Slicer Package Manager Documentation

Release 0.1.0

Pierre Assemat

May 11, 2023

Contents

The **Slicer Package Manager** includes a REST API service and CLI built on Girder for downloading, uploading and organizing application and extension packages for both 3D Slicer and 3D Slicer-based applications.

CHAPTER 1

Overview

The **Slicer Package Manager** includes a REST API service and CLI built on Girder for downloading, uploading and organizing application and extension packages for both 3D Slicer and 3D Slicer-based applications.

This software is licensed under the terms of the Apache Licence Version 2.0 and the source code is available at https://github.com/girder/slicer_package_manager.

In a nutshell:

- *Data model* specific to this project is implemented by organizing data using standard Girder constructs (collection, folder and item) and by associating metadata.
- By default, a top-level collection named `Applications` is created with a `packages` folder organizing the different application.
- Each application folder contain a `draft` folder where unreleased packages are uploaded and one or multiple release folders (e.g 1.0, 2.0, . . .).
- Each release folder contain application packages (installers for the different platforms), and an `extensions` folder containing a flat list of extension packages.
- Each extension packages is associated with metadata like application revision, extension revision, operating system and architecture. . .

The diagram below represents the organization:

```
Applications
    |--- packages
    |         |----- Slicer
    |         |          |----- 1.0
    |         |          |         |---- Slicer-linux.tar.gz
    |         |          |         |---- Slicer-macos.dmg
    |         |          |         |---- Slicer-win.exe
    |         |          |         |---- extensions
    |         |          |         |          |---- Extension1
    |         |          |         |          |---- Extension2
    |         |          |         |          |---- Extension3
    |         |          |         |          |---- Extension4
```

(continues on next page)

(continued from previous page)

```
  .        .         .        .         .
  .        .         .
  |        |         |----- 2.0
  .        .         .        |
  .        .         .
  |        |         |----- draft
  |        |         |        |--- r100
  |        |         |        |      |---- Slicer-linux.tar.gz
  |        |         |        |      |---- Slicer-macos.dmg
  |        |         |        |      |---- Slicer-win.exe
  |        |         |        |      |---- extensions
  |        |         |        |      |          |---- Extension1
  .        .         .        .      .          .
  .        .         .        .
  |        |         |        |--- r101
  .        .         .        .      |
  .        .
  |        |
  |        |------SlicerCustom
```

where

- `Slicer` and `SlicerCustom` are Girder items representing **Application**
- `1.0` and `2.0` are Girder folders representing **Release**
- `Slicer-linux.tar.gz` and `Slicer-macos.dmg` are Girder items representing **Package** (application package)
- `Extension1` and `Extension2` are Girder items representing **Extension** (extension package)

CHAPTER 2

Concepts

- **Application**:

  Applications are simple *Girder Folder*. They represent top-level folders which contain all the different releases, application and extensions packages of your application. By default when you create a new application, it will automatically be created within the default Girder collection 'Applications' (see Girder concept to learn more about Girder collections and Folder).

  Applications contain metadata that organize the application and extension packages following a same name:

  – `applicationPackageNameTemplate` (set as `{baseName}_{os}_{arch}_{revision}` by default).

  – `extensionPackageNameTemplate` (set as `{app_revision}_{baseName}_{arch}_{os}_{revision` by default).

  These template names correspond to the given name of all the uploaded application or extensions packages. Which means that all the packages (application or extension) will have a name that follow these templates depending on their given metadata during the upload.

  These templates can be changed at anytime using the Girder UI on the application view.

- **Release**:

  Release are also simple *Girder Folder*. They are part of an application and correspond to a specific revision of this application. They meant to contain application or extension packages which correspond to this specific application revision. It's why each release has the application `revision` as metadata.

- **Draft**:

  Draft is a simple *Girder Folder* which contain a flat list of **Release** named as the corresponding application revision by default. When a new application is created, the *draft* folder is also created. This *draft* folder is used as default release when the upload of an application or an extension package occurs and which its own *application revision* doesn't correspond to any release of the application (by checking the correspondence between the `revision` metadata stored on the release and the `app_revision` stored on the packages).

However the *draft* folder does not contain any metadata. Only the release that's contained into it got a `revision` metadata (corresponding to the application revision they are made for).

- **Package**:

  Package (application package) are *Girder Item* which contains only one binary file (the real application package). They are part of an application, and can only be found in a release folder. They are named following the `applicationPackageNameTemplate` set on the application they are made for.

  Each application package contain a bunch of metadata that give us information on which environment the package is made for like : Operating System: `os`, architecture: `arch`, application revision: `revision`, repository url… (see the list of parameters of Package on the server API to have an exhausted list of all the metadata).

  When uploading an application package, some of these metadata are required, the `revision` metadata is used to determine in which release to upload the application package. The release which have the same `revision` metadata will see the application package uploaded into it. If any release within the application has a matching revision, the application package will be uploaded into the corresponding *draft* release (by default). By searching for an existing draft release (with the matching revision) or if it doesn't already exist, by creating a new one.

  The package file (binary file) during the upload will be kept as it within Girder. So when the extension will be downloaded, the downloaded file will keep the same extension (.bin, .zip, …). For instance, if the uploaded package is named 'pkg.tar.gz', then each time this application package will be downloaded, the downloaded file will keep the same '.tar.gz' extension.

- **Extension**:

  As an application package, an Extension package is also a *Girder Item*, and has the same behavior than an application package. It contain a single binary file. The name of all uploaded extension follow the `extensionPackageNameTemplate` metadata stored in the **Application**.

  See the list of parameters of Extension on the server API to have an exhausted list of all the metadata.

CHAPTER 3

Commands shell (CLI)

3.1 Overview

The command `slicer_package_manager_client` allows to to interact with a Slicer Package Manager server.

There are 5 different subcommands that can be used to manage data:

- *app* command to create, list and delete applications.
- *release* command to create, list and delete releases.
- *draft* command to list and delete draft releases.
- *package* command to upload, download or just list application packages.
- *extension* command to upload, download or just list extensions packages.

