

---

# **Webstore Manager Documentation**

***Release***

**Martin Melka**

**Feb 26, 2018**



---

## Contents:

---

<b>1</b>	<b>Webstore Manager installation</b>	<b>3</b>
1.1	PyPI . . . . .	3
1.2	GitHub . . . . .	3
<b>2</b>	<b>Webstore Manager usage</b>	<b>5</b>
2.1	Generic arguments . . . . .	5
2.2	Logging . . . . .	5
2.3	Command mode . . . . .	6
2.4	Script mode . . . . .	6
<b>3</b>	<b>Supported platforms</b>	<b>7</b>
3.1	Google Chrome . . . . .	7
3.2	Mozilla Firefox . . . . .	9
<b>4</b>	<b>Source code documentation</b>	<b>11</b>
4.1	chrome_store package . . . . .	11
4.2	firefox_store package . . . . .	11
4.3	script_parser package . . . . .	11
4.4	store package . . . . .	11
4.5	webstore_manager package . . . . .	11
<b>5</b>	<b>Indices and tables</b>	<b>13</b>
	<b>Python Module Index</b>	<b>15</b>



Webstore Manager is a Chrome and Firefox extension manager.

It provides a command-line interface for automatization of tasks such as creating an extension, uploading newer versions and publishing it.



---

## Webstore Manager installation

---

### PyPI

Webstore Manager is distributed via PyPI under name `webstoremgr`.

Run `pip install webstoremgr` to install it.

### GitHub

Sources are publicly available on [GitHub](#).

Instructions about installation from sources can be found in the associated Readme.





---

### Webstore Manager usage

---

There are two basic modes of function:

- **Command mode** A single “target” of the manager is executed. Automatization involving multiple steps needs to be handled by an external script (i.e. bash).
- **Script mode** Targets of the manager tool are specified in a script file. This may involve several steps, variable assignment and more.

The upside of this mode is no presence of sensitive information (auth details) in the terminal history, as all such information may be passed along using environment variables.

Installing Webstore Manager creates an executable script, *webstoremgr*. It serves as a shortcut, it is functionally identical to running *python -m webstore\_manager*. Through the documentation, *webstoremgr* is used.

### Generic arguments

The following arguments are applicable for all running modes:

- **-v - verbose** Increases the level of verbosity. By default only *warn* and more critical messages are logged. This parameter may be repeated (*-vv*) to achieve even more detailed output. See [Logging](#) for details.

### Logging

**Logs are printed to standard output and to a file. The location is platform- and distribution-dependent.**

- **Windows:** %LOCALAPPDATA%\melkamar\webstore\_manager\Logs\
- **Linux: Depending on distribution. Examples:**
  - /var/tmp/webstore\_manager
  - /user/.cache/webstore\_manager/log

You can find the log location by enabling the verbose output.

## Command mode

Commands are invoked on the command line, such as: `webstoremgr chrome create <arguments>`.

List of commands differs based on the target browser. See the platform-specific documentation [here](#).

## Script mode

Webstore Manager's scripting mode consumes a single script file that defines its function. The general invocation syntax is

```
webstoremgr script <filename>
```

where `filename` is the script to execute.

### Syntax

- One command per line.
- Empty lines and lines starting with a hashtag (#) are ignored.
- *Variable assignment*: `ab = cd` assigns value 'cd' into variable 'ab'. Whitespaces around `=` are mandatory, maximum of one `=` sign per line.
- *Variable expansion*: `${ab}` is expanded with the value of `ab`. From the previous example, `${ab}` would equal `cd` when executing the script.
- *Environment variables*: `${env.xyz}` is replaced with the environment variable `xyz`. Example: `${env.PATH}` is resolved to the contents of `$PATH`.
- *Command execution*: `some.func ab cd ef` executes function `some.func` with positional parameters `ab`, `cd` and `ef`. For the list of commands, see below.
- *Example*: this script sets three variables and call two functions with them as parameters.

```
id = ${env.clientid}
secret = ${env.secret}
ref = ${env.reftoken}
chrome.init ${id} ${secret} ${ref}
chrome.setapp abcdef
```

### Generic functions

This is a list of generic functions, not directly tied to any platform. For the list of platform-specific functions, see [Supported platforms](#).

