jscrambler Documentation

Release 2.0

Audit Mark

Mar 12, 2017

Contents

Contents:

CHAPTER 1

Quickstart

Introduction

This is a Python module to interface with the JScrambler javascript transformation service. The Python module offers a trivally simple API to transform you javacsript and html files.

Installation

This module has been tested to work in both Python 2.7 and 3.4. The only non-standard Python dependency is the `requests` module.

To install jscrambler, simply install the `jscrambler` package from PyPI. You can use, pip, for instance:

```
$ pip install jscrambler
```

Credentials

To use jscrambler, you need to subscribe to the server and obtain access credentials to use the server. Get your API credentials at https://jscrambler.com/en/account/api_access. There you will see two values, *Access key* and *Secret key*, which allow you to submit projects.

Configuration file

It is easier if you create a json configuration file, such as this one (let's call it `config.json`):

```
{
    // acess credentials, replace with your own
    "keys": {
```

```
        "accessKey": "YOUR_ACCESS_KEY",
        "secretKey": "YOUR_SECRET_KEY"
    },

    // where to find the source .js files
    "filesSrc": ["lib/**/*.js"],

    // directory where to place the modified files
    "filesDest": "build/",

    // parameters that control the transformations available
    "params": {
        "function_outlining": "true",
        "rename_all": "true"
    }
}
```

Minimal python code

Here's some sample python code to process some files, assuming that the output build directory is already created:

```
import jscrambler
import json

# reads the json configuration file
with open("config.json", "rt") as jsonfile:
    config = json.load(jsonfile)

# creates a jscrambler client context
client = jscrambler.Client(config["keys"]["accessKey"],
                           config["keys"]["secretKey"])

# processes the files specified in the configuration
client.process(config)
```

Django integration

Although jscrambler can be integrated with any web framework in any programming language, the jscrambler python package comes with some support for Django projects out of the box.

If you have a Django project that uses the standard `django.contrib.staticfiles` application to support static files, then you already know about the `STATIC_ROOT` django setting. This setting contains the path of a directory to which all the static files will be collected. This is triggered by running the command `python manage.py collectstatic` when you want to deploy new static files into an HTTP server, such as nginx or apache httpd.

To add jscrambler into the workflow, you begin by adding the djcrambler configuration to the Django project settings (*myproject*`/settings.py`):

```
JSCRAMBLER_CONFIG = { my config }
```

The `JSCRAMBLER_CONFIG` setting has to contain dict based structure with the usual jscrambler configuration parameters. If you wish, you can easily load it from an external JSON file, thus:

```
import json
with open("config.json", "r") as configfile:
    JSCRAMBLER_CONFIG = json.load(configfile)
```

Another change in the Django settings that you need is to add `jscrambler` to the `INSTALLED_APPS`:

```
INSTALLED_APPS = (
   'django.contrib.staticfiles',
   #...
   'jscrambler', # <--- add this app to your project
)
```

After these changes, you will get a new Django management command called `scramblestatic`. This command, which should run after `collectstatic`, takes all files matching any of the `filesSrc` patterns from the config, relative to `STATIC_ROOT`, and replace them in-place with the scrambled versions:

```
$ python manage.py collectstatic
$ python manage.py scramblestatic
```

Note: if the config parameter `filesSrc` is missing, it defaults to `**/*.js` and `**/*.html`, which matches all Javascript and HTML files found under `STATIC_ROOT`.

There is no out-of-the-box support for processing Django templates yet, so you should make sure to write your valuable Javascript code that you wish to protect as clearly separated static files, instead of placing it inside Django templates.

Warning: If you have a setup in which the HTTP server is serving static files directly from `STATIC_ROOT`, then running the commands `collectstatic` and `scramblestatic` while the HTTP server is running will temporarily expose your original sources to the Internet. Therefore, it is recommanded that your `STATIC_ROOT` points to a temporary directory, which replaces the live one only after the `scramblestatic` command is finished.

CHAPTER 2

jscrambler.Client: high-level API to interact with jscrambler servers

CHAPTER 3

Configuration

This section describes the recommended configuration format for use with the jscrambler API. The recommendation is to use JSON format, but essencially the API can use just dict-based structures that can be either obtained by parsing a JSON configuration file or constructed programatically.

Configuration File Format

The snippet below illustrates an example configuration file

```
{
  "filesSrc": [""],
  "filesDest": "dist/",
  "host": "api.jscrambler.com", // default
  "port": 443, // default
  "apiVersion": 3, // default
  "keys": {
    "accessKey": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
    "secretKey": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
  },
  "params": {
    "string_splitting": "%DEFAULT%",
    "function_reorder": "%DEFAULT%",
    "function_outlining": "%DEFAULT%",
    "dot_notation_elimination": "%DEFAULT%",
    "expiration_date": "2199-01-01",
    "rename_local": "%DEFAULT%",
    "whitespace": "%DEFAULT%",
    "literal_duplicates": "%DEFAULT%"
  },
  "deleteProject": false // default
}
```

All entries marked with *%DEFAULT%* can be omitted and the client should assume those values. All entries in the "params" section are also optional. The above example only shows a subset of the existing parameters. For a

complete listing of possible parameters, please check here: Optional parameters (though this knowledge shouldn't impact anything on the client implementation).

filesSrc

This configuration entry is a *list of paths* to files that should be included in the project. By *project* we don't mean all the files per

- `["lib/**/*.js"]` should resolve to all JS files inside the lib folder and all the children folders
- `["lib/**"]` should resolve to all files inside the lib folder and all the children folders
- `["lib/*.js"]` should resolve to all JS files directly inside the `lib` folder

Params

For a complete listing of possible parameters, see Optional parameters.

Configuration usage in the API

The configuration is used as follows in the client API:

- In the `jscrambler.Client` constructor, the configuration is not read directly but all the constructor parameters (`accessKey`, `secretKey`, `host`, `port`, and `apiVersion`), can be taken directly from the configuration file;
- The `jscrambler.Client.process()` convenience method takes a configuration file as parameter; the `params` section of the configuration file is used directly in the upload request, and the `filesSrc` and `filesDest` configuration options are used to find the files to upload and the directory where to download the transformed versions, respectively;
- The `filesSrc` parameter is honored by the `scramblestatic` Django management command, see *Django integration*.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search