RetroArcher-plex

Mar 26, 2024

About

RetroArcher has this documentation hosted on Read the Docs.

CHAPTER 1

Overview

1.1 About

RetroArcher-plex is plug-in for Plex Media Player that connects plex to RetroArcher.

Note: This project is under development and not useful at this time. RetroArcher is in the process of being rewritten. You can follow progress on the RetroArcher GitHub.

1.2 Integrations

1.3 Downloads

CHAPTER 2

Changelog

CHAPTER 3

Contributing

Read our contribution guide in our organization level docs.

CHAPTER 4

Build

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

4.1 Clone

Ensure git is installed and run the following:

```
git clone https://github.com/lizardbyte/retroarcher-plex.git retroarcher-
↪plex.bundle
cd ./retroarcher-plex.bundle
```

4.2 Setup venv

It is recommended to setup and activate a venv.

4.3 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
```

4.4 Build Plist

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

4.5 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 5

Testing

Unless otherwise specified the `requirements.txt` file is located in the `scripts` directory.

5.1 Flake8

RetroArcher uses Flake8 for enforcing consistent code styling. Flake8 is included in the `requirements.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
```

5.2 Sphinx

RetroArcher uses Sphinx for documentation building. Sphinx is included in the `requirements.txt`.

RetroArcher follows numpydoc styling and formatting in docstrings. This will be tested when building the docs.

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
```

5.3 pytest

Todo: PyTest is not yet implemented.

RetroArcher uses pytest for unit testing. pytest is included in the `requirements.txt`.

No config is required for pytest.

Test with pytest

```
python -m pytest
```

CHAPTER 6

__init__

class Code.**RetroArcher**

Bases: plexhints.agent_kit.Movies

Class representing the RetroArcher Plex Movie Agent.

This class defines the metadata agent. See the archived Plex documentation Defining an agent class for more information.

References

name [str] A string defining the name of the agent for display in the GUI.

languages [list] A list of strings defining the languages supported by the agent. These values should be taken from the constants defined in the Locale API.

primary_provider [bool] A boolean value defining whether the agent is a primary metadata provider or not. Primary providers can be selected as the main source of metadata for a particular media type. If an agent is secondary (`primary_provider` is set to `False`) it will only be able to contribute to data provided by another primary agent.

fallback_agent [Optional[str]] A string containing the identifier of another agent to use as a fallback. If none of the matches returned by an agent are a close enough match to the given set of hints, this fallback agent will be called to attempt to find a better match.

accepts_from [Optional[list]] A list of strings containing the identifiers of agents that can contribute secondary data to primary data provided by this agent.

contributes_to [Optional[list]] A list of strings containing the identifiers of primary agents that the agent can contribute secondary data to.

Examples

```
>>> RetroArcher()
...
```

Attributes

contributes_to

Methods

search:	Search for an item.
update:	Add or update metadata for an item.

static search(*results*, *media*, *lang*, *manual*)
Search for an item.

When the media server needs an agent to perform a search, it calls the agent's `search` method. See the archived Plex documentation Searching for results to provide matches for media for more information.

Parameters

results [SearchResult] An empty container that the developer should populate with potential matches.

media [Media.Movie] An object containing hints to be used when performing the search.

lang [str] A string identifying the user's currently selected language. This will be one of the constants added to the agent's `languages` attribute.

manual [bool] A boolean value identifying whether the search was issued automatically during scanning, or manually by the user (in order to fix an incorrect match).

Returns

Optional[SearchResult] The search result object, if the search was successful.

Examples

```
>>> RetroArcher().search(results=..., media=..., lang='en', manual=True)
...
```

static update(*metadata*, *media*, *lang*, *force*)
Update metadata for an item.

Once an item has been successfully matched, it is added to the update queue. As the framework processes queued items, it calls the `update` method of the relevant agents. See the archived Plex documentation Adding metadata to media for more information.

Parameters

metadata [object] A pre-initialized metadata object if this is the first time the item is being updated, or the existing metadata object if the item is being refreshed.

media [object] An object containing information about the media hierarchy in the database.

lang [str] A string identifying which language should be used for the metadata. This will be one of the constants defined in the agent's `languages` attribute.

force [bool] A boolean value identifying whether the user forced a full refresh of the metadata. If this argument is `True`, all metadata should be refreshed, regardless of whether it has been populated previously.

Examples

```
>>> RetroArcher().update(metadata=..., media=..., lang='en', force=True)
...
```

Code.**SetRating**(*key*, *rating*)

This function is called when the user sets the rating of a metadata item returned by the plug-in. The *key* argument will be equal to the value of the item's `rating_key` attribute. See the archived Plex documentation Predefined functions for more information.

Parameters

key [str] This will be equal to the value of the item's rating_key attribute.

rating [float] A float between 0 and 10 specifying the item's rating.

Examples

```
>>> SetRating(key='123456', rating=8.8)
...
```

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.

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 7

helpers

Code.helpers.**get_game_name**(*media*, *media_filename*)

Get the game name from the given media object or media_filename.

Parameters

media [Media.Movie] Media object provided by plex framework.

media_filename [str] The filename of the media object representing the game.

Returns

str The game name.

Examples

```
>>> get_game_name(media=media, media_filename='007 - GoldenEye (USA).mp4')
'007 - GoldenEye'
```

Code.helpers.**get_game_platform**(*path*)

Get the game platform from the given path.

Parameters

path [str] The path of the media object representing the game.

Returns

str The platform of the game.

Examples

```
>>> get_game_platform(path='.../media/Nintendo 64/007 - GoldenEye (USA).mp4')
'Nintendo 64'
```

Code.helpers.**get_game_version**(*media_filename*)
Get the game version from the given media_filename.

Parameters

media_filename [str] The filename of the media object representing the game.

Returns

str The version of the game.

Examples

```
>>> get_game_version(media_filename='007 - GoldenEye (USA).mp4')
'(USA)'
```

Code.helpers.**get_list_of_substrings**(*string_subject*, *string1*, *string2*)
Get list of strings between two strings.

Parameters

string_subject [str] The string to search.

string1 [str] Search start string.

string2 [str] Search end string.

Returns

List List of strings.

Examples

```
>>> get_list_of_substrings(string_subject='{test1}{test2}{test3}', string1='{',␣
↪string2='}')
['test1', 'test2', 'test3']
```

Code.helpers.**get_media_fullpath**(*media*)
Get the fullpath of the plex media item.

Parameters

media [Media.Movie] The media item passed in from plex.

Returns

Tuple[str, str] A tuple containing the fullpath and media_filename

Examples

```
>>> get_media_fullpath(media=media)
('...', '...')
```

Python Module Index

C

Index