- **cd path** Changes current working dir to `path`.
- **pushd path** Changes current working dir to `path` and saves previous path to internal stack.
- **popd** Return to a dir previously set by `pushd`.
- **zip folder filename** Zips the contents of `folder` and saves the archive as a `filename` in the current working directory.

---

## Supported platforms

---

### Google Chrome

#### Commands

Operations for Chrome are invoked as `$ webstoremgr chrome <command>`. Parameters used in this section are:

- **client\_id, client\_secret**
  - Your client API id obtained in the Google Developers Console. Refer to [using webstore](#).
- **code**
  - One-time code used to generate a refresh token. Obtained through the `init` command.
- **refresh\_token**
  - Reusable token which is used for generating access tokens. It is obtained through the `auth` command.
- **app\_id**
  - ID of an extension as listed in the Dashboard. Click on More Info next to the extension to find out. Its format is e.g. `abcdefghijklmnopqrstuvxyzabcdef`.
- **filename**
  - Name of an extension file (.zip, .crx) on your filesystem.
- **target**
  - Audience for publishing. Two accepted values: `public` and `trusted`.

Supported Chrome Webstore commands are:

- **init Invocation:** `webstoremgr chrome init <client_id>`

Prints information for the user on how to begin authentication. Chrome webstore requires an OAuth authentication which requires opening of the provided link in a web browser whilst signed in to Google.

Opening the provided link will prompt the user with a request for access to their webstore. Upon approving the request, a code is provided. This code may be used to generate a refresh token using the `auth` command.

*Having access to the webstore is necessary for the function of this tool. It does not send any personal information anywhere.*

- **auth Invocation:** `webstoremgr chrome auth <client_id> <client_secret> <code>`

Exchanges your one-time code for a reusable `refresh_token` which can be later used for authentication.

These three parameters should be stored securely, as they grant automated access to your webstore identity.

- **gen-token Invocation:** `webstoremgr chrome gen-token <client_id> <client_secret> <refresh_token>`

Use refresh token to generate an access token. The access token has a limited lifespan (1 hour).

- **create Invocation:** `webstoremgr chrome create [-t,--filetype] <client_id> <client_secret> <refresh_token> <filename>`

Optional parameter `-t` or `--filetype` specifies what type of archive the given file is. Accepted values are `crx` (default) or `zip`.

Create (upload) a new extension to the webstore. It will not be published.

It will be assigned a new `app_id`, this will be printed on the standard output.

- **upload Invocation:** `webstoremgr chrome upload [-t,--filetype] <client_id> <client_secret> <refresh_token> <app_id> <filename>`

Optional parameter `-t` or `--filetype` specifies what type of archive the given file is. Accepted values are `crx` (default) or `zip`.

Upload a new version of an existing extension to the webstore. It will not be published.

- **publish Invocation:** `webstoremgr chrome publish --target <target> <client_id> <client_secret> <refresh_token> <app_id>`

Parameter `target` specifies the audience of users to publish to. Accepted values are `public` or `trusted`.

Publish an extension to a given audience.

- **repack Invocation:** `webstoremgr chrome repack <filename>`

Transform a `crx` archive into a `zip`.

`crx` archive is obtained through Chrome developer tools (pack an extension). When uploading to the Webstore, `zip` is needed.

*This tool accepts both `zip` and `crx` for uploading tasks. This is a convenience method if used in combination with other tools.*

## Script mode

Script mode for Chrome offers all functions of the command line tool. Heading of each list item is an example of how to call the given function in a script. The given parameters correspond to command mode parameters, see section above for details.

- **chrome.init client\_id client\_secret refresh\_token** Initialize the Chrome store. Saves the given parameters as a global state which is used in subsequent steps.

