Flask-Registry Documentation

Release 0.2.0

CERN

June 27, 2014

Contents

Flask-Registry is an extension for Flask that allow frameworks to dynamically assemble your Flask application from reusable packages consisting of blueprints, extensions, and configuration.

This part of the documentation will show you how to get started in using Flask-Registry with Flask.

CHAPTER 1

Installation

Install Flask-Registry with `pip`

```
pip install flask-registry
```

The development version can be downloaded from its page at GitHub.

```
git clone https://github.com/inveniosoftware/flask-registry.git
cd flask-registry
python setup.py develop
source run-tests.sh
```

1.1 Requirements

Flask-Registry has the following dependencies:

- Flask
- six

Flask-Registry requires Python version 2.6, 2.7 or 3.3+

CHAPTER 2

Quickstart

This guide assumes you have successfully installed Flask-Registry and a working understanding of Flask. If not, follow the installation steps and read about Flask at http://flask.pocoo.org/docs/.

2.1 A Minimal Example

A minimal Flask-Registry usage example looks like this. First create the application and initialize the extension:

```
>>> from flask import Flask
>>> from flask_registry import Registry
>>> from flask_registry import ListRegistry
>>> app = Flask('myapp')
>>> r = Registry(app=app)
```

Then, we can create a simple ListRegistry that just keeps a list of objects:

```
>>> r['my_namespace'] = ListRegistry()
>>> r['my_namespace'].register("something")
>>> r['my_namespace'].register("something else")
>>> for obj in r['my_namespace']:
...     print(obj)
something
something else
```

2.2 Application Discovery Example

Flask-Registry also has support for dynamically discovering Python modules, resources, entry points and the like. All this can be put together in your Flask application factory to create and easily extensible application.

Following is a small example how a Flask application can be assemble from reusable packages that each provides configuration, extensions and blueprints:

```
from flask import Flask
from flask_registry import Registry, PackageRegistry, ExtensionRegistry, \
    ConfigurationRegistry, BlueprintAutoDiscoveryRegistry


class Config(object):
    PACKAGES = ['tests']
    EXTENSIONS = ['tests.mockext']
```

```
    USER_CFG = True


def create_app(config):
    app = Flask('myapp')
    app.config.from_object(config)
    r = Registry(app=app)
    r['packages'] = PackageRegistry(app)
    r['extensions'] = ExtensionRegistry(app)
    r['config'] = ConfigurationRegistry(app)
    r['blueprints'] = BlueprintAutoDiscoveryRegistry(app=app)
    return app

if __name__ == '__main__':
    config = Config()
    app = create_app(config)
    app.run(debug=True)
```

Save this in a file named `app.py` next to the `tests` folder in the Flask-Registry distribution and run it using your Python interpreter.

```
$ python app.py
 * Running on http://127.0.0.1:5000/
$ curl http://localhost:5000
Hello from Flask-Registry
```

The blueprint is loaded from `tests.views` and only works if the extension `tests.mockext` and the configuration in `tests.config` has been loaded.

See *Application Discovery* for full explanation on what is happening in the example. Flask extension.

Flask-Registry is initialized like this:

```
>>> from flask import Flask
>>> from flask_registry import Registry, ListRegistry
>>> app = Flask('myapp')
>>> r = Registry(app=app)
```

A simple usage example of `ListRegistry` looks like this:

```
>>> app.extensions['registry']['my.namespace'] = ListRegistry()
>>> len(app.extensions['registry'])
1
>>> app.extensions['registry']['my.namespace'].register("something")
>>> app.extensions['registry']['my.namespace'].register("something else")
>>> len(app.extensions['registry']['my.namespace'])
2
>>> for obj in app.extensions['registry']['my.namespace']:
...     print(obj)
something
something else
```

CHAPTER 3

Module Discovery

The module discovery registries provide discovery functionality useful for searching a list of Python packages for a specific module name, and afterwards registering the module. This is used to e.g. load and register Flask blueprints by `BlueprintAutoDiscoveryRegistry`.