Warning: To run command requiring higher privileges, you will have to *authenticate*.

3.2 Installation

Install with:

```
$ pip install slicer-package-manager-client
```

or:

```
$ git clone https://github.com/girder/slicer_package_manager.git
$ cd slicer_package_manager/python_client
$ pip install -e .
```

for development.

3.3 Configuration

3.3.1 Authentication

There are few solutions to authenticate on your Girder instance when using the client:

- Using your login and your password:

  ```
  $ slicer_package_manager_client --username admin --password adminadmin
  ```

- Generating an API-KEY see the documentation for more details:

  ```
  $ slicer_package_manager_client --api-key EKTb15LjqD4Q7jJuAVPuUSuW8N7s3dmuAekpRGLD
  ```

or by using the `GIRDER_API_KEY` environment variable:

```
$ export GIRDER_API_KEY=EKTb15LjqD4Q7jJuAVPuUSuW8N7s3dmuAekpRGLD
```

Warning: *The API-KEY is given as an example, follow the documentation on* api-key *to create your own.*

Note: If you want to use the client to an external *Slicer package manager* instance, you will need to provide the API url by adding the option:

```
--api-url http://192.168.100.110/api/v1
```

(The IP is given as an example)

Then you can start using the API that allow you to easily create applications, manage releases, upload and download packages, see *Commands shell (CLI)* documentation for more details.

3.3.2 Bash completion

To use the **Bash completion** feature you just have to run the following command each time you use a new terminal:

```
$ eval "$(_SLICER_PACKAGE_MANAGER_CLIENT_COMPLETE=source slicer_package_manager_
↪client)"
```

Or you can add it on your `.bashrc` file to always have this feature available.

3.3.3 Custom application collection

In each command, the optional parameter `coll_id` allow to use the Slicer Package Manager Client within an existing collection and not in the default *Applications* collection.

When this is the case, to avoid repeating this parameter in each command it's also possible to set an environment variable named `COLLECTION_ID`.

3.4 Subcommands

3.4.1 Application

Use `slicer_package_manager_client app` to create, list and delete applications.

Create & Initialized a new application

You can either choose an existing collection by providing `coll_id` or create a specific one by providing `coll_name`. If none of this optional parameters are provided, the default collection *Application* will be got or created if it doesn't exist yet. This function will also create a top level folder named *packages* organizing the different application in the collection.

```
slicer_package_manager_client app create NAME [OPTIONS]
```

Arguments:

- `NAME` - The name of the new application

Options:

- `--desc` - The description of the new application
- `--coll_id` - ID of an existing collection
- `--coll_name` - The name of the new collection
- `--coll_desc` - The description of the new collection
- `--public` - Whether the collection should be publicly visible

List all the application within a collection

By providing `coll_id`, you are able to list all the applications from a specific collection. By default it will list the applications within the collection *Applications*.

```
slicer_package_manager_client app list
```

Delete an application

```
slicer_package_manager_client app delete NAME
```

Arguments:

- `NAME` - The name of the application which will be deleted
- `--coll_id` - ID of an existing collection

3.4.2 Release

Use `slicer_package_manager_client release` to create, list and delete releases.

Create a new release

```
slicer_package_manager_client release create APP_NAME NAME REVISION [OPTIONS]
```

Arguments:

- `APP_NAME` - The name of the application
- `NAME` - The name of the new release
- `REVISION` - The revision of the application corresponding to this release

Options:

- `--coll_id` - ID of an existing collection
- `--desc` - The description of the new application

List all the release from an application

```
slicer_package_manager_client release list APP_NAME
```

Arguments:

- `APP_NAME` - The name of the application

Options:

- `--coll_id` - ID of an existing collection

Delete a release

```
slicer_package_manager_client release delete APP_NAME NAME
```

Arguments:

- `APP_NAME` - The name of the application
- `NAME` - The name of the release which will be deleted

Options:

- `--coll_id` - ID of an existing collection

3.4.3 Draft

Use `slicer_package_manager_client draft` to list and delete draft releases.

List all the draft release within an application

Provide `revision` will list only one draft release corresponding to the revision store as metadata. The `--offset` option allow to list only the older draft release.

```
slicer_package_manager_client draft list APP_NAME [OPTIONS]
```

Arguments:

- `APP_NAME` - The name of the application

Options:

- `--revision` - The revision of a draft release
- `--offset` - The offset to list only the older draft release
- `--coll_id` - ID of an existing collection

Delete a specific draft release

```
slicer_package_manager_client draft delete APP_NAME REVISION [OPTIONS]
```

Arguments:

- `APP_NAME` - The name of the application
- `REVISION` - The revision of the draft release

Options:

- `--coll_id` - ID of an existing collection

3.4.4 Package

Use `slicer_package_manager_client package` to upload, download or just list application packages.

Upload a new application package

Give the `FILE_PATH` argument to be able to upload an application package. The application package will automatically be added to the release which has the same revision than the `--revision` value. If any release correspond to the given revision, the application package will be uploaded in the *draft* release, by default.

The final name of the application package will depend of the `applicationPackageNameTemplate` set as metadata on the application folder. The default name is `{baseName}_{arch}_{os}_{revision}`. It can be change at any time on the application setting page.

