lcu-driver

Aug 13, 2023

Contents:

lcu-driver is a Python library made to communicate with League of Legends Client API in a easy way. It provides an API capable of handling LCU connection status and websockets events for you and prepare HTTP requests to be used directly with endpoints. Inspired in lcu-connector. It works on Windows, Linux and MacOS.

CHAPTER 1

Installation

```
$ pip install lcu-driver
```

CHAPTER 2

Contents

2.1 Getting Started with lcu-driver

2.1.1 Basics

An instance of Connector is responsible for holding both event handlers and create connections.

The connection instances will fire two events, **ready** and **close** and you can handle those using the decorators it provides.

```
from lcu_driver import Connector

connector = Connector()


@connector.ready
async def connect(connection):
    print('LCU API is ready to be used.')


@connector.close
async def disconnect(connection):
    print('Finished task')

connector.start()
```

2.1.2 HTTP Requests

To easily make requests each connection provides a method wrapper around aiohttp.Request that allow us to make request without dealing with authentication or the port where it is running.

If you don't know where to find the Client APIs documentation checkout out Rift Explorer.

```
from lcu_driver import Connector

connector = Connector()


@connector.ready
async def connect(connection):
    summoner = await connection.request('get', '/lol-summoner/v1/current-summoner')
    print(await summoner.json())

connector.start()
```

You can find more about the method *here*.

2.1.3 Websocket

```
from lcu_driver import Connector

connector = Connector()


@connector.ready
async def connect(connection):
    print('LCU API is ready to be used.')


@connector.ws.register('/lol-summoner/v1/current-summoner', event_types=('UPDATE',))
async def icon_changed(connection, event):
    print(f'The summoner {event.data["displayName"]} was updated.')

connector.start()
```

If you close the client you may notice it will not stop running. That happens because by default, if you subscribed to any endpoint, it will look for new clients once it's done.

To stop the connector once you disconnect runs all connection tasks you can use

```
@connector.close
def disconnect(connection):
    print('The client was closed')
    await connector.stop()
```

URL Patterns

What if you wanted to subscribe to all summoner events? You can simple register */lol-summoner/* and since it ends with a trailing slash it will match every event url starting with it.

Examples

Not ending with trailing slash

@connector.ws.register('/lol-summoner') will only match */lol-summoner* and will never be fired because the endpoint doesn't exit.

Ending with trailing slash

@connector.ws.register('/lol-summoner/') will match every event starting with it.

It will match: */lol-summoner/v1/current-summoner*, */lol-summoner/v1/current-summoner/icon* and */lol-summoner/v1/summoners*

But not: */lol-perks/v1/pages*

2.2 Code Examples

2.2.1 Change summoner icon

Every time you run the code, if the summoner is logged in the client, it will change the current summoner icon for a random chinese icon (ids 50 to 78).

```
from random import randint

from lcu_driver import Connector

connector = Connector()


async def set_random_icon(connection):
    # random number of a chinese icon
    random_number = randint(50, 78)

    # make the request to set the icon
    icon = await connection.request('put', '/lol-summoner/v1/current-summoner/icon',
                                    data={'profileIconId': random_number})

    # if HTTP status code is 201 the icon was applied successfully
    if icon.status == 201:
        print(f'Chinese icon number {random_number} was set correctly.')
    else:
        print('Unknown problem, the icon was not set.')


# fired when LCU API is ready to be used
@connector.ready
async def connect(connection):
    print('LCU API is ready to be used.')

    # check if the user is already logged into his account
    summoner = await connection.request('get', '/lol-summoner/v1/current-summoner')
    if summoner.status != 200:
        print('Please login into your account to change your icon and restart the␣
↪script...')
    else:
        print('Setting new icon...')
        await set_random_icon(connection)


# fired when League Client is closed (or disconnected from websocket)
@connector.close
```

(continues on next page)

(continued from previous page)

```
async def disconnect(_):
    print('The client have been closed!')

# starts the connector
connector.start()
```

2.2.2 Listening for summoner profile updates

Once you run the code, the event handler (coroutine function) every time something about the summoner changes (e.g. name change, profile icon change, level, ...). The connection will keep alive until the client is closed.

```
from lcu_driver import Connector

connector = Connector()


# fired when LCU API is ready to be used
@connector.ready
async def connect(connection):
    print('LCU API is ready to be used.')


# fired when League Client is closed (or disconnected from websocket)
@connector.close
async def disconnect(_):
    print('The client have been closed!')
    await connector.stop()


# subscribe to '/lol-summoner/v1/current-summoner' endpoint for the UPDATE event
# when an update to the user happen (e.g. name change, profile icon change, level, ...
↪) the function will be called
@connector.ws.register('/lol-summoner/v1/current-summoner', event_types=('UPDATE',))
async def icon_changed(connection, event):
    print(f'The summoner {event.data["displayName"]} was updated.')


# starts the connector
connector.start()
```

