>
<td>Tests whether a machine is able to find structured metadata. This could be for example RDFa, embedded json, jsonld, or content-negotiated structured metadata such as RDF-Turtle.</td>
</tr>
<tr>
<td><strong>FAIR Metrics Gen2 - Grounded Metadata</strong></td>
<td>Tests whether a machine is able to find ‘grounded’ metadata. i.e. metadata terms that are in a resolvable namespace where resolution leads to a definition of the meaning of the term. Examples include JSON-LD, embedded schemas, or any form of RDF. This text currently excludes XML, even when terms are namespaceed. Future versions of this text may be more flexible.</td>
</tr>
<tr>
<td><strong>FAIR Metrics Gen2 - Data Identifier Explicitly in Metadata</strong></td>
<td>Metric to test if the metadata contains the unique identifier to the data. This is done by searching for a variety of properties, including doi/primary_hyphen, schema/mimetype, schema/distribution, so in a list, and so on. For data, schema/code/repository is used for software releases.</td>
</tr>
</tbody>
</table>

**FAIR Metrics Gen2 - Metadata Persistence**

Metric to test if the metadata contains a persistence policy, explicitly identified by a persistencePolicy key (as hashed data) or a http://www.w3.org/2000/10/swapi/persistencePolicy predicate in Linked Data.

**FAIR Metrics Gen2 - Data Knowledge Representation Language (strong)**

Maturity Indicator to test if the data uses a formal language broadly applicable for knowledge representation. This particular test takes a broad view of what defines a ‘knowledge representation language’, in this evaluation, a knowledge representation language is interpreted as one in which terms are semantically-grounded in ontologies. Any form of ontologically-grounded linked data will pass this test.

**FAIR Metrics Gen2 - Metadata uses FAIR vocabularies (strong)**

Maturity Indicator to test if the linked data metadata uses terms that resolve to linked (FAIR) data.

**FAIR Metrics Gen2 - Metadata Includes License**

Maturity Indicator to test if the metadata contains an explicit pointer to the license. This ‘weak’ test will use a case-insensitive regular expression, and scan both key-value style metadata, as well as linked data metadata. Tests: xml, dvid, dcremis, co, data.gov.au, and Schema license predicates in linked data, and validates the value of those properties.

**DESELECT ALL**

You can input your email below and we will notify you once this evaluation is complete. Inputting your email address is not mandatory and we will not store this information on the server once the process has finished.

Email Address: @nd.edu

**EVALUATE**

**Evaluation Results**
FAIRshare Evaluator Service

Submit FAIRshare Evaluator Request

Evaluation Results

Evaluation task is being processed on the FAIRshare server, this may take several minutes...

Evaluation Results

- FAIR Metrics Gen2 - Unique Identifier
  - Found an identifier of type ‘doi’
- FAIR Metrics Gen2 - Identifier Persistence
- FAIR Metrics Gen2 - Structured Metadata
- FAIR Metrics Gen2 - Grounded Metadata
- FAIR Metrics Gen2 - Data Identifier Explicitly in Metadata
- FAIR Metrics Gen2 - Metadata-Identifier Explicitly in Metadata
- FAIR Metrics Gen2 - Searchable in major search engine
- FAIR Metrics Gen2 - Uses open free protocol for metadata retrieval
- FAIR Metrics Gen2 - Metadata Persistence
- FAIR Metrics Gen2 - Data Knowledge Representation Language (strong)
13.1 Whole Tale Integration Proposal

Whole Tale (http://www.wholetale.org) is a platform for the creation, publication, and re-execution of reproducible computational artifacts. Researchers can create new tales that contain the code, data, workflow, and information about the computational environment required to reproduce their analysis. Tales have basic metadata including authors, title, keywords, description, and related identifiers (cited derived from). Tales can be published to archival repositories including DataONE network members and Zenodo.