The `--pre_release` option is used to specify if the uploaded package is ready for distribution or if it needs extra steps before that. In some cases, the package needs to be signed and then re-uploaded on the server.

```
slicer_package_manager_client package upload APP_NAME FILE_PATH [OPTIONS]
```

Arguments:

- `APP_NAME` - The name of the application
- `FILE_PATH` - The path to the application package file to upload

Options:

- `--os` - The target operating system of the package
- `--arch` - Architecture that is supported by the application package
- `--name` - The basename of the new application package
- `--repo_type` - The repository type of the application package
- `--repo_url` - The repository URL of the application package
- `--revision` - The revision of the application package

- `--coll_id` - ID of an existing collection
- `--pre_release` - Boolean to specify if the package is ready to be distributed
- `--desc` - The description of the new application

List application packages

Use options to filter the listed application packages. By default, the command will list all the application packages from the 'draft' release. It is possible to use the `--release` option to list the application package from a particular release.

```
slicer_package_manager_client package list APP_NAME [OPTIONS]
```

Arguments:

- `APP_NAME` - The name of the application

Options:

- `--os` - The target operating system of the package
- `--arch` - Architecture that is supported by the application package
- `--revision` - The revision of the application
- `--release` - The release within list all the application package
- `--name` - Basename of an application package
- `--limit` - Limit on the number of listed application package
- `--coll_id` - ID of an existing collection

Download an application package

By default the package will be store in the current folder

```
slicer_package_manager_client package download APP_NAME ID_OR_NAME [OPTIONS]
```

Arguments:

- `APP_NAME` - The name of the application
- `ID_OR_NAME` - The ID or name of the application package which will be downloaded

Options:

- `--dir_path` - Path where will be save the application package after the download
- `--coll_id` - ID of an existing collection

Delete an application package

Provide either the ID or the name of the application package to delete it.

```
slicer_package_manager_client package delete APP_NAME ID_OR_NAME
```

Arguments:

- `APP_NAME` - The name of the application

- `ID_OR_NAME` - The ID or name of the application package which will be deleted

Options:

- `--coll_id` - ID of an existing collection

3.4.5 Extension

Use `slicer_package_manager_client extension` to upload, download or just list extension packages.

Upload a new extension

Give the `FILE_PATH` argument to be able to upload an extension. The extension will then automatically be added to the release which has the same revision than the `--app_revision` value. By default, if any release corresponds to the given revision, the extension will be uploaded in the *draft* folder within the 'draft' release which has the given revision as metadata, or create it if it doesn't exist yet.

The final name of the extension will depend of the `extensionPackageNameTemplate` set as metadata on the application folder. The default name is `{app_revision}_{baseName}_{os}_{arch}_{revision}`. It can be change at any time on the application setting page.

```
slicer_package_manager_client extension upload APP_NAME FILE_PATH [OPTIONS]
```

Arguments:

- `APP_NAME` - The name of the application
- `FILE_PATH` - The path to the extension file to upload

Options:

- `--os` - The target operating system of the package
- `--arch` - Architecture that is supported by the extension
- `--name` - The basename of the new extension
- `--repo_type` - The repository type of the extension
- `--repo_url` - The repository URL of the extension
- `--revision` - The revision of the extension
- `--app_revision` - The revision of the application corresponding to this release
- `--coll_id` - ID of an existing collection
- `--desc` - The description of the new application

List extensions

Use options to filter the listed extensions. By default, the command will list all the extension from the 'draft' release. It is possible to use the `--release` option to list the extension from a particular release. Or use the flag `--all` to list all the extension present in the application. It is also possible to get only one extension by providing the `--fullname` option of an extension.

```
slicer_package_manager_client extension list APP_NAME [OPTIONS]
```

Arguments:

- `APP_NAME` - The name of the application

Options:

- `--os` - The target operating system of the package
- `--arch` - Architecture that is supported by the extension
- `--app_revision` - The revision of the application
- `--release` - The release within list all the extension
- `--limit` - Limit on the number of listed extension
- `--all` - Flag to list all the extension from all the release
- `--fullname` - Fullname of an extension
- `--coll_id` - ID of an existing collection

Download an extension

```
slicer_package_manager_client extension download APP_NAME ID_OR_NAME [OPTIONS]
```

Arguments:

- `APP_NAME` - The name of the application
- `ID_OR_NAME` - The ID or name of the extension which will be downloaded

Options:

- `--dir_path` - Path where will be save the extension after the download
- `--coll_id` - ID of an existing collection

Delete an extension

Provide either the ID or the name of the extension to delete it.

```
slicer_package_manager_client extension delete APP_NAME ID_OR_NAME
```

Arguments:

- `APP_NAME` - The name of the application
- `ID_OR_NAME` - The ID or name of the extension which will be deleted

Options:

- `--coll_id` - ID of an existing collection

CHAPTER 4

FAQ

Frequently Asked Questions

4.1 What is Girder?

Girder is a free and open source web-based **data management platform** developed by Kitware. What does that mean? Girder is both a standalone application and a platform for building new web services. To know more about Girder let's take a look at the documentation.

4.2 What is a Slicer package?

A slicer package is just an installer package for a specific release of Slicer. There is a specific Slicer package for each different platform (Windows, MACOSX, Linux).

4.3 What is a Slicer Extension?

An extension could be seen as a delivery package bundling together one or more Slicer modules. After installing an extension, the associated modules will be presented to the user as built-in ones.

To know more about Slicer extension, see the Extensions Manager documentation.

4.4 Does the server collect download statistics ?

Each time an extension is downloaded (using the Client or the UI), a metadata is incremented on the release folder. This allow to referenced all downloaded extension even after their deletion.

The download count is stored in the metadata following this rule:

```
$ {
    'downloadExtensions': {
        baseName: {
            os: {
                arch: downloadCount
            }
        }
    }
```

4.5 How to interface with the Slicer Package Manager server ?

There are 3 different ways to use the Slicer Package Manager server:

- By using the *Commands shell (CLI)*:

    This is the more easy way to use the basic feature of the Slicer Package Manager. These commands allow you to easily create, list, or delete applications and releases, and also list, upload, download or delete application or extension packages.

- By using the *Python Client API* within Python script:

    Using the Python Client API allow you to write scripts for create application, new release and automatically upload or download application or extensions packages.

- By using the User Interface:

    The default girder user interface allows to browse through the *Application*, *Release* and *Draft* Girder folders and their associated *Package* and *Extension* Girder items.

