SEP	9
Title	Singleton removal
Author	Pablo Hoffman
Created	2009-11-14
Status	Document in progress (being written)

SEP-009 - Singletons removal

This SEP proposes a refactoring of the Scrapy to get ri of singletons, which will result in a cleaner API and will allow us to implement the library API proposed in :doc:`sep-004`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\sep\(scrapy-master)(sep)sep-009.rst, line 13); *backlink***
>
> Unknown interpreted text role "doc".

Current singletons

Scrapy 0.7 has the following singletons:

- Execution engine (`scrapy.core.engine.scrapyengine`)
- Execution manager (`scrapy.core.manager.scrapymanager`)
- Extension manager (`scrapy.extension.extensions`)
- Spider manager (`scrapy.spider.spiders`)
- Stats collector (`scrapy.stats.stats`)
- Logging system (`scrapy.log`)
- Signals system (`scrapy.xlib.pydispatcher`)

Proposed API

The proposed architecture is to have one "root" object called `Crawler` (which will replace the current Execution Manager) and make all current singletons members of that object, as explained below:

- **crawler**: `scrapy.crawler.Crawler` instance (replaces current `scrapy.core.manager.ExecutionManager`) - instantiated with a `Settings` object
    - **crawler.settings**: `scrapy.conf.Settings` instance (passed in the `__init__` method)
    - **crawler.extensions**: `scrapy.extension.ExtensionManager` instance
    - **crawler.engine**: `scrapy.core.engine.ExecutionEngine` instance
        - `crawler.engine.scheduler`
            - `crawler.engine.scheduler.middleware` - to access scheduler middleware
        - `crawler.engine.downloader`
            - `crawler.engine.downloader.middleware` - to access downloader middleware
        - `crawler.engine.scraper`
            - `crawler.engine.scraper.spidermw` - to access spider middleware
    - **crawler.spiders**: `SpiderManager` instance (concrete class given in `SPIDER_MANAGER_CLASS` setting)
    - **crawler.stats**: `StatsCollector` instance (concrete class given in `STATS_CLASS` setting)
    - **crawler.log**: Logger class with methods replacing the current `scrapy.log` functions. Logging would be started (if enabled) on `Crawler` instantiation, so no log starting functions are required.
        - `crawler.log.msg`
    - **crawler.signals**: signal handling
        - `crawler.signals.send()` - same as `pydispatch.dispatcher.send()`
        - `crawler.signals.connect()` - same as `pydispatch.dispatcher.connect()`
        - `crawler.signals.disconnect()` - same as `pydispatch.dispatcher.disconnect()`

Required code changes after singletons removal

All components (extensions, middlewares, etc) will receive this `Crawler` object in their `__init__` methods, and this will be the only mechanism for accessing any other components (as opposed to importing each singleton from their respective module). This will also serve to stabilize the core API, something which we haven't documented so far (partly because of this).

So, for a typical middleware `__init__` method code, instead of this:

```
#!python
from scrapy.core.exceptions import NotConfigured
from scrapy.conf import settings

class SomeMiddleware(object):
   def __init__(self):
      if not settings.getbool('SOMEMIDDLEWARE_ENABLED'):
          raise NotConfigured
```

We'd write this:

```
#!python
from scrapy.core.exceptions import NotConfigured

class SomeMiddleware(object):
   def __init__(self, crawler):
      if not crawler.settings.getbool('SOMEMIDDLEWARE_ENABLED'):
          raise NotConfigured
```

Running from command line

When running from **command line** (the only mechanism supported so far) the `scrapy.command.cmdline` module will:

1. instantiate a `Settings` object and populate it with the values in SCRAPY_SETTINGS_MODULE, and per-command overrides
2. instantiate a `Crawler` object with the `Settings` object (the `Crawler` instantiates all its components based on the given settings)
3. run `Crawler.crawl()` with the URLs or domains passed in the command line

Using Scrapy as a library

When using Scrapy with the **library API**, the programmer will:

1. instantiate a `Settings` object (which only has the defaults settings, by default) and override the desired settings
2. instantiate a `Crawler` object with the `Settings` object

Open issues to resolve

- Should we pass `Settings` object to `ScrapyCommand.add_options()`?
- How should spiders access settings?
    - Option 1. Pass `Crawler` object to spider `__init__` methods too
        - pro: one way to access all components (settings and signals being the most relevant to spiders)
        - con?: spider code can access (and control) any crawler component - since we don't want to support spiders messing with the crawler (write an extension or spider middleware if you need that)
    - Option 2. Pass `Settings` object to spider `__init__` methods, which would then be accessed through `self.settings`, like logging which is accessed through `self.log`
        - con: would need a way to access stats too