**You must call this function before any others that require authentication.**

- **chrome.setapp app\_id** Set the `app_id` parameter for future method calls.
- **chrome.new filename** Create a new extension from the archive pointed to by `filename`. Calling this function will set the internal `app_id` variable.  
Only accepts ZIP archives. To upload a CRX, you need to run `chrome.unpack` and `zip` functions. See [generic functions](#).
- **chrome.update filename** Update an existing extension. Its ID must be set by calling `chrome.setapp` first. Details are identical to `chrome.new` function.
- **chrome.publish target** Publish an extension to the given target (`public` or `trusted`).  
Its ID must be set by calling `chrome.setapp` first.
- **chrome.check\_version expected\_version timeout** Assertion function to check if the published version is the same as expected.  
The currently published app is compared to the `expected_version` parameter. If they are not equal, the comparison is repeated after several seconds until the `timeout` duration expires. If they are still not equal, script terminates with a nonzero exit code.
- **chrome.unpack archive target\_dir** Unpack a CRX file to the given target directory.

## Mozilla Firefox

### Command mode

Operations for Mozilla Firefox are invoked as `$ webstoremgr firefox <command>`. Parameters used in this section are:

- **id, secret**
  - API key and secret obtained from Mozilla. See [Access credentials](#) for details.
- **filename**
  - Filename of the extension file (.xpi) on your filesystem.
- **addon-id, version**
  - Extension ID and Version. Specified in the `install.rdf` manifest file as `<em:id>` and `<em:version>` fields, respectively. For more information, refer to [Install Manifests](#).

The usecase for Firefox now only supports self-distributed extensions. Mozilla needs to sign such extensions, which is what Webstore Manager offers.

**Supported Firefox addon commands are:**

- **gen-token Invocation:**

```
webstoremgr firefox gen-token --id <id> --secret <secret>
```

Generates a JSON Web Token from the given parameters. This token is used to authenticate for all secured methods. For further details, see [Mozilla API authentication](#).

- **upload Invocation:**

```
webstoremgr firefox upload --id
                             --secret
                             --filename
```

```
[--addon_id]
[--version]
```

Uploads the given extension to Mozilla store for signing. The signing is not done instantaneously, the client is responsible for downloading the file when ready.

Both `addon_id` and `version` parameters are optional. If they are not set, their value will be parsed from the given extension file. If specified, they *must* be the same as the values in manifest file. If the values differ, the task will fail. This may be used as a safeguard that a correct version is being uploaded, but omitting them is generally recommended.

- **download Invocation:**

```
webstoremgr firefox download --id
                             --secret
                             --addon_id
                             --version
                             [--interval]
                             [--attempts]
                             [--folder]
                             [--target-name]
```

Downloads an extension identified by `addon_id` and `version` from the Mozilla store if its processing (verification, signing) is successfully completed.

If the processing is not yet completed, its download will be reattempted `attempts` times with `interval` seconds between each attempt. Default values are 10 attempts in 30-second intervals.

Downloaded file(s) are placed in the current working directory. To override this, set the `--folder` argument.

Optionally, if the extension entry consists of a single file (usual case), supply the `--target-name` parameter to set the name of the downloaded file.

- **sign Invocation:**

```
`webstoremgr firefox sign --id
                           --secret
                           --filename
                           --addon-id
                           --version
                           [--interval]
                           [--attempts]
                           [--folder]
                           [--target-name]`
```

Combines upload and download tasks into a single command. The parameters are directly related to the parameters of commands above, see them for explanation.

## Script mode

Scripting mode is currently not supported for Firefox.

#### **chrome\_store package**

Package containing code to interface with Chrome Webstore. May be used as a library by a third party.

#### **chrome\_store.chrome\_store module**

#### **firefox\_store package**

Package containing code to interface with Mozilla Addon store. May be used as a library by a third party.

#### **firefox\_store.firefox\_store module**

#### **script\_parser package**

Parsing script input to the program.

#### **script\_parser.parser module**

#### **store package**

#### **store.store module**

#### **webstore\_manager package**

Core workflow of the program. Wraps around the individual browser platforms' functionality.

`webstore_manager.constants` module

`webstore_manager.logging_helper` module

`webstore_manager.manager` module

`webstore_manager.util` module



## CHAPTER 5

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### W

`webstore_manager.constants`, [12](#)

## W

`webstore_manager.constants` (module), [12](#)