

# Élaboration de langages de script SS22| Marc-Alexander Richts

---

- Une copie de ce document se trouve en .pdf dans ./docs
- Github Repo : <https://github.com/VIEWVIEWVIEW/Lightstreamer-6.1-python>

## Préface

Dans le cadre du module, nous avons pu choisir un projet personnel qui devait contenir l'application des compétences apprises. J'ai d'écire une bibliothèque pour déterminer en direct les prix des cours des actions.

## Contexte

Les bourses comme le LSE, NYSE, XETRA sont les plus <sup>grandes</sup><sup>1</sup> bourses du monde. Malheureusement, ces bourses ne fournissent pas de prix en temps réel pour les investisseurs privés.

- [LSE ne propose des données de marché qu'aux redistributeurs.](#)
- [NYSE demande 7 500 USD pour l'accès, plus 16 USD par appareil.](#)

Les abonnements aux prix en temps réel des entreprises qui se concentrent sur la revente des données se situent généralement entre 99 et 200 USD par mois :

- [Polygon.io - 200 USD par mois](#)
- [Alpaca - 99 USD par mois](#)

Comme de tels abonnements ne sont généralement pas rentables pour les petits investisseurs, j'ai décidé de trouver une alternative la moins chère possible.

<sup>1</sup>Mesuré par le volume des transactions.

## Définition du problème

[Lang & Schwarz AG](#) est le teneur de marché de nombreux courtiers par les investisseurs de détail. Pour ce faire, L&S publie en permanence les dernières cotes des produits sur son [site web](#). Celles-ci sont publiées sans Le client pourrait par exemple passer des ordres auprès d'un courtier comme TradeRepublic ou la Caisse d'épargne sur ces quotes.

Malheureusement, L&S ne propose pas d'API documentée pour le streaming des prix. L&S utilise "[Lightstreamer 6](#)". Cette librairie est obsolète depuis une demi-décennie et n'est plus utilisée par le fabricant.

n'est pas supporté. Il n'existe pas non plus de client Python pour cette ancienne version héritée, mais seulement un client Java précompilé et un bundle Javascript minifié qui établit une connexion websocket avec L&S directement dans le navigateur.

et met en œuvre le protocole Lightstreamer. Lightstreamer 6 est également complètement non documenté en ce qui concerne les fonctions internes et le protocole.

*Dans l'insouciance de la jeunesse, on se dit "oui, il suffit de regarder dans le Networktab et de les requêtes qui passent par le websocket". Nous y reviendrons plus tard.*

## Instructions d'installation

### Installer Ghostscript

- **Ghostscript** est nécessaire pour télécharger tous les produits négociables. Les produits négociables sont publiés quotidiennement sur **LS-X** sous forme de PDF. Ghostscript permet d'analyser ce PDF afin de produits dans la base de données. Ghostscript est utilisé en interne comme backend de **camelot-py** est utilisé. Camelot extrait les tableaux des PDF.
- **Tkinter** est une dépendance de **camelot-py**

Ubuntu :

```
sudo apt-get install ghostscript python3-tk
```

MacOS :

```
brew install ghostscript tcl-tk
```

Windows Installer :

- Ghostscript : •Tkinter : <https://www.activestate.com/products/tcl/>

### Installer des paquets Python

Il est recommandé de créer un **venv** au préalable ou de travailler dans un conteneur afin de ne pas contaminer l'installation principale. Vous trouverez un exemple de venv dans le repo.