Assume e.g. we want to discover the `helpers` module from the `tests` package. First we initialize the registry:

```
>>> from flask import Flask
>>> from flask_registry import Registry, ModuleDiscoveryRegistry
>>> from flask_registry import ImportPathRegistry
>>> app = Flask('myapp')
>>> r = Registry(app=app)
```

We then create the list of packages to search through using an `ImportPathRegistry`:

```
>>> r['mypackages'] = ImportPathRegistry(initial=['tests'])
```

Then, initialize the `ModuleDiscoveryRegistry` and run the discovery:

```
>>> r['mydiscoveredmodules'] = ModuleDiscoveryRegistry(
...     'helpers', registry_namespace='mypackages')
>>> len(r['mydiscoveredmodules'])
0
>>> r['mydiscoveredmodules'].discover(app=app)
>>> len(r['mydiscoveredmodules'])
1
```

3.1 Lazy discovery

Using `RegistryProxy` you may lazily discover modules. Above example using lazy loading looks like this:

```
>>> from flask_registry import RegistryProxy
>>> app = Flask('myapp')
>>> r = Registry(app=app)
>>> pkg_proxy = RegistryProxy('mypackages', ImportPathRegistry,
...     initial=['tests'])
>>> mod_proxy = RegistryProxy('mydiscoveredmodules',
...     ModuleDiscoveryRegistry,
...     'helpers',
...     registry_namespace=pkg_proxy)
>>> 'mypackages' in r
False
```

```
>>> 'mydiscoveredmodules' in r
False
>>> with app.app_context():
...     mod_proxy.discover(app=app)
>>> 'mypackages' in r
True
>>> 'mydiscoveredmodules' in r
True
```

CHAPTER 4

Application Discovery

The application discovery registries provide discovery functionality useful for dynamically constructing Flask applications based on configuration variables. This allows a developer to package config, blueprints and extensions into isolated and reusable packages which a framework can dynamically install into a Flask application.

Such a package (named `tests`) could look like:

- `tests.views` – contains blueprints which should be registered on the application object.
- `tests.mockext` – contains a `setup_app()` method which be used to install any Flask extensions on the application object.
- `tests.config` – contains configuration variables specific for this module.

Following is a simplified example of a Flask application factory, that will load config, extensions and blueprints:

```
>>> from flask import Flask, Blueprint
>>> from flask_registry import Registry, PackageRegistry
>>> from flask_registry import ExtensionRegistry
>>> from flask_registry import ConfigurationRegistry
>>> from flask_registry import BlueprintAutoDiscoveryRegistry
>>> class Config(object):
...     PACKAGES = ['tests']
...     EXTENSIONS = ['tests.mockext']
...     USER_CFG = True
>>> def create_app(config):
...     app = Flask('myapp')
...     app.config.from_object(config)
...     r = Registry(app=app)
...     r['packages'] = PackageRegistry(app)
...     r['extensions'] = ExtensionRegistry(app)
...     r['config'] = ConfigurationRegistry(app)
...     r['blueprints'] = BlueprintAutoDiscoveryRegistry(app=app)
...     return app
>>> config = Config()
>>> app = create_app(config)
```

4.1 Packages

The config variable `PACKAGES` specifies the list of Python packages, which `ConfigurationRegistry` and `BlueprintAutoDiscoveryRegistry` will search for `config.py` and `views.py` modules inside.

```
>>> for pkg in app.extensions['registry']['packages']:
...     print(pkg)
tests
```

4.2 Extensions

The config variable `EXTENSIONS` specifies the list of Python packages, which the `ExtensionRegistry` will load and call `setup_app(app)` on, to dynamically initialize Flask extensions.

```
>>> for pkg in app.extensions['registry']['extensions']:
...     print(pkg)
tests.mockext
```

4.3 Configuration

The `ConfigurationRegistry` will merge any package defined config, with the application config without overwriting already set variables in the application config:

```
>>> config.USER_CFG
True
>>> import tests.config
>>> tests.config.USER_CFG
False
>>> app.config['USER_CFG']
True
```

4.4 Blueprints

The `BlueprintAutoDiscoveryRegistry` will search for blueprints defined inside a `views` module in each package defined in `PACAKGES`. It will also register the discovered blueprints on the Flask application. Each `views` module should define either a single blueprint in the variable `blueprint` and/or multiple blueprints in the variable `blueprints`:

```
>>> from tests import views
>>> isinstance(views.blueprint, Blueprint)
True
>>> len(views.blueprints)
2
>>> for k in sorted(app.blueprints.keys()):
...     print(k)
test
test1
test2
```

CHAPTER 5

Package Resources

Package resource registries may be used to discover e.g. package resources as well as loading entry points.

