SEP	21
Title	Add-ons
Author	Pablo Hoffman
Created	2014-02-14
Status	Draft

SEP-021: Add-ons

This proposal introduces add-ons, a unified way to manage Scrapy extensions, middlewares and pipelines.

Scrapy currently supports many hooks and mechanisms for extending its functionality, but no single entry point for enabling and configuring them. Instead, the hooks are spread over:

- Spider middlewares (SPIDER_MIDDLEWARES)
- Downloader middlewares (DOWNLOADER_MIDDLEWARES)
- Downloader handlers (DOWNLOADER_HANDLERS)
- Item pipelines (ITEM_PIPELINES)
- Feed exporters and storages (FEED_EXPORTERS, FEED_STORAGES)
- Overridable components (DUPEFILTER_CLASS, STATS_CLASS, SCHEDULER, SPIDER_MANAGER_CLASS, ITEM_PROCESSOR, etc)
- Generic extensions (EXTENSIONS)
- CLI commands (COMMANDS_MODULE)

One problem of this approach is that enabling an extension often requires modifying many settings, often in a coordinated way, which is complex and error prone. Add-ons are meant to fix this by providing a simple mechanism for enabling extensions.

Design goals and non-goals

Goals:

- simple to manage: adding or removing extensions should be just a matter of adding or removing lines in a `scrapy.cfg` file
- backward compatibility with enabling extension the "old way" (i.e. modifying settings directly)

Non-goals:

- a way to publish, distribute or discover extensions (use pypi for that)

Managing add-ons

Add-ons are defined in the `scrapy.cfg` file, inside the `[addons]` section.

To enable the "httpcache" addon, either shipped with Scrapy or in the Python search path, create an entry for it in your `scrapy.cfg`, like this:

```
[addons]
httpcache =
```

You may also specify the full path to an add-on (which may be either a .py file or a folder containing __init__.py):

```
[addons]
mongodb_pipeline = /path/to/mongodb_pipeline.py
```

Writing add-ons

Add-ons are Python modules that implement the following callbacks.

addon_configure

Receives the Settings object and modifies it to enable the required components. If it raises an exception, Scrapy will print it and exit.

Examples:

```
def addon_configure(settings):
    settings.overrides['DOWNLADER_MIDDLEWARES'].update({
        'scrapy.contrib.downloadermiddleware.httpcache.HttpCacheMiddleware': 900,
    })
```

```
def addon_configure(settings):
    try:
        import boto
    except ImportError:
        raise RuntimeError("boto library is required")
```

crawler_ready

`crawler_ready` receives a Crawler object after it has been initialized and is meant to be used to perform post-initialization checks

like making sure the extension and its dependencies were configured properly. If it raises an exception, Scrapy will print and exit.

Examples:

```
def crawler_ready(crawler):
    if 'some.other.addon' not in crawler.extensions.enabled:
        raise RuntimeError("Some other addon is required to use this addon")
```