4.6 How to create a new release with existing uploaded packages?

Follow this few steps to be able to update a draft release into a stable release:

- Open the Girder UI, go under your application folder (Slicer here). By default it should be inside the `Applications` collection, that you can find under the `Collections` item in the main menu.

```
Applications
       |--- packages
       |        |----- Slicer
```

- Look for the specific application revision folder under the `draft` folder within the application. All the packages which are contained in this folder will be part of the futur new stable release.
- Select all the element contained in this folder by using the `Pick all checked resources for Copy or Move` action
- Go to the new release folder, that you can create both by using the CLI or the Girder UI. In the case of the Girder UI you will need to give a specific metadata on the folder: `revision: <revision-of-the-application>` corresponding the this release.
- Once you created the new release folder, enter inside it, then use the `Copy picked resources here`
- You will just need to delete the draft sub-folder used to make the new stable release

4.7 How to clean up the Draft release folder?

Since a large number of draft packages may be uploaded on a regular basis. Older draft packages may be removed applying this process:

The command `slicer_package_manager_client draft list <APP_NAME> --offset <N>` allows to list the oldest draft subfolders related to old application revision. Using this command, you will be able to get a list of `revision` and then use the command `slicer_package_manager_client draft delete <APP_NAME> <REVISION>` in a loop to delete them all.

Note: The draft packages associated with the https://slicer-packages.kitware.com instance are automatically cleared using the clean-nightly-slicer-package-manager.sh script.

CHAPTER 5

Server Installation

The section below describes a convenient way to setup a server for evaluation purpose.

For production deployment, read more details in the `Administrator Documentation` section at https://girder.readthedocs.io.

5.1 Run via Docker

First, install Docker and Docker compose follow the instruction on the official website. The community edition (CE) is sufficient for using this plugin. See https://docs.docker.com/install/ and https://docs.docker.com/compose/install/.

Then, assuming the sources are available in `slicer_package_manager` folder, you may run the server running the following commands:

```
$ cd slicer_package_manager
$ docker-compose up -d
```

Note: The `-d` option is running the container in daemon mode. Remove it to display the logs on the running containers.

To rebuild the container after changing the source code use the `--build` option when you run the command.

Warning: *Run the containers can take few moments, the application will not be ready instantly.*

The Girder application should then be running at http://localhost:8080/ and be already setup:

- Creation of an **Admin User** (username: *admin*, password: *adminadmin*)
- Creation of a local **Assetstore** (in *~/slicer_package_manager/assetstore*), let's read the Filesystem documentation for more detail about it

Note: You will have the possibility to create more users and/or change the password of the **Admin User** via the Girder UI.

CHAPTER 6

Developer Guide

6.1 Overview

Since **Slicer Package Manager** is part of Girder plugins, it's also split in 2 different parts:

- Back-end/server side (a CherryPy-based Python module)
- Front-end/client side (a vue.js-based UI)

To have a better idea of how contributing on a plugin within the Girder community, let's read the Plugin Development documentation.

6.2 Installation

You can either install the Slicer Package Manager natively on your machine or inside it's own virtual environment.

6.2.1 Virtual environment

While not strictly required, it is recommended to install the Slicer Package Manager and Girder within its own virtual environment to isolate its dependencies from other python packages. To generate a new virtual environment, first install/update the `virtualenv` and `pip` packages:

```
$ sudo pip install -U virtualenv pip
```

Now create a virtual environment using the virtualenv command. You can place the virtual environment directory wherever you want, but it should not be moved. The following command will generate a new directory called `slicer_package_manager_env` in your home directory:

```
$ virtualenv ~/slicer_package_manager_env
```

Then to enter in the virtual environment, use the command:

```
$ . ~/slicer_package_manager_env/bin/activate
```

Note: The (slicer_package_manager_env) prepended to your prompt indicates you have entered the virtual environment.

6.2.2 Install from Git

To easily develop the Slicer Package Manager, you will need to use some of Girder commands. So let's start by installing Girder:

```
$ git clone https://github.com/girder/girder.git
$ cd girder
$ pip install -e .
```

Then, let's install the Slicer Package Manager server plugin:

```
$ git clone https://github.com/girder/slicer_package_manager
$ cd slicer_package_manager
$ pip install -e .[test]
```

This will provide you all the package needed to run the development environment. Then install the front-end web client development dependencies:

```
$ girder build --dev
```

6.2.3 Run

To run the server, first make sure the Mongo daemon is running. To manually start it, run:

```
$ mongod &
```

Then to run Girder itself, just use the following command:

```
$ girder-server --dev
```

The application should be accessible on http://localhost:8080/ in your web browser.

6.3 During development

Once Girder is started via `girder-server`, the server will reload itself whenever a server source file is modified. If you are doing front-end development, it's much faster to use a watch process to perform automatic fast rebuilds of your code whenever you make changes to source files.

If you are front-end development of Slicer package manager plugin, use:

```
$ girder build --watch-plugin slicer_package_manager
```

6.4 Server side development

See the Server Development documentation to know more about the good development practise in Girder

6.5 Client side development

See the Client Development documentation to know more about the good development practise in Girder

6.6 Python client development

The development of the **Slicer Package Manager Client** is in Python, and it uses ruff for code style validation.

The python client use click, a command line library for Python.

6.7 Testing

Tests are the base of software development, they meant to check if what you've expected is really happening and find issues you didn't even think about. There are few thing you should know about test within the Slicer Package Manager.

6.7.1 Server Side Testing

As part of Girder, server test are done using pytest. Let's read the server test documentation to know more about Girder testing.

6.7.2 Python Client Testing

The Python Client use pytest to test its API. It also uses a tool named pytest-vcr to record the server responses and be able to test the client within CircleCI.

Note: Each time the client will change, or the tests, you will have to record the server an other time by running the tests manually. But first, you will have to delete the old records. All the server records should be saved as `.yml` file into the `cassettes` folder next to your tests. Delete this folder, and then run the tests again, it should create new records automatically.

To run manually these test run the following command:

```
$ pytest --tb=long plugin_tests/python_client_tests/test_python_client.py
```

The CLI is also briefly tested using a shell script. To see an example let's take a look at the Source Code

This test is also used within CircleCi.

