The Flametree Web API User's Guide

Contents

1. INTRODUCTION

Welcome to the Flametree Web Application Programming Interface (API) User's Guide. This document contains everything you need to know about calling Flametree Attribution over the Internet.

If you are used to long and complex API manuals, you may find the small size of this one a pleasant surprise. We have designed the Flametree Web API to be as concise and straightforward as possible, without compromising on the flexibility and power of the system in any way.

2. QUICK START

To start using the Web API immediately, write the files that accompany this document (`gringotts_p.csv, gringotts_s.csv, gringotts_y.csv, gringotts_i.csv, gringotts.py, fia.py`) to a directory on your local computer.

Next, open file gringotts.py with an editor. At the start of the file, enter the username and password you have been given into the following fields as shown:

```
username = 'hpotter@hogwarts.com'
password = 'nimbus2000'
```

Save the edit and close the file. Assuming you are using Python version 3, type

```
python gringotts.py,
```

or just

```
gringotts
```

at the command prompt. You should then see the Web API called from your program.

3. WHY A WEB API?

Flametree's Web API allows you to run Flametree Attribution remotely from any location. All you need to access the full power of this industry-leading system is an internet connection, a username and a password, available from our Customer Support desk.

Providing FIA's capabilities through a Web API provides the following advantages:

- No third-party software is installed on your computer network. All you need to run FIA is the ability to put together raw data files and to call the Web API's commands from your own software.
- No specialized or high-powered hardware is required. The Web API allows Flametree to be run from any computer with an Internet connection, even a tablet or a low-powered netbook.
- You always run the most up-to-date version of our software. We run servers in three different time zones to minimize latency and the risk of any downtime.
- Our servers are extremely fast and make use of GPU (graphical processing units) to deliver order of magnitude speed increases over conventional PC architectures. By calling Flametree

via the Web API, you will probably find even the largest calculations only take seconds rather than hours.

4. CALLING THE API

The most straightforward way to use the Web API is to use our Python wrapper routines to interact with the program. For instance, running

```
ListFiles(username, password)
```

in a Python program will return a list of the files available for this user. We assume that the Python wrapper routines are used throughout this document.

If you prefer, you can call the API directly using the Python *requests* (or similar) library from any programming language . Full documentation is provided in Appendix B.

5. ARCHITECTURE AND WORKFLOW

The filestore

In principle, FIA can be run by uploading all required data, together with values of any configuration settings required, and then running the program. For most users, this approach has several disadvantages:

- It requires that the user wait while files are uploaded, the calculation runs, and results downloaded. This ties up resources.
- The same data may have to be uploaded many times – for instance, where the same security master file or benchmark is used for all portfolios.

To avoid these issues, FIA is designed so that file operations are distinct from program execution. FIA uses a server-based filestore, to which raw data files are written, and from which results are read. The program is then run using the following sequence of commands:

- Files required by the program are uploaded to the filestore.
- Once all files are in place, the user sends an 'execute' command, together with parameter values to describe the type of attribution analysis required.
- The API submits this request to a batch queue on the remote server.
- The server runs the attribution analysis as soon as resources become available.
- When the analysis is complete, results and reports are written to the filestore as a compressed file.
- The results file is downloaded.

The batch queue

By using a batch queue for processing, Flametree Attribution allows 'fire and forget' processing for large numbers of portfolios.

The batch queue uses multiple worker processes so that several jobs can be executed at the same time.

A batch processing job has one of the following status values:

- Pending
- Executing
- Completed
- Failed

Security

Each call to the API requires that the user provide credentials identifying themselves as a registered user of the system. This must be either:

- A valid user ID and associated password, or
- A valid security token, generated previously using the *GetToken* command. The security token contains the user ID, so once the token is obtained the user can run the program anonymously.

No user can access any other user's data or usage records. Passwords are stored internally in hashed form so that nobody has access to the plaintext versions. In addition, Flametree Attribution automatically encrypts all traffic to and from the site using HTTPS, so that your identity and results are protected.

6. CALLING THE API

These examples assume:

- that you have been issued a username and password to access the system;
- that you have data files for security master, portfolio weights and returns, and yield curves, called fia_s.csv, fia_p.csv, fia_y.csv respectively;
- that you are using Python, and have a copy of Flametree's fia_wrapper package installed.

The examples shown use our suite of wrapper functions, which encapsulate the complexities of calling the API directly. If you prefer to call the API directly, please refer to Appendix X.