Les paquets suivants devraient être installés (**pip install -r requirements.txt**) :

```
beautifulsoup4==4.11.1
camelot-py==0.9.0
numpy==1.23.2
requests==2.28.1
typing_extensions==4.3.0
urllib3==1.26.11
prisma~=0.6.6
websocket-client~=1.3.3
```

---

Bref aperçu des principales dépendances :

- BeautifulSoup pour charger les ID d'instruments pour Lightstreamer depuis L&S
- Camelot-py pour charger les données d'instruments depuis L&S
- Requêtes pour charger des données d'instrument à partir de L&S
- Prisma agit en tant qu'ORM pour gérer la base de données

- Client Websocket pour se connecter au serveur Lightstreamer de L&S

## Configuration

Si vous ne souhaitez pas utiliser la base de données actuelle, si vous l'avez perdue ou si vous souhaitez passer de SQLite à Postgresql, par exemple, vous pouvez exécuter `prisma db push`. Cela permet de récupérer le schéma de la base de données de

`schema.prisma` est appliqué à la base de données. Si l'on souhaite s'éloigner de SQLite, on peut simplement modifier la `datasource` dans le fichier de schéma. Plus d'informations sur [prisma.io](https://prisma.io)

Maintenant que l'on dispose d'une `database.sqlite3` dans le répertoire racine, on peut exécuter le programme :

```
$ python main.py
```

Un contrôle de mise à jour est ensuite effectué. [L&S suit dans un PDF](#) les actions qui sont respectivement à la sont négociables à la date du jour. Si un nouveau PDF est disponible (il est généralement publié vers 7 heures, heure française), ou si la dernière vérification des ID d'instruments remonte à plus de 24 heures, la mise à jour a été réalisée.

Lors de la mise à jour, le PDF est analysé (cela peut prendre quelques minutes selon le système). L'analyse de 300+ pages prend un peu de temps), puis toutes les informations extraites sont enregistrées dans notre base de données.

écrit.

Ensuite, on peut voir une démo avec les prix actuels de quelques actions liquides, comme Tesla et Porsche. Il faut tenir compte des heures de négoce de la bourse concernée, car seul le prix le plus récent, qui est entré par le flux, est affiché. Les prix historiques ne sont pas disponibles via le Websocket.

Cette démonstration vise simplement à montrer l'utilisation de la bibliothèque. L'utilisateur doit lui-même utiliser cette bibliothèque par la suite, par exemple pour effectuer des analyses techniques et envoyer des ordres au courtier de son choix.

## Mise en place du protocole & douleurs

Lightstreamer utilise en interne d'autres identifiants que ceux qui nous sont fournis par le [fichier Handelsuniversum.pdf](#). Nous donc ce PDF, analysons toutes les données, puis interrogeons le point de fin de recherche de L&S avec l'ISIN extrait du PDF (ici `DE000PAH0038`) :

```
https://www.ls-tc.de/_rpc/json/.lstc/instrument/search/main ? q=DE000PAH0038&localeId=2
```

En réponse, nous recevons une chaîne JSON :

```
[
  {
    "id" : 41519,
    "displayname" : "PORSCHE AUTOM.HLDG VZO",
```

```
    "isin" : "DE000PAH0038",  
    "wkn" : "PAH003",  
    "categoryid" : 5,  
    "productcount" : 19, "alias"  
    : "",  
    "categorysort" : 1,  
    "instrumentId" : 41519,  
    "categorySymbol" : "STK",  
    "categoryName" :  
    "Action", "url" : 41519,  
    "link" : "/fr/action/41519"  
  }  
]
```

Nous enregistrons l'`instrumentId`. La plupart des autres données proviennent du PDF analysé et se trouvent déjà dans notre base de données.

Nous pouvons maintenant commencer à communiquer avec Lightstreamer.

Lightstreamer veut que nous créions une session avec une `phase` de valeur magique.

Je ne peux malheureusement pas expliquer à quoi elle sert, mais Giuseppe Corti, un programmeur de Lightstreamer, [a répondu sur le forum officiel](#) à une question posée :

Les anciennes bibliothèques web utilisaient un protocole privé non documenté spécifique aux langages javascript. Ce n'est donc pas une bonne idée de tenter d'inverser l'ingénierie de ce protocole afin de l'appliquer à d'autres langages.  
Par exemple, la LS\_Phase est un élément très spécifique du protocole javascript utilisé pour reconnaître possible duplicate replies but which do not make sense in other languages.