2.3 API Reference

2.3.1 Connector Module

class lcu_driver.connector.**BaseConnector**(*loop=None*)

Bases: *lcu_driver.events.managers.ConnectorEventManager*, abc.ABC

register_connection(*connection: lcu_driver.connection.Connection*)

Creates a connection and saves a reference to it

should_run_ws

unregister_connection(*lcu_pid*)
Cancel the connection

class lcu_driver.connector.**Connector**(*, *loop=None*)
Bases: *lcu_driver.connector.BaseConnector*

register_connection(*connection*)
Creates a connection and saves a reference to it

should_run_ws

start() → None
Starts the connector. This method should be overridden if different behavior is required.

Return type None

stop() → None
Flag the connector to don't look for more clients once the connection finishes his job.

Return type None

unregister_connection(*_*)
Cancel the connection

class lcu_driver.connector.**MultipleClientConnector**(*, *loop=None*)
Bases: *lcu_driver.connector.BaseConnector*

register_connection(*connection*)
Creates a connection and saves a reference to it

should_run_ws

start() → None

unregister_connection(*lcu_pid*)
Cancel the connection

2.3.2 Connection module

class lcu_driver.connection.**Connection**(*connector*, *process_or_string: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7f41957dc550>, str]*)
Bases: object

Parameters

- **connector** (*lcu_driver.connector.Connector*) – Connector instance where connection should look for events handlers
- **process_or_string** (psutil.Process or string) – psutil.Process object or lockfile string

address
Return HTTPS Base URL

Returns str

auth_key
Return League Client API current connection password

Return type str or None

init()
Initialize the connection. It's called by the connector when it finds a connection

Return type none

installation_path
Return League Client installation path

Returns optional[str]

pid
League Client Process Id

Return type int

port
Return the League Client API current connection port

Return type int

protocols
Return a tuple with League Client API supported protocols

Return type tuple(str, str)

request(*method: str*, *endpoint: str*, ***kwargs*)
Run an HTTP request against the API

Parameters **method** – HTTP method :type method: str :param endpoint: Request Endpoint :type endpoint: str :param

kwargs: Arguments for aiohttp.Request. The **data** keyworded argument will be JSON encoded automatically.

run_ws()
Start the websoocket connection. This is responsible to raise Connector close event and handling the websocket events.

Returns None

ws_address
Return Websocket Base URL

Returns str

2.3.3 Events package

class lcu_driver.events.managers.**ConnectorEventManager**
Bases: abc.ABC

Connector Events Manager Base Class

close(*coro_func*)

handlers

open(*coro_func*)

ready(*coro_func*)

run_event(*event_name*, **args*, ***kwargs*)

class lcu_driver.events.managers.**WebsocketEventManager**
Bases: abc.ABC

Connector Events Manager Base Class

static match_event(*connector*, *connection*, *data*)
Match registered websocket events and create a task with each handler

register(*uri: str*, ***, *event_types: Iterable[T_co] = ('CREATE', 'UPDATE', 'DELETE')*)
Register an event for the given handler.

Parameters

- **uri** (*string*) – Endpoint to call. If the endpoint last character is a slash it will match all events starting with the endpoint.
- **event_types** (*tuple(str, str)*) – Expects an iterable. The allowed types are CREATE, UPDATE and DELETE (case-sensitive).

registered_uris
Websocket registered handlers

Return type list

class lcu_driver.events.responses.**WebsocketEventResponse**(***kwargs*)
Bases: object

2.4 Other tools

2.4.1 Rift Explorer

Created by Pupix and currently maintained by Ray, **Rift Explorer** is a must have when developing for the LCU. It generates always up to date documentation so you can find the endpoints you need.

Github: https://github.com/Pupix/rift-explorer

CHAPTER 3

References

- genindex
- modindex
- search

CHAPTER 4

Endorsement

lcu-driver isn't endorsed by Riot Games and doesn't reflect the views or opinions of Riot Games or anyone officially involved in producing or managing League of Legends. League of Legends and Riot Games are trademarks or registered trademarks of Riot Games, Inc. League of Legends © Riot Games, Inc.

Python Module Index

l

Index

U

W