5.1 Entry points

`setuptools` entry points are a simple way for packages to "advertise" Python objects, so that frameworks can search for these entry points. `setup.py` files for instance allows you to specify `console_scripts` entry points, which will install scripts into system path for you.

The `EntryPointRegistry` allows you to easily register these entry points into your Flask application:

```
>>> from flask import Flask
>>> from flask_registry import Registry, EntryPointRegistry
>>> app = Flask('myapp')
>>> r = Registry(app=app)
>>> r['scripts'] = EntryPointRegistry('console_scripts')
>>> 'easy_install' in r['scripts']
True
```

Entry points are specified in you `setup.py`, e.g.:

```
setup(
    # ...
    entry_points={
        'flask_registry.test_entry': [
            'testcase = flask_registry:RegistryBase',
        ]
    },
    # ...
)
```

```
>>> r['entrypoints'] = EntryPointRegistry(
...     'flask_registry.test_entry', load=True)
>>> 'testcase' in r['entrypoints']
True
>>> from flask_registry import RegistryBase
>>> r['entrypoints']['testcase'][0] == RegistryBase
True
```

See http://pythonhosted.org/setuptools/pkg_resources.html#entry-points for more information on entry points.

5.2 Resource files

The `PkgResourcesDirDiscoveryRegistry` will search a list of Python packages for a specific resource directory and register all files found in the directories.

Assume e.g. a package `tests` have a directory `resources` with one file in it called `testresource.cfg`. This file can be discovered in the following manner:

```
>>> import os
>>> app = Flask('myapp')
>>> r = Registry(app=app)
>>> from flask_registry import ImportPathRegistry
>>> from flask_registry import PkgResourcesDirDiscoveryRegistry
>>> r['packages'] = ImportPathRegistry(initial=['tests'])
>>> r['res'] = PkgResourcesDirDiscoveryRegistry('resources', app=app)
>>> os.path.basename(r['res'][0]) == 'testresource.cfg'
True
```

CHAPTER 6

Extending Flask-Registry

You can easily create your own type of registries by subclassing one of the existing registries found in the modules under `flask_registry.registries`.

If you for instance want to create a list registry that only accepts integers, you could create it like this:

```
>>> from flask import Flask
>>> from flask_registry import Registry, RegistryError, ListRegistry
>>> class IntListRegistry(ListRegistry):
...     def register(self, item):
...         if not isinstance(item, int):
...             raise ValueError("Object must be of type int")
>>> app = Flask('myapp')
>>> r = Registry(app=app)
>>> r['myns'] = IntListRegistry()
>>> r['myns'].register(1)
>>> r['myns'].register("some string")
Traceback (most recent call last):
  File "/usr/lib/python2.7/doctest.py", line 1289, in __run
    compileflags, 1) in test.globs
  File "<doctest default[7]>", line 1, in <module>
    r['myns'].register("some string")
  File "<doctest default[2]>", line 4, in register
    raise ValueError("Object must be of type int")
ValueError: Object must be of type int
```

CHAPTER 7

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

7.1 API Docs

Flask extension.

Flask-Registry is initialized like this:

```
>>> from flask import Flask
>>> from flask_registry import Registry, ListRegistry
>>> app = Flask('myapp')
>>> r = Registry(app=app)
```

A simple usage example of `ListRegistry` looks like this:

```
>>> app.extensions['registry']['my.namespace'] = ListRegistry()
>>> len(app.extensions['registry'])
1
>>> app.extensions['registry']['my.namespace'].register("something")
>>> app.extensions['registry']['my.namespace'].register("something else")
>>> len(app.extensions['registry']['my.namespace'])
2
>>> for obj in app.extensions['registry']['my.namespace']:
...     print(obj)
something
something else
```

class `flask_registry.`**`Registry`**(*app=None*)

Flask extension.

Initialization of the extension:

```
>>> from flask import Flask
>>> from flask_registry import Registry
>>> app = Flask('myapp')
>>> r = Registry(app)
>>> app.extensions['registry']
<Registry ()>
```

or alternatively using the factory pattern:

```
>>> app = Flask('myapp')
>>> r = Registry()
>>> r.init_app(app)
>>> r
<Registry ()>
```

init_app(*app*)
Initialize a Flask application.

Only one Registry per application is allowed.