Whole Tale is an integration partner for the PresQT project. As part of the integration testing process, we explored two different use cases:

- Publish Tales from the WT platform to Zenodo and CurateND using the PresQT APIs
- Import an OSF project into WT using the PresQT APIs

BagIt Serialization

Tales are exported as BDBag-compatible bags with the following structure:

```
Se696df5f1c291f11ae9e1a8/
  README.md  <-- Top-level readme (Tag file)
  bagit.txt
  bag-info.txt
  fetch.txt  <-- Fetch file
  manifest-md5.txt
  manifest-sha256.txt
  run-local.sh  <-- Script to run Tale locally (Tag file)
  tagmanifest-md5.txt
  tagmanifest-sha256.txt
  data/
    workspace/  <-- Tale workspace (user code, data, etc)
      environment.yml
      index.ipynb
    LICENSE  <-- Tale license (Tag file)
    metadata/
      environment.json
      manifest.json
```

The Tale bag is not currently compatible with PresQT, since PresQT currently requires a single top-level folder in the payload representing the project. The folder name is used as the dataset title when publishing to a target. Since the Tale bag has a separate structure, one option is to double-bag. In the following example, the above Tale is zipped and bagged:
Publishing to Zenodo

The following figures illustrate the Tale as published to Zenodo using the WT internal integration and using PresQT.

The first image illustrates a tale published to Zenodo using the WT internal integration. The title, author, and description are all provided by the user during tale creation. The WT platform adds a note with a custom link allowing the user to import and re-execute the tale in the WT system. WT also supports related identifiers, license, and keywork metadata. The tale is published as a zipped bag.

The second image illustrates a tale published to Zenodo using the PresQT integration. The user is directed to the draft dataset creation form where they are required to manually enter relevant metadata.

Metadata Support

To support our current Zenodo integration using the PresQT system would require the ability to specify additional metadata during dataset creation. We use the following fields:

- Title
- Creator/Authors (first name, last name, ORCID)
- Publication date
- Description
- Subject/keywords
- Rights/license
- Related identifiers (Cites, Derived From)
- References
- Notes

The notes field is important as it provides a way for us to embed a link in the record that allows users to easily re-import and run the published tale.

For more information, see https://github.com/whole-tale/serialization-format/

13.2 Whole Tale Integration Implementation

An ‘extra_metadata’ field has been added to the PRESQT_FTS_METADATA.json. To get these extra metadata fields to the new resource being created, the uploaded resources must have a PRESQT_FTS_METADATA.json file at the highest level.

The following is an example:
Fig. 1: Tale published to Zenodo via WT

13.2. Whole Tale Integration Implementation
Fig. 2: Tale published to Zenodo via PresQT
• genindex
• modindex
• search
<table>
<thead>
<tr>
<th>Method</th>
<th>Path</th>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/api_v1/job_status/download.json/</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/job_status/download.zip/</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/job_status/transfer/</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/job_status/upload/</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/services/</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/services/(str: service_name)/</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/services/eaasi/(str: ticket_number)/?eaasi_token=(str: eaasi_token)</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/services/eaasi/proposals/(str: proposal_id)/</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/services/fairshake/rubric/{str: rubric_id}/</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/services/fairshare/evaluator/</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/services/presqt/keyword_enhancement/</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/targets/</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/targets/(str: target_name)/</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/targets/(str: target_name)/resources/</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/targets/(str: target_name)/resources/(str: resource_id).json/</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/targets/(str: target_name)/resources/(str: resource_id).zip/</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/api_v1/targets/(str: target_name)/resources/(str: resource_id)/keywords/</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/api_v1/services/eaasi/proposals/</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/api_v1/services/fairshake/rubric/{str: rubric_id}/</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/api_v1/services/fairshare/evaluator/</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/api_v1/targets/(str: target_name)/resources/(str: resource_id)/resources/</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/api_v1/targets/(str: target_name)/resources/(str: resource_id)/keywords/</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>PATCH</td>
<td>/api_v1/job_status/transfer/</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>PATCH</td>
<td>/api_v1/job_status/upload/</td>
<td>54</td>
<td></td>
</tr>
</tbody>
</table>