-- giuseppe.corti, 2 août 2022, 05:12 PM

Pas vraiment utile non plus, car je ne sais toujours pas comment elle est générée exactement. Le bundle Javascript est malheureusement minifié et j'ai passé de nombreuses heures à déduire les noms des variables et des fonctions,  
mais je ne suis pas arrivé jusqu'au générateur de la phase.

Quoi qu'il en soit, le serveur nous déconnecte avec "Unexpected Error" si nous envoyons une phase invalide pour lui, nous ne pouvons donc tester la boîte noire qu'en aveugle. La phase doit être composée de trois à quatre chiffres et se terminer par "3".

Après avoir généré notre phase, nous pouvons aller sur [https://push.lstc.de/lightstreamer/create\\_session.js](https://push.lstc.de/lightstreamer/create_session.js) envoyer un x-www-formencoded les données suivantes :

```
données= {  
  "LS_op2" : "create",  
  "LS_phase" : self.phase, // <-- par exemple 2103  
  "LS_cause" : "new.api",  
  "LS_polling" : "true",  
  "LS_polling_millis" : "0",  
  "LS_idle_millis" : "0",  
  "LS_cid" : "pcYgxn8m8 feOojyA1S681m3g2.pz478mF4Dy", // valeur magique  
  pour identifier le client
```

```
    "LS_adapter_set" : self.adapter_set, "LS_container" :  
    "lsc"  
}
```

Dans ce cas, "WALLSTREETONLINE" est défini comme kit d'adaptation. Le jeu d'adaptateurs a été écrit par L&S et est en fait utilisé comme adaptateur de streaming. Nous analyserons plus tard également les données de cet adaptateur, mais de très triviale.

D'autres paramètres, comme le `LS_container`, décrivent uniquement l'endroit où tous les éléments doivent être montés dans le client Javascript proprement dit. En contrepartie, le serveur nous envoie le JS "over-the-wire" :

```
var myEnv= lsc ; var  
phase= null ;  
function setPhase(ph) {  
    phase= ph ;  
}  
  
function start(sID, addr, kaMs, reqLim, srv, ip) {  
    LS_window.LS_e(1, phase, sID, addr, kaMs, reqLim, srv, ip) ;  
}  
  
function loop(holdMs) {  
    LS_window.LS_e(2, phase, holdMs) ;  
}  
setPhase(9701);start('S7da7fa826970728cT0118850', null, 0, 50000, 'Lightstreamer HTTP  
Server', '10.0.16.33');loop(0) ;
```

Que voyons-nous ici ?

Le serveur nous envoie des JS avec notre phase. Celle-ci ne se termine pas à "3", il y a donc aussi des phases qui ne correspondent pas au modèle que nous avons identifié plus haut. Malheureusement, je ne sais toujours pourquoi il en est ainsi. J'ai aussi essayé de calculer des sommes transversales et de des modèles dans les représentations des octets des phases, mais je n'y arrive pas. Après tout, notre méthode précédente fonctionne...

Et nous voyons un SessionID. Nous l'extrayons simplement avec une belle regex et nous l'enregistrons. Derrière cela, le serveur web divulgue d'ailleurs encore une IP interne, on ne fait pas vraiment ce genre de choses.

---

Bref, nous avons maintenant tout ce qu'il nous faut : Une phase et un identifiant de session ! Il est temps d' le websocket.

Ici, nous devons veiller à indiquer `js.lightstreamer.com` comme sous-protocole.

```
self.ws= websocket.WebSocketApp(self.websocket_url,  
                                subprotocols=self.subprotocols, header=[  
                                    "User-Agent : Mozilla/5.0 (Windows NT  
10.0 ; Win64 ; x64 ; rv:103.0) Gecko/20100101 Firefox/103.0"
```

```
1)
```

Ensuite, nous envoyons notre premier paquet avec son contenu : nous pouvons lier notre websocket à la session que nous venons de créer.