Parameters **app** (*flask.Flask*) – Flask application

Raises RegistryError if the registry is already initialized

class flask_registry.**RegistryProxy**(*namespace*, *registry_class*, **args*, ***kwargs*)
Lazy proxy object to a registry in the current_app

Allows you to define a registry in your local module without needing to initialize it first. Once accessed the first time, the registry will be initialized in the current_app, thus you must be working in either the Flask application context or request context.

```
>>> from flask import Flask
>>> app = Flask('myapp')
>>> from flask_registry import Registry, RegistryProxy, RegistryBase
>>> r = Registry(app=app)
>>> proxy = RegistryProxy('myns', RegistryBase)
>>> 'myns' in app.extensions['registry']
False
>>> with app.app_context():
...     print(proxy.namespace)
...
myns
>>> 'myns' in app.extensions['registry']
True
```

Parameters

- **namespace** – Namespace for registry
- **registry_class** – The registry class - i.e. a sublcass of RegistryBase.
- **args** – Arguments passed to registry_class on initialization.
- **kwargs** – Keyword arguments passed to registry_class on initialization.

class flask_registry.**RegistryBase**
Abstract base class for all registries.

Each subclass must implement the register() method. Each subclass may implement the unregister() method.

Once a registry is registered in the Flask application, the namespace under which it is available is injected into it self.

Please see flask_registry.registries.core for simple examples of subclasses.

namespace
Namespace. Used only by the Flask extension to inject the namespace under which this instance is registered in the Flask application. Defaults to None if not registered in a Flask application.

register(**args*, ***kwargs*)
Abstract method which MUST be overwritten by subclasses. A subclass does not need to take the same number of arguments as the abstract base class.

unregister(**args*, ***kwargs*)
Abstract method which MAY be overwritten by subclasses. A subclass does not need to take the same number of arguments as the abstract base class.

class flask_registry.**RegistryError**
Exception class raised for user errors.

e.g. creating two registries in the same namespace)

7.1.1 Core Registries

The core registries are useful to use as subclasses for other more advanced registries. The provide the basic functionality for `list` and `dict` style registries, as well as simple import path and module style registries.

class flask_registry.registries.core.**ListRegistry**
Basic registry that just keeps a list of objects. Provides normal list-style access to the registry:

```
>>> from flask import Flask
>>> from flask_registry import Registry, ListRegistry
>>> app = Flask('myapp')
>>> r = Registry(app=app)
>>> r['myns'] = ListRegistry()
>>> r['myns'].register("something")
>>> len(r['myns'])
1
>>> r['myns'][0]
'something'
>>> "something" in r['myns']
True
>>> for obj in r['myns']:
...     print(obj)
something
```

register(*item*)
Register a new object

Parameters **item** – Object to register

unregister(*item*)
Unregister an existing object. Raises a `ValueError` in case object does not exists. If the same object was registered twice, only the first registered object will be unregister.

Parameters **item** – Object to unregister

class flask_registry.registries.core.**DictRegistry**
Basic registry that just keeps a key, value pairs. Provides normal dict-style access to the registry:

```
>>> from flask import Flask
>>> from flask_registry import Registry, DictRegistry
>>> app = Flask('myapp')
>>> r = Registry(app=app)
>>> r['myns'] = DictRegistry()
>>> r['myns'].register("mykey", "something")
>>> len(r['myns'])
1
>>> r['myns']["mykey"]
```

```
'something'
>>> "mykey" in r['myns']
True
>>> for k, v in r['myns'].items():
...     print("%s: %s" % (k,v))
mykey: something
```

register(*key*, *value*)
Register a new object under a given key.

Parameters

- **key** – Key to register object under
- **item** – Object to register

unregister(*key*)
Unregister an object under a given key. Raises `KeyError` in case the given key doesn't exists.

class `flask_registry.registries.core.`**SingletonRegistry**
Basic registry that just keeps a single object.

```
>>> from flask import Flask
>>> from flask_registry import Registry, SingletonRegistry
>>> app = Flask('myapp')
>>> r = Registry(app=app)
>>> r['singleton'] = SingletonRegistry()
>>> r['singleton'].register("test string")
>>> r['singleton'].get()
'test string'
>>> r['singleton'].register("another string")
Traceback (most recent call last):
    ...
RegistryError: Object already registered.
>>> r['singleton'].unregister()
>>> r['singleton'].get() is None
True
>>> r['singleton'].unregister()
Traceback (most recent call last):
    ...
RegistryError: No object to unregister.
```

