Plugger

Jun 13, 2024

About

CHAPTER 1

Overview

LizardByte has the full documentation hosted on Read the Docs.

1.1 About

Plugger is a plug-in for Plex Media Player. The plug-in helps you manage your Plex Media Server plug-ins.

Some of the main features of the Plugger are:

- install/update/uninstall plugins
- search for plugins
- view stats about plugins (e.g. number of GitHub stars, forks, issues, etc.)
- view installed plugin logs

1.2 Integrations

1.3 Downloads

CHAPTER 2

Installation

The recommended method for running Plugger is to use the *bundle* in the latest release.

2.1 Bundle

The bundle is cross platform, meaning Linux, macOS, and Windows are supported.

1. Download the `plugger.bundle.zip` from the latest release
2. Extract the contents to your Plex Media Server Plugins directory.

Tip: See How do I find the Plug-Ins folder for information specific to your Plex server install.

2.2 Docker

Docker images are available on Dockerhub and ghcr.io.

See *Docker* for additional information.

2.3 Source

Caution: Installing from source is not recommended most users.

1. Follow the steps in *Build*.
2. Move the compiled `plugger.bundle` to your Plex Media Server Plugins directory.

CHAPTER 3

Docker

3.1 lizardbyte/plugger

This is a docker-mod for plex that adds Plugger to plex as a plugin, to be downloaded/updated during container start.

This image extends the plex image, and is not intended to be created as a separate container.

3.2 Installation

In plex docker arguments, set an environment variable `DOCKER_MODS=lizardbyte/plugger:latest` or `DOCKER_MODS=ghcr.io/lizardbyte/plugger:latest`

If adding multiple mods, enter them in an array separated by `|`, such as `DOCKER_MODS=lizardbyte/plugger:latest|linuxserver/mods:other-plex-mod`

3.3 Supported Architectures

Linuxserver.io docker mods do not support multi-arch images; however this image should run on any architecture. If you have issues with this image on a specific architecture, please open an issue on GitHub.

CHAPTER 4

Usage

Minimal setup is required to use Plugger. In addition to the installation, a couple of settings must be configured.

1. Navigate to the *Plugins* menu within the Plex server settings.
2. Select the gear cog when hovering over the Plugger plugin tile.
3. Set the values of the preferences and save.

4.1 Preferences

4.1.1 Web UI Locale

Description The localization value to use for translations.

Default `en`

4.1.2 Web UI Host Address

Description The host address to bind the Web UI to.

Default `0.0.0.0`

4.1.3 Web UI Port

Description The port to bind the Web UI to.

Default `9595`

4.1.4 Log all web server messages

Description If set to `True`, all web server messages will be logged. This will include logging requests and status codes when requesting any resource. It is recommended to keep this disabled unless debugging.

Default `False`

CHAPTER 5

Troubleshooting

5.1 Plugin Logs

See Plugin Log Files for the plugin log directory.

Plex uses rolling logs. There will be six log files available. The newest log file will be named `dev.lizardbyte.plugger.log`. There will be additional log files with the same name, appended with a *1-5*.

It is best to replicate the issue you are experiencing, then review the latest log file. The information in the log file may seem cryptic. If so it would be best to reach out for support.

Attention: Before uploading logs, it would be wise to review the data in the log file. Plex does not filter the masked settings (e.g. credentials) out of the log file.

5.2 Plex Media Server Logs

If you have a more severe problem, you may need to troubleshoot an issue beyond the plugin itself. See Plex Media Server Logs for more information.

CHAPTER 6

Changelog

CHAPTER 7

Contributing

Read our contribution guide in our organization level docs.

CHAPTER 8

Database

The database of plugins is held in our PluggerDB repository. The majority of plugin metadata is pulled using GitHub's API, but categories are added manually. To add plugins, or update categories for a plugin, you can open an issue on PluggerDB. Please follow the instructions in the PluggerDB readme.

CHAPTER 9

Build

Compiling Plugger is fairly simple; however it is recommended to use Python 2.7 since the Plex framework is using Python 2.7.

9.1 Clone

Ensure git is installed and run the following:

```
git clone https://github.com/lizardbyte/plugger.git plugger.bundle
cd ./plugger.bundle
```

9.2 Setup venv

It is recommended to setup and activate a venv.

9.3 Install Requirements

Install Requirements

```
python -m pip install --upgrade --target=./Contents/Libraries/Shared -r␣
↪requirements.txt --no-warn-script-location
```

Development Requirements

```
python -m pip install -r requirements-dev.txt
```

9.4 Build Plist

```
python ./scripts/build_plist.py
```

9.5 npm dependencies

Install nodejs and npm. Downloads available here.