Dans le navigateur, cela à ceci :

```
bind_session
LS_session=S1d3970ef9ad97e22T1132600&LS_phase=1303&LS_cause=loop1&LS_keepalive_mil
lis=5000&LS_container=lsc&``
```

Mais ce que le Networktab ne vous dit pas dans le navigateur, c'est qu'il ne faut pas simplement utiliser le Linux Linebreak `\n` sont utilisés. On obtient alors une déconnexion avec "Unexpected Error". Au lieu de cela, il faut envoyer Carriage-Return comme dans Windows-Fashion :

```
bind\r\nLS_session[..]
```

Je ne l'ai remarqué que lorsque j'ai commencé à compresser tous les paquets avec Wireshark et que je les ai ensuite visualisés avec l'éditeur hexadécimal intégré.

## Problèmes de développement

Points de douleur de Lightstreamer :

- Le serveur nous déconnecte si nous n'indiquons pas "js.lightstreamer.com" comme sous-protocole pour le websocket
- Le serveur nous déconnecte si nous n'indiquons pas Origin "https://www.ls-tc.de".
- Le serveur nous déconnecte si nous ne mettons pas 0001 comme octets à la position 4 - 7 dans la trame du paquet websocket. Apparemment, les navigateurs font cela par défaut. Malheureusement, **Websocket-Client** pour Python ne le fait pas par défaut. D'autres bibliothèques de websocket pour Python ont également été inappropriés et déclenchaient des déconnexions lors de diverses actions. Cependant, **Websocket-Client** a tout gardé à un niveau très bas et n'a pas intégré d'abstractions inutiles.
- Le serveur nous déconnecte si nous utilisons `\n` comme linebreak. Nous devons utiliser `\n\r`.

### Camelot

La bibliothèque "Camelot" est pour analyser les PDF. Camelot tente reconstruire les tableaux à l'intérieur des PDF. Cependant, si les textes trop proches les uns des autres, plusieurs cellules peuvent être considérées à tort comme une seule.

### Exemple

Dans cet exemple, "ZZUH" est ajouté par erreur à la cellule du nom de l'asset, car l'espace entre les deux textes du PDF n'est pas assez grand.

WKN	ISIN	Name	Shortcode
A270EB	CH0508793459	21SHARES SYGNUM MOON	ZZUH
A3GW2D	CH1135202096	21SHARES UNISWAP ETP	2UNI
A2JN55	FR0013341781	2CRSI S.A. EO-,09	52C
A0HL8N	DE000A0HL8N9	2G ENERGY AG	2GB

Le résultat après l'analyse syntaxique est le tableau suivant :