get()
Get the registered object

register(*obj*)
Register a new singleton object

Parameters **obj** – The object to register

unregister()
Unregister the singleton object

class `flask_registry.registries.core.`**ImportPathRegistry**(*initial=None*, *exclude=None*, *load_modules=False*)
Registry of Python import paths. Supports simple discovery of modules without loading them.

```
>>> from flask import Flask
>>> from flask_registry import Registry, ImportPathRegistry
>>> app = Flask('myapp')
>>> r = Registry(app=app)
>>> r['myns'] = ImportPathRegistry(initial=[
```

```
... 'flask_registry.registries.*',
... 'flask_registry'])
>>> for imp_path in r['myns']:
...     print(imp_path)
flask_registry.registries.appdiscovery
flask_registry.registries.core
flask_registry.registries.moduleDiscovery
flask_registry.registries.pkgresources
flask_registry
```

When using star imports it is sometimes useful to exclude certain imports:

```
>>> r['myns2'] = ImportPathRegistry(
... initial=['flask_registry.registries.*',     ],
... exclude=['flask_registry.registries.core']
... )
>>> for imp_path in r['myns2']:
...     print(imp_path)
flask_registry.registries.appdiscovery
flask_registry.registries.moduleDiscovery
flask_registry.registries.pkgresources
```

Parameters

- **initial** – List of initial import paths.
- **exclude** – A list of import paths to not register. Useful together with star imports (`'*'`). Defaults to `[]`.
- **load_modules** – Load the modules instead of just registering the import path. Defaults to `False`.

register(*import_path*)
Register a new import path

Parameters **import_path** – A full Python import path (e.g. `somepackge.somemodule`) or Python star import path to find all modules inside a package (e.g. `somepackge.*`).

unregister(**args*, ***kwargs*)
It is not possible to unregister import paths.

class `flask_registry.registries.core.`**ModuleRegistry**(*with_setup=True*)
Registry for Python modules with setup and teardown functionality.

Each module may provide a `setup()` and `teardown()` function which will be called when the module is registered. The name of the methods can be customized by subclassing and setting the class attributes `setup_func_name` and `teardown_func_name`.

Any extra arguments and keyword arguments to `register` and `unregister` is passed to the setup and teardown functions.

Example:

```
import mod

registry = ModuleRegistry(with_setup=True)
registry.register(mod, arg1, arg2, kw1=...)
# Will call mod.setup(arg1, arg2, kw1=...)
```

Parameters **with_setup** – Call setup/teardown function when registering/unregistering modules. Defaults to `True`.

register(*module*, **args*, ***kwargs*)

Parameters

- **module** – Module to register.
- **args** – Argument passed to the module setup function.
- **kwargs** – Keyword argument passed to the module setup function.

setup_func_name = 'setup'
Name of setup function. Defaults to `setup`.

teardown_func_name = 'teardown'
Name of teardown function. Defaults to `teardown`.

unregister(*module*, **args*, ***kwargs*)

Parameters

- **module** – Module to unregister.
- **args** – Argument passed to the module teardown function.
- **kwargs** – Keyword argument passed to the module teardown function.

7.1.2 Application Discovery

The application discovery registries provide discovery functionality useful for dynamically constructing Flask applications based on configuration variables. This allows a developer to package config, blueprints and extensions into isolated and reusable packages which a framework can dynamically install into a Flask application.

Such a package (named `tests`) could look like:

- `tests.views` – contains blueprints which should be registered on the application object.
- `tests.mockext` – contains a `setup_app()` method which be used to install any Flask extensions on the application object.
- `tests.config` – contains configuration variables specific for this module.

Following is a simplified example of a Flask application factory, that will load config, extensions and blueprints:

```
>>> from flask import Flask, Blueprint
>>> from flask_registry import Registry, PackageRegistry
>>> from flask_registry import ExtensionRegistry
>>> from flask_registry import ConfigurationRegistry
>>> from flask_registry import BlueprintAutoDiscoveryRegistry
>>> class Config(object):
...     PACKAGES = ['tests']
...     EXTENSIONS = ['tests.mockext']
...     USER_CFG = True
>>> def create_app(config):
...     app = Flask('myapp')
...     app.config.from_object(config)
...     r = Registry(app=app)
...     r['packages'] = PackageRegistry(app)
...     r['extensions'] = ExtensionRegistry(app)
...     r['config'] = ConfigurationRegistry(app)
...     r['blueprints'] = BlueprintAutoDiscoveryRegistry(app=app)
```

```
...     return app
>>> config = Config()
>>> app = create_app(config)
```

