SEP	19
Title	Per-spider settings
Author	Pablo Hoffman, NicolÃ¡s Ramirez, Julia Medina
Created	2013-03-07
Status	Final (implemented with minor variations)

SEP-019: Per-spider settings and Crawl Process Cleanup

This is a proposal to add support for overriding settings per-spiders in a consistent way, while taking the chance to refactor the settings population and whole crawl workflow.

In short, you will be able to overwrite settings (on a per-spider basis) by implementing a class method in your spider:

```
class MySpider(Spider):

    @classmethod
    def custom_settings(cls):
        return {
            "DOWNLOAD_DELAY": 5.0,
            "RETRY_ENABLED": False,
        }
```

Proposed changes

- new `custom_settings` class method will be added to spiders, to give them a chance to override settings *before* they're used to instantiate the crawler
- spider managers will maintain loading spider classes functionality (with a new `load` method that will return a spider class given its name), but spider initialization will be delegated to crawlers (with a new `from_crawler` class method in spiders, that will allow them access to crawlers directly)
- spider manager will be striped out of Crawler class, as it will no longer need it
- `SPIDER_MODULES` and `SPIDER_MANAGER_CLASS` settings will be removed and replaced by entries on `scrapy.cfg`. Thus spider managers won't need project settings to configure themselves
- CrawlerProcess will be remove, since crawlers will be created independently with a required spider class and optional `SettingsReader` instance
- Settings class will be split into two classes: `SettingsLoader` and `SettingsReader`, and a new concept of "setting priority" will be added

Settings

Settings class will be split into two classes `SettingsLoader` and `SettingsReader`. First one will be used to settle all different levels of settings across the project, and the later will be a frozen version of the already loaded settings and will be the preferred way to access them. This will avoid the current possible misconception that you can change settings after they have been populated. There'll be a new concept of settings priorities, and `settings.overrides` will be deprecated in favor of explicitly loaded settings with priorities, as it'll make the settings overriding not order-dependent.

Because of this, `CrawlerSettings` (with its overrides, settings_module and defaults) will be remove, but its interface could be maintained for backward compatibility in `SettingsReader` (since on `SettingsLoader`, overrides dictionary and settings with priorities don't get along with a consistent implementation). Maintaining this attributes and their functionality is not advisable since it breaks the read-only access nature of the class.

With the new per-spider settings, there's a need of a helper function that will take a spider and return a `SettingsReader` instance populated with defaults', project's and the given spider's settings. Motive behind this is that `get_project_settings` can't continue being used for getting settings instance for crawler usage when using the API directly, as the project is not the only source of settings anymore. `get_projects_settings` will become an internal function because of that.

SettingsLoader

`SettingsLoader` is going to populate settings at startup, then it'll be converted to a `SettingsReader` instance and discarded afterwards.

It is supposed to be write-only, but many previously loaded settings are needed to be accessed before freezing them. For example, the `COMMANDS_MODULE` setting allows loading more command default settings. Another example is that we need to read `LOG_*` settings early because we must be able to log errors on the load settings process. `ScrapyCommands` may be configured based upon current settings, as users can plug custom commands. These are some of the reasons that suggest that we need a read-write access for this class.

- Will have a method `set(name, value, priority)` to register a setting with a given priority. A `setdict(dict, priority)` method may come handy for loading project's and per-spider settings.
- Will have current Settings getter functions (`get`, `getint`, `getfloat`, `getdict`, etc.) (See above for reasons behind this).
- Will have a `freeze` method that returns an instance of `SettingsReader`, with a copy of the current state of settings (already prioritized).

SettingsReader

It's intended to be the one used by core, extensions, and all components that use settings without modifying them. Because there are logical objects that change settings, such as `ScrapyCommands`, use cases of each settings class need to be comprehensively explained.

New crawlers will be created with an instance of this class (The one returned by the `freeze` method on the already populated `SettingsReader`), because they are not expected to alter the settings.

It'll be read-only, keeping the same getter methods of current `Settings` (`get`, `getint`, `getfloat`, `getdict`, etc.). There could be a `set` method that will throw a descriptive explanatory error for debugging compatibility, avoiding its inadvertently usage.

Setting priorities

There will be 5 setting priorities used by default:

- 0: global defaults (those in `scrapy.settings.default_settings`)
- 10: per-command defaults (for example, shell runs with `KEEP_ALIVE=True`)
- 20: project settings (those in `settings.py`)
- 30: per-spider settings (those returned by `Spider.custom_settings` class method)
- 40: command line arguments (those passed in the command line)

There are a couple of issues here:

- `SCRAPY_PICKLED_SETTINGS_TO_OVERRIDE` and `SCRAPY_{settings}` need-to-be deprecated environment variables: Can be kept, with a new or existing priority.
- We could have different priorities for settings given with the `-s` argument and other named arguments in the command line (For example, `-s LOG_ENABLE=False --loglevel=ERROR` will set `LOG_ENABLE` to True, because named options are overridden later in the current implementation), but because the processing of command line options is done in one place we could leave them with the same priority and depend on the order of the set calls just for this case.