To run locally this test, from the `slicer_package_manager` folder run:

```
$ cd plugin_tests/python_client_tests
$ ./slicer_extension_manager_client_test.sh
```

It will run some of the commands available with the `slicer_package_manager_client`, check if the upload and the download works and then delete everything.

This script could be take as a good example of using the *Commands shell (CLI)*.

6.7.3 CircleCI tests

In the CircleCI configuration file, there are several test going on:

- *Server Side Testing*

  It will occurs each time a commit will be pushed on the repository.

- *Python Client Testing*

  Both the python client API and the CLI are tested

- Docker containers testing

  Test the build and the deploy of the different *Docker containers*.

6.8 Regenerate Documentation Locally

When developing new feature it's very important to add some documentation to explain the community what is it and how to use it. The Slicer Package Manager Documentation is build thanks to Sphinx, an open source documentation generator.

Here is some tools very useful to rapidly see what is result of your documentation.

In the `slicer_package_manager/docs` directory, just run:

```
$ make docs
```

This will automatically create the API documentation for you and open a web browser tab to visualize the documentation. If you don't want to open a new tab and just rebuild the documentation run:

```
$ make docs-only
```

6.9 Docker containers

Docker containers allow an easy use and setup of the Slicer Package Manager. There are 3 different containers that communique between themselves.

- The application container

  It contains both the **Girder** application with the **Slicer Package Manager** plugin enabled.

- The database container

  This one contains the **MongoDB** instance that allow the Girder and the Slicer Package Manager to store all the data as Applications, Releases, Application or Extension packages.

- The provisioning container

  This container is special, it is only used once both the Girder server and the Mongo server are running and connected to each other. It is meant to handle the server configuration and make the use of the Slicer Package Manager much easier. By doing that it **enables the Slicer Package Manager plugin**

within Girder, create the first **admin user**, and set up the **assetstore** used to store the binary files (In fact the DB only store reference to these files, the real data are stored on your own machine in this assetstore).

6.10 Server API

6.10.1 Subpackages

slicer_package_manager.api package

Submodules

slicer_package_manager.api.app module

slicer_package_manager.models package

Submodules

slicer_package_manager.models.extension module

slicer_package_manager.models.package module

6.10.2 Submodules

6.10.3 slicer_package_manager.constants module

6.10.4 slicer_package_manager.utilities module

6.11 Python Client API

6.11.1 slicer_package_manager_client package

class `slicer_package_manager_client.`**`Constant`**

Bases: `object`

A bunch of utilities constant, as to handle `Error` or set default parameters.

`CURRENT_FOLDER = '/home/docs/checkouts/readthedocs.org/user_builds/slicer-package-mana`

`DEFAULT_LIMIT = 50`

`DRAFT_RELEASE_NAME = 'draft'`

`EXTENSION_AREADY_UP_TO_DATE = 32`

`EXTENSION_NOW_UP_TO_DATE = 33`

`PACKAGE_NOW_UP_TO_DATE = 31`

`WIDTH = 25`

class `slicer_package_manager_client.`**`SlicerPackageClient`**(*host=None*, *port=None*, *apiRoot=None*, *scheme=None*, *apiUrl=None*, *progressReporterCls=None*)

Bases: `girder_client.GirderClient`

The SlicerPackageClient allows to use the slicer_package_manager plugin of Girder, which allows you to manage the following top-level entities:

- Application
- Release
- Draft
- Package
- Extension

You may also choose the collection in which to create the application. It's also possible to provide a collection ID to use as the parent collection for creating the application.

In this case, you must provide the `coll_id` argument to use all the commands on these applications. By default, all commands look for applications that are under a collection named `Applications`.

`createApp`(*name*, *desc=None*, *coll_id=None*, *coll_name=None*, *coll_desc=None*, *public=None*)

Create a new application in the collection which correspond to `coll_id`, by default it will create the application in the collection named `Applications`. The application will contain a `draft` folder. Two templates names will be set as a metadata of this new application. One for determine each future uploaded application package and the other to determine each future uploaded extension. It's also possible to create a new collection by specifying "coll_name". If this collection already exist it will use it.

Parameters

- **`name`** – name of the new application
- **`desc`** – Optional description of the application
- **`coll_id`** – Id of an existing collection
- **`coll_name`** – Name of the collection
- **`coll_desc`** – Optional description of the new collection
- **`public`** – Whether the collection should be publicly visible

Returns The new application

`createRelease`(*app_name*, *name*, *revision*, *coll_id=None*, *desc=None*)

Create a new release within the application corresponding to `app_name`.

Parameters

- **`app_name`** – Name of the application
- **`name`** – Name of the release
- **`revision`** – Revision of the application
- **`coll_id`** – Collection ID
- **`desc`** – Description of the release

Returns The new release

deleteApp(*name*, *coll_id=None*)
Delete the application by ID.

Parameters

- **name** – application name
- **coll_id** – Collection ID

Returns The deleted application

deleteApplicationPackage(*app_name*, *id_or_name*, *coll_id=None*)
Delete an application package within an application.

Parameters

- **app_name** – Name of the application
- **id_or_name** – Package ID or name
- **coll_id** – Collection ID

Returns The deleted application package

deleteDraftRelease(*app_name*, *revision*, *coll_id=None*)
Delete a specific revision within the Draft release.

Parameters

- **app_name** – Name of the application
- **revision** – Revision of the release
- **coll_id** – Collection ID

Returns The deleted release

deleteExtension(*app_name*, *id_or_name*, *coll_id=None*)
Delete an extension within an application.

Parameters

- **app_name** – Name of the application
- **id_or_name** – Extension ID or name
- **coll_id** – Collection ID

Returns The deleted extension

deleteRelease(*app_name*, *name*, *coll_id=None*)
Delete a release within an application.