Packages

The config variable `PACKAGES` specifies the list of Python packages, which `ConfigurationRegistry` and `BlueprintAutoDiscoveryRegistry` will search for `config.py` and `views.py` modules inside.

```
>>> for pkg in app.extensions['registry']['packages']:
...     print(pkg)
tests
```

Extensions

The config variable `EXTENSIONS` specifies the list of Python packages, which the `ExtensionRegistry` will load and call `setup_app(app)` on, to dynamically initialize Flask extensions.

```
>>> for pkg in app.extensions['registry']['extensions']:
...     print(pkg)
tests.mockext
```

Configuration

The `ConfigurationRegistry` will merge any package defined config, with the application config without overwriting already set variables in the application config:

```
>>> config.USER_CFG
True
>>> import tests.config
>>> tests.config.USER_CFG
False
>>> app.config['USER_CFG']
True
```

Blueprints

The `BlueprintAutoDiscoveryRegistry` will search for blueprints defined inside a `views` module in each package defined in `PACAKGES`. It will also register the discovered blueprints on the Flask application. Each `views` module should define either a single blueprint in the variable `blueprint` and/or multiple blueprints in the variable `blueprints`:

```
>>> from tests import views
>>> isinstance(views.blueprint, Blueprint)
True
>>> len(views.blueprints)
2
>>> for k in sorted(app.blueprints.keys()):
...     print(k)
test
test1
test2
```

class flask_registry.registries.appdiscovery.**PackageRegistry**(*app*)
Specialized ImportPathRegistry that takes the initial list of import paths from the PACKAGES configuration variable in the application.

Parameters **app** – The Flask application object from which includes a PACAKGES variable in it's configuration.

class flask_registry.registries.appdiscovery.**ExtensionRegistry**(*app*)
Flask extensions registry (Specialized ListRegistry). Loads all extensions specified by EXTENSIONS configuration variable. The registry will look for a setup_app function in the extension and call it if it exists.

Example configuration:

```
EXTENSIONS = [
    'invenio.ext.debug_toolbar',
    'invenio.ext.menu:MenuAlchemy',
]
```

Parameters **app** – Flask application to get configuration from.

register(*app*, *ext_name*)
Register a Flask extensions and call setup_app() on it.

Parameters

- **app** – Flask application object
- **ext_name** – An import path (e.g. a package, module, object) which when loaded has an method setup_app().

unregister()
It is not possible to unregister configuration.

class flask_registry.registries.appdiscovery.**ConfigurationRegistry**(*app*, *registry_namespace=None*)
Specialized ModuleDiscoveryRegistry that search for config modules in a list of Python packages and merge them into the Flask application config without overwriting already set variables.

Parameters

- **app** – A Flask application
- **registry_namespace** – The registry namespace of an ImportPathRegistry with a list Python packages to search for config modules in. Defaults to packages.

register(*new_object*)
Register a new config module.

Parameters **new_object** – The configuration module. app.config.from_object() will be called on it.

unregister(**args*, ***kwargs*)
It is not possible to unregister configuration.

class flask_registry.registries.appdiscovery.**BlueprintAutoDiscoveryRegistry**(*module_name=None*, *app=None*, *with_setup=False*, *silent=False*)
Specialized ModuleAutoDiscoveryRegistry that search for views modules in a list of Python packages and register blueprints found inside them.

Blueprints are loaded by searching for a variable blueprints (list of Blueprint instances) or blueprint (a Blueprint instance). If found, the blueprint will be registered on the Flask application.

A blueprint URL prefix can be overwritten using the `BLUEPRINTS_URL_PREFIXES` variable in the application configuration:

```
BLUEPRINTS_URL_PREFIXES = {
    '<blueprint name>': '<new url prefix>',
    # ...
}
```

7.1.3 Module Discovery

The module discovery registries provide discovery functionality useful for searching a list of Python packages for a specific module name, and afterwards registering the module. This is used to e.g. load and register Flask blueprints by `BlueprintAutoDiscoveryRegistry`.