Deprecated code

`scrapy.conf.settings` singleton is a deprecated implementation concerning settings load. Could be maintained as it is, but the singleton should implement new `SettingsReader` interface in order to work.

Spider manager

Currently, the spider manager is part of the crawler which creates a cyclic loop between settings and spiders and it shouldn't belong there. The spiders should be loaded outside and passed to the crawler object, which will require a spider class to be instantiated.

This new spider manager will not have access to the settings (they won't be loaded yet) so it will use scrapy.cfg to configure itself.

The `scrapy.cfg` would look like this:

```
[settings]
default = myproject.settings

[spiders]
manager = scrapy.spidermanager.SpiderManager
modules = myproject.spiders
```

- `manager` replaces `SPIDER_MANAGER_CLASS` setting and will, if omitted, default to `scrapy.spidermanager.SpiderManager`
- `modules` replaces `SPIDER_MODULES` setting and will be required

These ideas translate to the following changes on the `SpiderManager` class:

- `__init__(spider_modules)` -> `__init__()`. `spider_modules` will be looked in `scrapy.cfg`.
- `create('spider_name', **spider_kargs)` -> `load('spider_name')`. This will return a spider class, not an instance. It's basically a `__get__` to `self._spiders`.
- All remaining functions should be deprecated or remove accordingly, since a crawler reference is no longer needed.
- New helper `get_spider_manager_class_from_scrapycfg` in `scrapy/utils/spidermanager.py`.

Spiders

A new class method `custom_settings` is proposed, that could be use to override project and default settings before they're used to instantiate the crawler:

```
class MySpider(Spider):

    @classmethod
    def custom_settings(cls):
        return {
            "DOWNLOAD_DELAY": 5.0,
```

```
            "RETRY_ENABLED": False,
        }
```

This will only involve a `set` call with the corresponding priority when populating `SettingsLoader`.

Contributing to API changes, new `from_crawler` class method will be added to spiders to give them a chance to access settings, stats, or the crawler core components themselves. This should be the new way to create a spider from now on (instead of normally instantiate it as is currently).

Scrapy commands

As already stated, `ScrapyCommands` modify the settings, so they need a `SettingsLoader` instance reference in order to do that.

Present `process_option` implementations on Base and other commands read and override settings. These overrides should be changed to `set` calls with the allocated priority.

Each command with a custom `run` method should be modified to reflect the new refactored API (Particularly `crawl` command).

CrawlerProcess

`CrawlerProcess` should be remove because Scrapy crawl command no longer supports running multiple spiders. The preferred way for doing this is using the API manually, instantiating a separate Crawler for each spider, so `CrawlerProcess` has loosen its utility.

This change is not directly related to the project (it's not focus on settings but it fits in the API clean up task), but it's a great opportunity to consider since we're changing the crawling startup flow.

This class will be deleted and the attributes and methods will be merge with `Crawler`. For that effect, these are the specific merges and removals:

- `self.crawlers` doesn't make sense is this new set up, each reference will be replace with self.
- `create_crawler` will be `__init__` of `Crawler`
- `_start_crawler` will be merge with `Crawler.start`
- `start` will be merge with `Crawler.crawl` but this will need from the later an extra boolean parameter `start_reactor` (default: True) to crawl with or without starting twisted reactor (This is required in `commands.shell` in order to start the reactor in another thread).

Startup process

This summarizes the current and new proposed mechanisms for starting up a Scrapy crawler. Imports and non representative functions are omitted for brevity.

Current (old) startup process

```
# execute in cmdline

# loads settings.py, returns CrawlerSettings(settings_module)
settings = get_project_settings()
settings.defaults.update(cmd.default_settings)

cmd.crawler_process = CrawlerProcess(settings)
cmd.run # (In a _run_print_help call)

    # Command.run in commands/crawl.py

    self.crawler_process.create_crawler()
    spider = crawler.spiders.create(spider_name, **spider_kwargs)
crawler.crawl(spider)
    self.crawler_process.start() # starts crawling spider

        # CrawlerProcess._start_crawler in crawler.py

        crawler.configure()
```

Proposed (new) startup process

```
# execute in cmdline

smcls = get_spider_manager_class_from_scrapycfg()
sm = smcls() # loads spiders from module defined in scrapy.cfg
spidercls = sm.load(spider_name) # returns spider class, not instance

settings = get_project_settings() # loads settings.py
settings.setdict(cmd.default_settings, priority=40)

settings.setdict(spidercls.custom_settings(), priority=30)
```

```
settings = settings.freeze()
cmd.crawler = Crawler(spidercls, settings=settings)

    # Crawler.__init__ in crawler.py

    self.configure()

cmd.run # (In a _run_print_help call)

    # Command.run in commands/crawl.py

    self.crawler.crawl(**spider_kwargs)

        # Crawler.crawl in crawler.py

        spider = self.spidercls.from_crawler(self, **spider_kwargs)
    # starts crawling spider
```