Let's look at the example program as a whole:

from fia_wrapper import *	Import reference to the fia_wapper functions
username = 'user1' password = 'user1'	Assign values for username and password. These are used each time an API function is called.

data = {
"PortfolioFile" : "fia_p.csv",
"SecurityFile" : "fia_s.csv",
"YieldCurveFile" : "fia_y.csv",
"IndexFile" : "fia_i.csv",
"batch_id" : "1001",
"SovereignCurveDecomposition" : "KRD",
"DateFormat" : "%Y-%b-%d",
"SummaryAttributionReport" : "true",
"InteractiveAttributionReport" : "true",
"SecurityAttributionReport" : "true",
"CurveReport" : "false",
"ExPostRiskReport" : "false",
"CSVreport" : "true",
"XLSreport" : "true",
"ZipFile" : "test500.zip"
}
	Assign values to be passed to the API to define the attribution. Some fields (PortfolioFile, SecurityFile,YieldCurveFile, ZipFile) must have values assigned; others are optional.

UploadFile (data['PortfolioFile'], username, password) UploadFile (data['SecurityFile'], username, password) UploadFile (data['YieldCurveFile'], username, password) UploadFile (data['IndexFile'], username, password) UploadFile (data['IndexFile'], username, password)	Upload all required files. If a file is already present in the filestore, it need not be uploaded again.
ListFiles(username, password)	Show the names of all this user's files present in the filestore
job_id = RunJson (data, username, password)	Submit this job to the batch queue. The job is assigned a unique job ID.

while True:
if IsRunning (job_id, username, password) == 'Complete':
break
	Repeatedly interrogate the batch queue to find this job's status. When it has run, its status is 'COMPLETE'.

DownloadFile(data['ZipFile'], username, password)	Now that the job has run, download the results in 'ZipFile'

Uploading files

To upload a file, call the following statements in Python, which issue a 'post' command to the API and some diagnostic data:

```
UploadFile(filename, username, password)
```

If all has gone well, this will display the message

```
{200, "message": "File uploaded OK"}
```

If there is already a file with this name in the filestore, Python will display

```
{403, "message": "Duplicate file name"}
```

If you want to replace a file with another that has the same name, delete the older copy from the filestore by using the commands

```
DeleteFile(filename, username, password)
```

If the file was deleted successfully, you will see the message

```
{200, "message": "File 'fia_s.csv' deleted"}
```

You can now upload a new copy of the file.

To see a list of all files in the filestore, issue the commands

```
ListFiles (username, password)
```

Note that no parameters need be entered. The names of all the files present will be returned as a list in the 'response.text' field.

The same file can be called from multiple runs. For instance, a common security master, or benchmark, can be referenced during multiple runs. In this situation there is no need to upload the file repeatedly.

Uploaded files are subject to the following restrictions:

- File name must be valid: at least N characters, '.' in name, must have csv format
- The API is the only way to access the filestore. The filestore is not accessible by FTP.
- File sizes are restricted to a maximum of 100 MB
- Limit of 1 GB storage per user. Please contact us if you need more remote storage.

Running the program

Once you have uploaded all the files required by FIA, the next step is to set the program's parameters and run the program. This is done by setting up a JSON structure and initializing its values. If you have not used JSON before, don't worry – it's very straightforward and we show you how to do it.

FIA has a long list of parameters that can be set to tailor your analysis. A few must be set, but most are optional and have sensible defaults. For a full list of the parameters and their defaults, refer to Appendix A.

Required fields

The following parameters must be set each time FIA is run. The API will return immediately with a 404 error if any of these quantities has not been set:

Compulsory fields	**Type**
PortfolioFile	String

SecurityFile	String
YieldCurveFile	String
ZipFile	String

In this example we will set the following parameters:

Name	Type	Compulsory?	Value
PortfolioFile	String	Y	fia_p.csv
SecurityFile	String	Y	fia_s.csv
YieldCurveFile	String	Y	fia_y.csv
ZipFile	String	Y	Results
SovereignCurveDecomposition	String	N	STB
batch_id	Integer	N	100
xls_reports	Boolean	N	True

Parameter lists

FIA is run using the execute command. This function requires a JSON structure to be passed as an argument.

In this example, we set up a JSON structure directly. However, many languages allow conversion from other data structures to JSON. For instance, the Python 'json.dumps' function converts a Python dictionary to JSON.

The parameters shown here are represented in JSON as

```
data = {
"PortfolioFile"  : "fia_p.csv",
"SecurityFile"   : "fia_s.csv",
"YieldCurveFile": "fia_y.csv",
“SovereignCurveDecomposition”: “STB”,
“xls_reports”: True,
"batch_id": 100,
“ZipFile” : “DemoZipFile.zip”
}
```

The program is run by issuing the command

job_id = execute(data, username, password)

When the user issues one or more of the above run commands, two things happen:

(i) the job is placed into the API's batch queue for processing. The queue contains the IDs of all jobs that have not yet started execution. If there are no other pending jobs, a job will immediately be executed, in which case it will not appear in the queue at all.

(ii) a unique ID code is returned that uniquely labels each job.

Interrogating the batch queue

Once you have a job's ID code, the *status* command allows you to interrogate the batch queue and return the current status of that job.