Assume e.g. we want to discover the `helpers` module from the `tests` package. First we initialize the registry:

```
>>> from flask import Flask
>>> from flask_registry import Registry, ModuleDiscoveryRegistry
>>> from flask_registry import ImportPathRegistry
>>> app = Flask('myapp')
>>> r = Registry(app=app)
```

We then create the list of packages to search through using an `ImportPathRegistry`:

```
>>> r['mypackages'] = ImportPathRegistry(initial=['tests'])
```

Then, initialize the `ModuleDiscoveryRegistry` and run the discovery:

```
>>> r['mydiscoveredmodules'] = ModuleDiscoveryRegistry(
...     'helpers', registry_namespace='mypackages')
>>> len(r['mydiscoveredmodules'])
0
>>> r['mydiscoveredmodules'].discover(app=app)
>>> len(r['mydiscoveredmodules'])
1
```

Lazy discovery

Using `RegistryProxy` you may lazily discover modules. Above example using lazy loading looks like this:

```
>>> from flask_registry import RegistryProxy
>>> app = Flask('myapp')
>>> r = Registry(app=app)
>>> pkg_proxy = RegistryProxy('mypackages', ImportPathRegistry,
...     initial=['tests'])
>>> mod_proxy = RegistryProxy('mydiscoveredmodules',
...     ModuleDiscoveryRegistry,
...     'helpers',
...     registry_namespace=pkg_proxy)
>>> 'mypackages' in r
False
>>> 'mydiscoveredmodules' in r
False
>>> with app.app_context():
...     mod_proxy.discover(app=app)
>>> 'mypackages' in r
True
```

```
>>> 'mydiscoveredmodules' in r
True
```

class flask_registry.registries.moduledicovery.**ModuleDiscoveryRegistry**(*module_name*, *registry_namespace=None*, *with_setup=False*, *silent=False*)

Specialized ModuleRegistry that will search a list of Python packages in an ImportPathRegistry or ModuleRegistry for a specific module name. By default the list of Python packages is read from the packages registry namespace.

Packages may be excluded during the discovery using a configuration variables constructed according to the following pattern:

```
<NAMESPACE>_<MODULE_NAME>_EXCLUDE
```

where <NAMESPACE> should be replaced by the registry_namepsace, and <MOUDLE_NAME> should be replaced with module_name. Example: PACKAGES_VIEWS_EXCLUDE. All namespaces are capitalized and have dots replaced with underscores.

Subclasses of ModuleDiscoveryRegistry may overwrite the internal _discover_module() method to provide specialized discovery (see e.g. BlueprintAutoDiscoveryRegistry).

Parameters

- **module_name** – Name of module to search for in packages.
- **registry_namespace** – The registry namespace of an ImportPathRegistry or ModuleRegistry with a list Python packages to search for module_name modules in. Alternatively to a registry namespace an instance of a RegistryProxy or Registry may also be used. Defaults to packages.
- **with_setup** – Call setup and teardown function on discovered modules. Defaults to False (see ModuleRegistry).
- **silent** – if set to True import errors are ignored. Defaults to False.

discover(*app=None*)

Perform module discovery, by iterating over the list of Python packages in the order they are specified.

Parameters **app** – Flask application object from where the list of Python packages is loaded (from the registry_namespace). Defaults to current_app if not specified (thus requires you are working in the Flask application context).

class flask_registry.registries.moduledicovery.**ModuleAutoDiscoveryRegistry**(*module_name*, *app=None*, *registry_namespace=None*, *with_setup=False*, *silent=False*)

Specialized ModuleDiscoveryRegistry that will discover modules immediately on initialization.

Parameters

- **module_name** – Name of module to search for in packages.
- **app** – Flask application object
- **registry_namespace** – The registry namespace of an ImportPathRegistry or ModuleRegistry with a list Python packages to search for module_name modules in.

Alternatively to a registry namespace an instance of a `RegistryProxy` or `Registry` may also be used. Defaults to `packages`.

- **with_setup** – Call setup and teardown function on discovered modules. Defaults to `False` (see `ModuleRegistry`).
- **silent** – if set to True import errors are ignored. Defaults to `False`.

7.1.4 Package Resources

Package resource registries may be used to discover e.g. package resources as well as loading entry points.

Entry points

`setuptools` entry points are a simple way for packages to "advertise" Python objects, so that frameworks can search for these entry points. `setup.py` files for instance allows you to specify `console_scripts` entry points, which will install scripts into system path for you.

