Home Middleman API

Hubert Kasperek

March 15, 2023

Contents

1 Introduction

1.1 proxies

Home Middleman has 4 types of proxy:
HTTP **/api/httpp**
HTTPS **/api/httpps**
TXT HTTP **/api/txt/httpp**
TXT HTTPS **/api/txt/httpps**
Http and https as the name suggests, just do http/https request and return the content of the html page, while the txt versions do the same, but instead of returning the html code, they return the text content of the website
We can use proxies to use the internet more anonymously, more accessible to tools without page rendering options (txt proxy), or to collect html code or text from a website

1.2 file storage

File listing **/api/files/list** allows you to list files in the *upload/* folder (this folder is used to store various types of files uploaded by the user or returned by the API)
File upload **/api/upload** allows you to upload files to the server
File upload from link **/api/uploadLink** allows you to upload files to the server by providing a link to a website containing a file, e.g. *https://example.com/image.jpg*
File write **/api/write** this api allows you to save content directly to a file of any format
File move **/api/files/mv** allows you to change the path to the file (only in the *upload/* folder area) or to change the name/format of the file
File delete **/api/files/del** allows you to delete files (only in the *upload/* folder area)
File send **/api/files/send** this api allows you to send a file buffer via http post request
File storage is basically a cloud, we can store files on it that we can access from any device in the network. Since the same folder as file storage is used by some scrapers, we can also have access to the scrapped data from any device

1.3 scraper

Link scraper **/api/scraper/links** allows you to save all links from the webpage in the *websitelink.json* format
Img scraper **/api/scraper/imgs/** allows you to save all the available images from the webpage in the same format as they are on the web page
Cheerio html scraper **/api/scraper/cheeriohtml** allows you to save any page elements in the .html format, the syntax of the Cheerio library is used to select page elements
Scraper can be useful for collecting specific data of interest to us from html files

hosted by other servers, in routine they can be extremely useful, for example, to collect news from the world every 24 hours

1.4 clipboard

Get the latest saved clip **/api/clip** this api function is designed to return text, which is then to be used by home middleman clients to save this text to the clipboard (at least that's how it works in the Python client and the front-end of the application)
Save to clipboard **/api/clip/save** this api gives you the ability to save text to the clipboard for later use
Clipboard history **/api/clip/history** this api allows you to see history of saved clips in clipboard, to clear history you can use:
Clipboard erase **/api/clip/erase** this api clears all clipboard history including last saved text
Clipboard fulfills its task fully only when hmm client has the option of saving the returned text to the clipboard

1.5 notes

List notes **/api/notes** this api returns you saved notes
Add note **/api/notes/add** this api allows you to add a note
Remove note **/api/notes/del/** this api allows you to delete a note
Notes require the following: title, note text, and date.

1.6 tasks

Tasks listing **/api/task** this api returns the tasks saved on the server
Tasks returns listing **/api/task/log/** this api returns the values returned by tasks, e.g. if the scrapper is running and returns text instead of e.g. images or json, then this value is saved as a log entry
Count tasks **/api/task/count** this api returns the number of tasks on the server
Get the log entry at the specified index **/api/task/log/:num** this api returns one log entry, 0 is the first index, and 'n' is the newest one
Add new task **/api/task/add** this api allows you to add a new task
Delete task **/api/task/del/** this api allows you to delete a task
Start task **/api/task/run** allows you to start a task by specifying the name of the task you want to start
Start time task **/api/task/time/run** allows you to start a task by specifying the name of the task you want to start, but this one, unlike the previous one, is executed only after the time you specified
Tasks are saved specific configurations of the use of other api's, therefore only they can be used in the routine.