Parameters

- **app_name** – Name of the application
- **name** – Name of the release
- **coll_id** – Collection ID

Returns The deleted release

downloadApplicationPackage(*app_name*, *id_or_name*, *coll_id=None*, *dir_path='/home/docs/checkouts/readthedocs.org/user_builds/slicer-package-manager/checkouts/latest/docs'*)
Download an application package by ID and store it in the given option `dir_path`. When we use the package id in `id_or_name`, the parameter `app_name` is ignored.

Parameters

- **`app_name`** – Name of the application
- **`id_or_name`** – ID or name of the package
- **`coll_id`** – Collection ID
- **`dir_path`** – Path of the directory where the application package has to be downloaded

Returns The downloaded package

`downloadExtension`(*app_name*, *id_or_name*, *coll_id=None*, *dir_path='/home/docs/checkouts/readthedocs.org/user_builds/ package-manager/checkouts/latest/docs'*)

Download an extension by ID and store it in the given option `dir_path`. When we use the extension id in `id_or_name`, the parameter `app_name` is ignored.

Parameters

- **`app_name`** – Name of the application
- **`id_or_name`** – ID or name of the extension
- **`coll_id`** – Collection ID
- **`dir_path`** – Path of the directory where the extension has to be downloaded

Returns The downloaded extension

`listApp`(*name=None*, *coll_id=None*)

List all the applications within a specific collection by providing the option `coll_id`. By default it will list within the collection `Applications`. It can also lead to get the application by name.

Parameters

- **`name`** – application mame
- **`coll_id`** – Collection ID

Returns A list of applications

`listApplicationPackage`(*app_name*, *coll_id=None*, *name=None*, *pkg_os=None*, *arch=None*, *revision=None*, *version=None*, *release=None*, *limit=50*)

List the application packages filtered by some optional parameters (os, arch, . . .).

By default only the first N application packages are listed. Setting the `limit` parameter to *0* removes this restriction.

It's also possible to specify the `--release` option to list all the package from a specific release.

Parameters

- **`app_name`** – Name of the application
- **`coll_id`** – Collection ID
- **`name`** – Base name of the application package
- **`pkg_os`** – The target operating system of the package
- **`arch`** – The os chip architecture
- **`revision`** – Revision of the application
- **`version`** – Version of the application
- **`release`** – Name or ID of the release
- **`limit`** – Limit of the number of applications listed (see *`Constant.DEFAULT_LIMIT`*)

Returns A list of application package filtered by optional parameters

`listDraftRelease`(*app_name*, *coll_id=None*, *revision=None*, *limit=50*, *offset=0*)
List the draft releases with an offset option to list only the older ones.

By default only the first N releases are listed. Setting `limit` parameter to *0* removes this restriction.

It's also possible to list one release within the Draft release by providing its specific revision.

Parameters

- **`app_name`** – Name of the application
- **`coll_id`** – Collection ID
- **`revision`** – Revision of the release
- **`limit`** – Limit of the number of draft releases listed (see *`Constant.DEFAULT_LIMIT`*)
- **`offset`** – offset to list only older revisions

Returns The list of draft release

`listExtension`(*app_name*, *coll_id=None*, *name=None*, *ext_os=None*, *arch=None*, *app_revision=None*, *release='draft'*, *query=None*, *limit=50*, *all=False*)
List the extensions of a specific application `app_name`.

By default only the first N extensions within the `draft` release are listed. Setting `limit` parameter to *0* removes this restriction.

Specifying optional parameters like *ext_os* or *arch* allows to return the corresponding subset.

Passing `all=True` option allow to list all the extensions from all the releases of an application.

Parameters

- **`app_name`** – Name of the application
- **`coll_id`** – Collection ID
- **`name`** – Base name of the extension
- **`ext_os`** – The target operating system of the package
- **`arch`** – The os chip architecture
- **`app_revision`** – Revision of the application
- **`release`** – Name of the release
- **`query`** – Text expected to be found in the extension name or description
- **`limit`** – Limit of the number of extensions listed (see *`Constant.DEFAULT_LIMIT`*)
- **`all`** – Boolean that allow to list extensions from all the release

Returns A list of extensions filtered by optional parameters

`listRelease`(*app_name*, *name=None*, *coll_id=None*)
List all the release within an application. It's also able to get one specific release by name.

Parameters

- **`app_name`** – Name of the application
- **`name`** – Name of the release
- **`coll_id`** – Collection ID

Returns A list of all the release within the application

uploadApplicationPackage (*filepath*, *app_name*, *pkg_os*, *arch*, *name*, *repo_type*, *repo_url*, *revision*, *version*, *build_date=None*, *coll_id=None*, *desc=''*, *pre_release=False*)

Upload an application package by providing a path to the file. It can also be used to update an existing one.

Parameters

- **filepath** – The path to the file
- **app_name** – The name of the application
- **pkg_os** – The target operating system of the package
- **arch** – The os chip architecture
- **name** – The baseName of the package
- **repo_type** – Type of the repository
- **repo_url** – Url of the repository
- **revision** – The revision of the application
- **version** – The version of the application
- **build_date** – The build timestamp specified as a datetime string. Default set to current date and time.
- **coll_id** – Collection ID
- **desc** – The description of the application package
- **pre_release** – Boolean to specify if the package is ready to be distributed

Returns The uploaded application package

uploadExtension (*filepath*, *app_name*, *ext_os*, *arch*, *name*, *repo_type*, *repo_url*, *revision*, *app_revision*, *desc=''*, *icon_url=''*, *category=None*, *homepage=''*, *screenshots=None*, *contributors=None*, *dependency=None*, *coll_id=None*, *force=False*)

Upload an extension by providing a path to the file. It can also be used to update an existing one, in this case the upload is done only if the extension has a different revision than the old one.