The `EntryPointRegistry` allows you to easily register these entry points into your Flask application:

```
>>> from flask import Flask
>>> from flask_registry import Registry, EntryPointRegistry
>>> app = Flask('myapp')
>>> r = Registry(app=app)
>>> r['scripts'] = EntryPointRegistry('console_scripts')
>>> 'easy_install' in r['scripts']
True
```

Entry points are specified in you `setup.py`, e.g.:

```
setup(
    # ...
    entry_points={
        'flask_registry.test_entry': [
            'testcase = flask_registry:RegistryBase',
        ]
    },
    # ...
)
```

```
>>> r['entrypoints'] = EntryPointRegistry(
...     'flask_registry.test_entry', load=True)
>>> 'testcase' in r['entrypoints']
True
>>> from flask_registry import RegistryBase
>>> r['entrypoints']['testcase'][0] == RegistryBase
True
```

See http://pythonhosted.org/setuptools/pkg_resources.html#entry-points for more information on entry points.

Resource files

The `PkgResourcesDirDiscoveryRegistry` will search a list of Python packages for a specific resource directory and register all files found in the directories.

Assume e.g. a package `tests` have a directory `resources` with one file in it called `testresource.cfg`. This file can be discovered in the following manner:

```
>>> import os
>>> app = Flask('myapp')
>>> r = Registry(app=app)
>>> from flask_registry import ImportPathRegistry
>>> from flask_registry import PkgResourcesDirDiscoveryRegistry
>>> r['packages'] = ImportPathRegistry(initial=['tests'])
>>> r['res'] = PkgResourcesDirDiscoveryRegistry('resources', app=app)
>>> os.path.basename(r['res'][0]) == 'testresource.cfg'
True
```

class flask_registry.registries.pkgresources.**EntryPointRegistry**(*entry_point_ns*, *load=True*)

Entry point registry. Based on `DictRegistry` with keys being the entry point group, and the value being a list of objects referenced by the entry points.

Parameters

- **entry_point_ns** – Entry point namespace
- **load** – if False, entry point will not be loaded. Defaults to `True`.

register(*entry_point*)

Register a new entry point

Parameters **entry_point** – The entry point

class flask_registry.registries.pkgresources.**PkgResourcesDirDiscoveryRegistry**(*module_name*, *app=None*, *registry_namespace=None*, *with_setup=False*, *silent=False*)

Specialized `ModuleAutoDiscoveryRegistry` that will search a list of Python packages in an `ImportPathRegistry` or `ModuleRegistry` for a specific resource directory and register all files found in the directories. By default the list of Python packages is read from the `packages` registry namespace.

CHAPTER 8

Additional Notes

Notes on how to contribute, legal information and changelog are here for the interested.

8.1 Contributing

See <http://invenio-software.org/wiki/Development/Contributing> for now.

8.2 Changelog

Here you can see the full list of changes between each Flask-Registry release.

8.2.1 Version 0.2.0 (released 2014-06-27)

- ListRegistry now fuly behaves as a list.
- DictRegistry now fuly behaves as a dict.
- Fixes issue with app in ModuleAutoDiscoveryRegistry.
- Excludes option for ImportPathRegistry.
- Fixes handling of missing package resource directory.
- Fixes issue in configuration loading.
- Allows removal of registries.
- Fixes ImportError and SyntaxError handling.
- Documentation and code coverage improvements.
- Differentiates between missing and broken modules.
- New BlueprintAutoDiscoveryRegistry.
- New SingletonRegistry.

8.2.2 Version 0.1

- Initial public release

8.3 License

The full license text can be found below (*GNU General Public License*).

8.3.1 Authors

Flask-Registry is developed for use in Invenio digital library software.

Contact us at info@invenio-software.org

Contributors

- Lars Holm Nielsen <lars.holm.nielsen@cern.ch>
- Jiri Kuncar <jiri.kuncar@cern.ch>
- Esteban J. G. Gabancho <esteban.jose.garcia.gabancho@cern.ch>
- Tibor Simko <tibor.simko@cern.ch>
- Yoan Blanc <yoan@dosimple.ch>

8.3.2 GNU General Public License

GNU GENERAL PUBLIC LICENSE Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your

freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not

price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid

anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether

gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and

(2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain

that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software

patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and

modification follow.

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains

a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's

source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion

of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it,

under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program

except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received

copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not

signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the

Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent

infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in

certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions

of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free

programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY

FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING

WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest

possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest

to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> <signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

f