This status code can be used with the IsRunning function to interrogate the queue to find the current status of your job. For instance, to run a single job you can use the following commands:

```
id = RunJson (data, username, password)
while True:
    GetQueue(username, password)
    if IsRunning (id, username, password) == 'Complete':
        break
```

7. RETRIEVING RESULTS

Once the program has run, the reports you have requested from the configuration settings will be generated and written to a compressed ZIP file, using the name that you supplied earlier. This ZIP file is written to the filestore and also contains a log file containing information about the run.

To access the reports, use *unzip* or a similar tool.

8. TROUBLESHOOTING

The only requirements to run the examples in this document are that you have Python installed, together with the *requests* and *json* libraries, and a valid username and password. If these are not already installed, please contact your helpdesk or Flametree support for assistance.

APPENDIX A: DIFFERENCES BETWEEN THE LOCAL API AND THE WEB API

If you have used the Flametree API before, you will find the Web version very similar.

There are some small differences between the two API types of which you should be aware. For the Web version

(i) the name of a ZipFile must be supplied. This is the name of the file to which all your reports are written. The ZipFile is written to the filestore and can be downloaded after processing is complete.
(ii) The output directory is not used. If a value is specified, it is ignored.
(iii) The LibDir setting (location of pricing library) is not used. If a value is specified, it is ignored.
(iv) Instead of supplying a file and putting its name in the configuration file, you upload a file to the filestore and assign its name in the parameter list.

Otherwise, the Web API works in exactly the same way as the standard version. In particular, the data dictionary by which run-time parameters are labelled is identical.

APPENDIX B: WRAPPER COMMANDS

General user commands

These commands can be run by any user. Username and password must be added to parameters (if any). If a login token is used, this can be substituted for the username, and the password field can then be left blank.

Command	Description	Arguments	Returns
LatencyTest	Ping the Flametree server	(None)	
ListFiles	Generate list of all files in user's account	(None)	
UploadFile	Upload single file to user's account. If the file already exists, it will be overwritten.	FILE_NAME	Boolean status, indicating whether user account was created successfully. Can include path on host system
DeleteFile	Delete single file from user's account	FILE_NAME	Boolean status, indicating whether file was deleted successfully.
DownloadFile	Download single file from user's account	FILE_NAME	
Execute	Submits job with parameters supplied in JSON structure 'settings' to batch queue	SETTINGS	If successful, returns UUID string job id. This can be used to interrogate the batch queue via the 'Status' command
GetQueue	Shows list of all jobs submitted that have status 'PENDING' (not yet started)	(None)	
GetQueueLength	Shows length of queue.	(None)	
PurgeQueue	Remove all jobs from user's queue	(None)	
Status	Interrogate queue to find status of job with given ID. Return values are 'Pending', 'Running', 'Complete', 'Failed'	JOB_ID	
UserReport	Generates raw system usage report for user (user ID, date and time of call, CPU usage, IP address of caller)		

GetToken	Returns security token that can be used in place of user name and password	(None)	If successful, returns a string token that can be used in place of a username and password.

Administrator commands

These commands can only be run by the system's administrator. The admin password must be added to parameters, if any.

Command	Description	Arguments	Returns
CreateUser	Create new user account, using supplied name and password	NEW_USER_NAME, NEW_PASSWORD	Boolean status, indicating whether user account was created successfully
DeleteUser	Delete existing user account	USER_NAME	True or False, indicating whether user account was deleted successfully
ListUsers	Generate list of all users	(None)	Python list of all user accounts
UsageReport	Generates raw system usage report for all users (user ID, date and time of call, CPU usage, IP address of caller)	(None)	

APPENDIX C: COMMANDS AND STATUS CODES

Flametree Attribution uses REST (Representational State Transfer), an API interface protocol that is rapidly becoming the standard for communication between software components. Commands are issued using a simple set of verbs (GET, POST) and a data dictionary of parameters that can be set by the user.

For the purposes of this tutorial, all you need to know is that all interaction with the API happens through the use of simple verbs (get, post, delete) which are appended to URLs – the paths that tell you which web page to load.

The API returns information about the status of each request in the 'response' object. A response is returned every time the API is called, and this contains both a status code and some text with information about the all.

- A status code of 200 means everything worked OK.
- A status code of 201 means something was created successfully.
- A status code of 402 means a request was unauthorized – for instance, a wrong password was used.
- A value of 403 means a request was forbidden. For instance, the API does not let you overwrite a file with another that has the same name, which is why we got a 403 code in the example above.
- A value of 404 means something was not found. This error can be issued if one of the files FIA needs to run is not present.
- A value of the form 5XX means that the server is not running.

If everything is working correctly, all your status codes will be of the form 2XX. You may wish to check this from within the program that calls FIA.