Parameters

- **filepath** – The path to the file
- **app_name** – The name of the application
- **ext_os** – The target operating system of the package
- **arch** – The os chip architecture
- **name** – The baseName of the extension
- **repo_type** – Type of the repository
- **repo_url** – Url of the repository
- **revision** – The revision of the extension
- **app_revision** – The revision of the application supported by the extension
- **desc** – The description of the extension
- **icon_url** – Url of the extension's logo

- **`category`** – Category of the extension
- **`homepage`** – Url of the extension's homepage
- **`screenshots`** – Space-separate list of URLs of screenshots for the extension.
- **`contributors`** – List of contributors of the extension.
- **`dependency`** – List of the required extensions to use this one.
- **`coll_id`** – Collection ID
- **`force`** – To force update the binary file

Returns The uploaded extension

exception `slicer_package_manager_client.`**`SlicerPackageManagerError`**
Bases: `Exception`

Submodules

slicer_package_manager_client.cli module

CHAPTER 7

Credits

Please see the GitHub project page contributors.

CHAPTER 8

Making a release

A core developer should use the following steps to create a release *X.Y.Z* of **slicer-package-manager** and **slicer-package-manager-client** on PyPI.

8.1 Prerequisites

- All CI tests are passing on CircleCI.
- You have a GPG signing key.

8.2 Documentation conventions

The commands reported below should be evaluated in the same terminal session.

Commands to evaluate starts with a dollar sign. For example:

```
$ echo "Hello"
Hello
```

means that `echo "Hello"` should be copied and evaluated in the terminal.

8.3 Setting up environment

1. First, register for an account on PyPI.
2. If not already the case, ask to be added as a `Package Index Maintainer`.
3. Create a `~/.pypirc` file with your login credentials:

```
[distutils]
index-servers =
  pypi
  pypitest

[pypi]
username=__token__
password=<your-token>

[pypitest]
repository=https://test.pypi.org/legacy/
username=__token__
password=<your-token>
```

where `<your-token>` correspond to the API token associated with your PyPI account.

8.4 PyPI: Step-by-step

1. Make sure that all CI tests are passing on CircleCI.
2. Download the latest sources

```
$ cd /tmp && \
  git clone git@github.com:girder/slicer_package_manager && \
  cd slicer_package_manager
```

3. List all tags sorted by version

```
$ git fetch --tags && \
  git tag -l | sort -V
```

4. Choose the next release version number

```
$ release=X.Y.Z
```

Warning: To ensure the packages are uploaded on PyPI, tags must match this regular expression: `^[0-9]+(\.[0-9]+)*(\.post[0-9]+)?$`.

5. In `CHANGES.rst` replace `Next Release` section header with `X.Y.Z`, in `python_client/slicer_package_manager_client/__init__.py` update version with `X.Y.Z` and commit the changes.

```
$ git add CHANGES.rst && \
  git add python_client/slicer_package_manager_client/__init__.py && \
  git commit -m "slicer-package-manager[-client] ${release}"
```

6. Tag the release

```
$ git tag --sign -m "slicer-package-manager[-client] ${release}" ${release}␣
↪main
```

Warning: We recommend using a GPG signing key to sign the tag.

7. Create the source distribution and wheel for **slicer-package-manager**

```
$ pipx run build
```

Note: pipx allows to directly run the build frontend without having to explicitly install it.

To install *pipx*:

```
$ python3 -m pip install --user pipx
```

8. Create the source distribution and wheel for **slicer-package-manager-client**

```
$ pipx run build ./python_client
```

8. Publish the both release tag and the main branch

```
$ git push origin ${release} && \
  git push origin main
```

9. Upload the distributions on PyPI

```
$ pipx run twine upload dist/*
$ pipx run twine upload python_client/dist/*
```

Note: To first upload on TestPyPI , do the following:

```
$ pipx run twine upload -r pypitest dist/*
$ pipx run twine upload -r pypitest python_client/dist/*
```

10. Create a clean testing environment to test the installation

```
$ pushd $(mktemp -d) && \
  mkvirtualenv slicer-package-manager-${release}-install-test && \
  pip install slicer-package-manager==${release}

$ pushd $(mktemp -d) && \
  mkvirtualenv slicer-package-manager-client-${release}-install-test && \
  pip install slicer-package-manager-client==${release} && \
  slicer_package_manager_client --version
```

Note: If the `mkvirtualenv` command is not available, this means you do not have virtualenvwrapper installed, in that case, you could either install it or directly use virtualenv or venv.

To install from TestPyPI, do the following:

```
$ pip install -i https://test.pypi.org/simple slicer-package-manager==$
↪{release}
```

12. Cleanup

```
$ popd && \
  deactivate  && \
  rm -rf dist/* && \
  rmvirtualenv slicer-package-manager-${release}-install-test && \
  rm -rf python_client/dist/* && \
  rmvirtualenv slicer-package-manager-client-${release}-install-test
```

13. Add a `Next Release` section back in *CHANGES.rst*, commit and push local changes.

```
$ git add CHANGES.rst && \
  git commit -m "CHANGES.rst: Add \"Next Release\" section [ci skip]" && \
  git push origin main
```

CHAPTER 9

Release Notes

This is the list of **Slicer Package Manager** changes between each release. For full details, see the commit logs at https://github.com/girder/slicer_package_manager

9.1 Next Release

9.2 0.8.0

9.2.1 New Features

Server

- Associate application & extension package item with checksum.
  - After uploading an application or extension package, the item metadata will include *sha512* metadata entry.
  - After uploading additional files, the item metadata remains unchanged.
  - After removing the second to last files, the *sha512* item metadata is updated to match the checksum of the last file.
  - After removing all the files, the *sha512* item metadata is set to an empty string.

9.2.2 Documentation

- Remove obsolete `cleanNightly.sh` script and update faq.
- Update developer installation instructions to use Girder 3.x commands.

9.2.3 Bug fixes

Python Client

- Fix python client test requirements adding "pytest" and "pytest-girder".
- Attempting to install the python client using Python `< 3.7` will now report an error message.

Server

- Update extension & package delete endpoints
  - Explicitly check that user can access the associated application folder.
  - Return a confirmation message.