Install npm dependencies.

```
npm install
```

Move modules directory.

Linux/macOS

```
mv ./node_modules ./Contents/Resources/web
```

Windows

```
move .\node_modules .\Contents\Resources\web
```

9.6 Remote Build

It may be beneficial to build remotely in some cases. This will enable easier building on different operating systems.

1. Fork the project
2. Activate workflows
3. Trigger the *CI* workflow manually
4. Download the artifacts from the workflow run summary

CHAPTER 10

Testing

10.1 Flake8

Plugger uses Flake8 for enforcing consistent code styling. Flake8 is included in the `requirements-dev.txt`.

The config file for flake8 is `.flake8`. This is already included in the root of the repo and should not be modified.

Test with Flake8

```
python -m flake8
```

10.2 Sphinx

Plugger uses Sphinx for documentation building. Sphinx is included in the `requirements-dev.txt`.

Plugger follows numpydoc styling and formatting in docstrings. This will be tested when building the docs. *numpydoc* is included in the `requirements-dev.txt`.

The config file for Sphinx is `docs/source/conf.py`. This is already included in the root of the repo and should not be modified.

Test with Sphinx

```
cd docs
make html
```

Alternatively

```
cd docs
sphinx-build -b html source build
```

10.3 pytest

Plugger uses pytest for unit testing. pytest is included in the `requirements-dev.txt`.

No config is required for pytest.

Test with pytest

```
python -m pytest
```

CHAPTER 11

__init__

Code.**Start**()
Start the plug-in.

This function is called when the plug-in first starts. It can be used to perform extra initialisation tasks such as configuring the environment and setting default attributes. See the archived Plex documentation Predefined functions for more information.

First preferences are validated using the `ValidatePrefs()` method. Then the flask web app is started.

Examples

```
>>> Start()
...
```

Code.**ValidatePrefs**()
Validate plug-in preferences.

This function is called when the user modifies their preferences. The developer can check the newly provided values to ensure they are correct (e.g. attempting a login to validate a username and password), and optionally return a `MessageContainer` to display any error information to the user. See the archived Plex documentation Predefined functions for more information.

Returns

MessageContainer Success or Error message dependeing on results of validation.

Examples

```
>>> ValidatePrefs()
...
```

CHAPTER 12

webapp

Code.webapp.**get_locale**()

Get the locale from the config.

Get the locale specified in the config. This does not need to be called as it is done so automatically by *babel*.

Returns

str The locale.

See also:

pyra.locales.get_locale Use this function instead.

Examples

```
>>> get_locale()
en
```

Code.webapp.**home**()

Serve the webapp home page.

This page is where most of the functionality for Plugger is provided.

Returns

render_template The rendered page.

Notes

The following routes trigger this function.

- */*
- */home*

Examples

```
>>> home()
```

Code.webapp.**image**(*img*)
Get image from static/images directory.

Returns

flask.send_from_directory The image.

Notes

The following routes trigger this function.

- */favicon.ico*

Examples

```
>>> image('favicon.ico')
```

Code.webapp.**install_plugin**()
Install a plugin.

Todo: Complete this function.

Code.webapp.**installed_plugins**()
Serve the list of installed plugins.

Code.webapp.**log_stream**(*plugin_identifier*)
Serve the plugin logs in plain text.

Collect and format the logs for the specified plugin.

Parameters

plugin_identifier [str] The reverse domain name of the plugin, e.g. *dev.lizardbyte.plugger*.

Returns

Response The text of the log files.

Notes

The following routes trigger this function.

- */log_stream/*
- */log_stream/<plugin name>*

Examples

```
>>> log_stream(plugin_identifier='dev.lizardbyte.plugger')
```

Code.webapp.**logs**(*plugin_identifier*)
Serve the plugin logs.

Collect and format the logs for the specified plugin.

Parameters

plugin_identifier [str] The reverse domain name of the plugin, e.g. *dev.lizardbyte.plugger*.

Returns

render_template The logs template with the requested information.

Notes

The following routes trigger this function.

- */logs/*
- */logs/<plugin name>*

Examples

```
>>> logs(plugin_identifier='dev.lizardbyte.plugger')
```

Code.webapp.**status**()
Check the status of Plugger.

This is useful for a healthcheck from Docker, and may have many other uses in the future.

Returns

dict A dictionary of the status.

Examples

```
>>> status()
```

Code.webapp.**translations**()
Serve the translations.

Returns

Response The translations.

Examples

```
>>> translations()
```

Python Module Index

C

Index