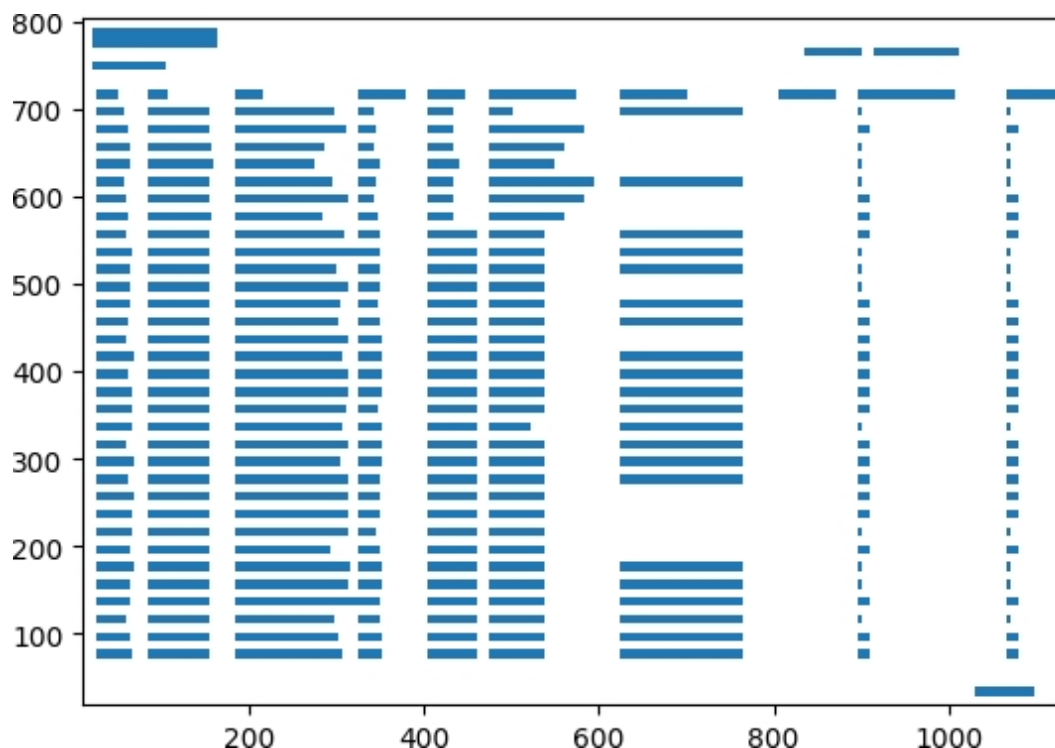
WKN	ISIN	Nom	Code court
A270EB	CH0508793459	21SHARES SYGNUM MOON	ZZUH
A3GW2D	CH1135202096	21SHARES UNISWAP ETP	2UNI
A2JN55	FR0013341781	2CRSI S.A. EO-,09	52C
A0HL8N	DE000A0HL8N9	2G ENERGY AG	2GB

Pour corriger cette erreur, on peut passer le paramètre `columns` à `camelot.read_pdf`.  
`columns` accepte une liste de coordonnées verticales à l'intérieur des PDF : `columns=`  
`['72,95,209,327,442,529,566,606,683']`

Cette liste contient les chaînes des coordonnées verticales pour les séparations de colonnes.

Quirk : on n'utilise pas pour cela une liste bidimensionnelle, mais une liste avec une chaîne de caractères par tableau, dans laquelle les coordonnées sont séparées par des virgules.

Maintenant, tous les textes sont reconnus correctement :



---

Comme les tables paramétrées de Camelot peuvent également contenir l'en-tête, nous filtrons à l'aide d'une regex toutes les lignes qui ne contiennent pas d'ISIN valide.

Un ISIN se compose de 12 caractères. Les deux premiers représentent le pays (CH, DE, US, etc.) et les dix caractères suivants sont un identifiant alphanumérique attribué de manière aléatoire.

Une simple regex suffit dans ce cas :

```
isin_regex= re.compile(r'[A-Z]{2}\w{10}$')
```

Les choses que j'avais prévues, mais que je n'ai pas pu faire.

- Je voulais [utiliser Texualize.io](#) pour un terminal dans le style d'[OpenBB](#). Malheureusement, je n'ai pas eu le temps de le faire, car j'aurais dû refactoriser beaucoup de code.
- Je voulais mesurer l'écart du teneur de marché (différence entre l'offre et la demande). Je suis déçu de ne pas y être parvenu.

## Conclusion

---

Je regrette de ne pas avoir beaucoup parlé de Prisma.io ici, mais le client Prisma-ORM m'a vraiment convaincu. C'est très amusant de l'utiliser, surtout à cause des types qui sont alors Typescriptfashion fournissent une très bonne autocomplétion. Pour le reste : Je n'essaierai probablement plus jamais de lire un bundle minifié qui a été par un chargeur de modules JS comme [AMD](#). L'outillage dans le Le système de codage Python est assez bipolaire : d'un côté, on trouve de super outils comme Camelot-py pour analyser des tableaux dans des PDF, de l'autre, la moitié des clients Websocket que j'ai essayés n'ont que des interfaces très abstraites. Sans accès à l'implémentation interne.