9.2.4 Internal

- Require Python `>= 3.6` for the server. This is consistent with the version associated with the Girder test Docker image girder/girder_test:latest built from girder/.circleci/Dockerfile.
- The required version previously set to "3.7" in version "0.7.0" for both client and server but it was not enforced due to an incorrect setup parameter. It should have been specified as `python_requires` instead of `python_require` (as defined in PEP 440).
- Re-factor and simplify code based on the newly introduced pre-commit hooks and ruff checks (`codespell`, `pyupgrade` and `ruff`).
- Add type annotations to python client CLI.

9.2.5 Tests

- Add GitHub Actions workflow to run pre-commit hooks.
  - Add "codespell" pre-commit hook and fix typos.
  - Add pyupgrade pre-commit hook specifying "–py36-plus" and updates codes accordingly.
  - Add ruff pre-commit hook enabling the following checks:

```
"A",           # flake8-builtins
"ARG",         # flake8-unused-arguments
"B",           # flake8-bugbear
"BLE",         # flake8-blind-except
"C4",          # flake8-comprehensions
"COM",         # flake8-commas
"D",           # pydocstyle (aka flake8-docstrings)
"E", "F", "W", # flake8
"EXE",         # flake8-executable
"EM",          # flake8-errmsg
"G",           # flake8-logging-format
"ICN",         # flake8-import-conventions
"ISC",         # flake8-implicit-str-concat
"N",           # pep8-naming
"PIE",         # flake8-pie
"PGH",         # pygrep-hooks
"PL",          # pylint
```

(continues on next page)

(continued from previous page)

```
"PT",           # flake8-pytest-style
"Q",            # flake8-quotes
"RSE",          # flake8-raise
"RUF",          # Ruff-specific
"S",            # flake8-bandit
"SIM",          # flake8-simplify
"SLF",          # flake8-self
"YTT",          # flake8-2020
```

9.3 0.7.1

9.3.1 Bug fixes

Python Client

- Fix wheel ensuring `_vendor.bson` package is distributed.

9.4 0.7.0

9.4.1 Documentation

- Re-organize and simplify documentation.

9.4.2 Internal

- Require Python >= 3.7 for both python client and server.
- Update development status to `Production/Stable`.
- Vendorize `bson.objectid` from PyMongo to support installing the client alongside the server and workaround incompatibilities between standalone `bson` package and the one provided by PyMongo.

Python Client

- Support publishing python client sdist and wheel named `slicer-package-manager-client`.

9.5 0.6.0

9.5.1 New Features

- Support listing extension with a `query` parameter specifying the text expected to be found in the extension name or description.

9.5.2 Bug fixes

Server

- Fix creation of extension in private application.
- Ensure user or administrator errors associated with API endpoints are displayed and associated with HTTP error code 400 by raising a `RestException` instead of a generic `Exception`.
- Update API endpoint *GET /app/{app_id}/extension* to always check user credentials.

9.6 0.5.0

9.6.1 New Features

- Require version information to be specified when uploading application packages. See #97.
- Add application package `build_date` metadata. User may specify a custom value formatted as a datetime string using the API endpoint or the python client. Default is set to current date and time.

Server

- Automatically update `release` metadata when packages are moved (or copied) between draft and release folders.
- Add convenience functions `slicer_package_manager.utilities.isApplicationFolder()`, `slicer_package_manager.utilities.isReleaseFolder()` and `slicer_package_manager.utilities.isDraftReleaseFolder()`.
- Add `slicer_package_manager.utilities.getReleaseFolder()` and simplify update of `downloadStats` release metadata to use the new function.

9.6.2 Bug fixes

- Remove partially implemented `codebase` metadata.
- Remove support for unused `packagetype` metadata.

9.6.3 Tests

- ExternalData:
  - Fix re-download of files if checksum does not match.
  - Re-factor fixture introducing `downloadExternals`.

9.7 0.4.0

9.7.1 New Features

- Support querying application packages given a release name. See #96.

9.7.2 Bug fixes

Server

- Ensure permissions are consistently checked in API endpoints implementation. See #95.
- Fix support for unauthenticated use of public API endpoints. See #95.

9.8 0.3.0

9.8.1 Bug fixes

Server

- Update implementation of `GET /app/:app_id/package` endpoint to properly handle `limit=0` parameter. See #94.

9.8.2 Documentation

- Add documentation to `slicer_package_manager.utilities.getOrCreateReleaseFolder()`.

9.9 0.2.0

9.9.1 Bug fixes

Server

- Update access level of API endpoints. See #89.
  - Creating or updating packages now always require credentials.
  - Retrieving list of applications, releases and packages are now public. Note that credentials are still required to retrieve data associated with private applications.

Python Client

- Fix handling of `--public`, `--all` and `--pre_release` flags. See #85.
- Update `draft list` command to support `--limit` argument. See #82.

9.9.2 Documentation

- Add maintainer documentation along with *Making a release* section.
- Improve description of `limit` in *`slicer_package_manager_client.SlicerPackageClient.listExtension()`* and *`slicer_package_manager_client.SlicerPackageClient.listApplicationPackage()`*. See #84.

9.9.3 Tests

- Simplify and refactor python client tests to facilitate maintenance. See #83 and #88.

9.10 0.1.0

9.10.1 New Features

- Transition server plugin from Girder 2.x to Girder 3.x. See #88.

9.11 Initial version

Developed by @Pierre-Assemat during his internship at Kitware in 2018.

9.11.1 Features

- Girder plugin implementing REST API endpoints.
- CLI *slicer_package_manager_client*
- Python client class `SlicerPackageClient`.

9.11.2 Documentation

- Administrator, user and developer documentation written in reStructuredText (RST), generated using sphinx and published at https://slicer-package-manager.readthedocs.io

9.11.3 Tests

- Continuous integration (CI) configured to run on CircleCI.
- Girder plugin tests.
- CLI and Python client tests leveraging pytest-vcr.

9.11.4 Provisioning

- Dockerfile and docker-compose files for provisioning a demo server.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index