1.7 routine

Get information about the routine **/api/task/interval** returns information about tasks that are added to the routine (they are performed regularly according to the given time interval)
Add task to routine **/api/task/interval/add** allows you to add a task to the routine along with specifying how often (milliseconds by default, clients usually have it set to minutes) the given task should be performed
Delete task from routine **/api/task/interval/kill/** allows you to stop the routine of a given task
Count the tasks in the routine **/api/task/interval/count** this api function returns how many tasks run in intervals (routine)
Routine allows you to perform tasks at intervals, so you don't have to manually perform a given activity from time to time - the software will do it for you!

1.8 console

Write text in the server console **(GET, POST) /api/console** allows you to print the sent text in the server console
Useful for debugging, testing values or server settings

1.9 configurations

config export **/api/cfg/export** allows you to export task settings and routines as .json files in the *upload/* folder on the server
config import **/api/cfg/import** allows you to import task settings and routines from a .json file located on the server in the *upload/* folder, you can load an unlimited number of configurations on one run
restart **/api/restart** allows you to reset all global Home Middleman values (including logs) and configuration without having to shut down the entire server
configurations can be very useful if we want to be able to quickly load it with previously saved settings in the event of a server crash, or if we want to use many servers with the same configurations and we don't want to rewrite/reclick the same

2 technical informations

3 dynamic values

Dynamic values can be used in tasks, routines to perform the same operations with other values, e.g. displaying the current time in the console, sorting files into folders based on date, iterating through a website for scrapping

3.1 ∼hour∼

Dynamic hour value is replaced with the current time on the server in the format (minute_hour)

3.2 ∼date∼

Dynamic date value is replaced with the current date on the server in the format (day_month_year)

3.3 ∼inc$num∼

Dynamic value inc$ requires an additional value (we replace **num** with a numerical value, e.g. 1), that is converted into a numerical value, and in the next iteration this value is increased by one

3.4 ∼dec$num∼

Dynamic value dec$ requires an additional value (we replace **num** with a numerical value, e.g. 100), that is converted into a numerical value, and in the next iteration this value is decreased by one

3.5 ∼num1 + num2∼

This dynamic value allows **num2** to be added to **num1** every iteration, for example if **num1** is 5 and **num2** is 1 then after the first iteration num1 will be 6, after the second 7 etc. (num1 value will be used as the value in the request, num2 is to inform the program how much to add every iteration)

3.6 ∼num1 - num2∼

This dynamic value allows **num2** to be subtracted from **num1** every iteration, for example if **num1** is 5 and **num2** is 1 then after the first iteration num1 will be 4, after the second 3 etc. (num1 value will be used as the value in the request, num2 is to inform the program how much to substract every iteration)

3.7 ∼files path 0∼

This dynamic value allows you to iterate filenames in the given folder (change *'path'* to the path in the **'upload/'** folder, if you want to list files in this folder, change path to /), the last number in this value is the iteration index, I recommend always setting it as **0** to iterate through the entire folder without skipping files

3.8 ∼lines path 0∼

This dynamic value allows you to iterate through the contents of a file separated by a newline (\n), (change *'path'* to the path to the file in the **'upload/'** folder), the last number is the iteration index, if you want to iterate from the beginning of the file, I recommend leaving the value **0**

3.9 ∼words path 0∼

This dynamic value allows you to iterate through the contents of a file separated by a space, (change *'path'* to the path to the file in the **'upload/'** folder), the last number is the iteration index, if you want to iterate from the beginning of the file, I recommend leaving the value **0**

3.10 dynamic lists

In Home Middleman there is also a dynamic list in which words will be iterated infinitely. The first word in the list after use will be removed from the beginning and added to the end of the list so that it can be used again.
Example usage: ∼{one two three}∼ - in the first iteration only 'one' will be used and the list will become ∼{two three one}∼, then only 'two' will be used and the list will become ∼{three one two}∼ etc.
Of course you can use as many words as you want in the list, it can be 1, it can be 512

3.11 summary

You can use dynamic values in HTTP/HTTPS GET/POST requests, naming folders, files, website query parameters, subdomains, domains, html forms and clipboards, additionally you can use more than one value at a time in a request!

4 examples