Flask-Via

Release 2014.05.19

May 19, 2014

Inspired by the Django URL configuration system, `Flask-Via` is designed to add similar functionality to Flask applications which have grown beyond a simple single file application.

CHAPTER 1

Example

```
from flask import Flask
from flask.ext.via import Via
from flask.ext.via.routers.default import Functional

app = Flask(__name__)

def foo(bar=None):
    return 'Foo View!'

routes = [
    Functional('/foo', foo),
    Functional('/foo/<bar>', foo, endpoint='foo2'),
]

via = Via()
via.init_app(app, route_module='flask_via.examples.basic')

if __name__ == "__main__":
    app.run(debug=True)
```

CHAPTER 2

Why?

Growing your application can be quite difficult when it's not always clear where and how your routes are discovered. This can lead to a cluttered application factory method when all your routes are defined at application creation - resulting in code which is difficult to maintain, not to mention messy.

A better solution is to define your routes in a `routes.py` and automatically load them at application start up. This is what `Flask-Via` helps to do.

Third party Flask extensions don't always follow the same conventions for adding routes to an application, so `Flask-Via` has been designed to be easy for developers to write their own custom routers. For an example of this, take a look at the bundled `Flask-Restful` Resource router.

If you do write a custom router that is useful to you, it will probably be useful to someone else so please do contribute back :)

CHAPTER 3

Links

- Documentation: http://flask-via.thisissoon.com
- CI: https://travis-ci.org/thisissoon/Flask-Via
- Coverage: https://coveralls.io/r/thisissoon/Flask-Via?branch=master

CHAPTER 4

Usage Documentation

4.1 Quickstart

4.1.1 Installation

`Flask-Via` is simple to install, just use your favourite python package manage, for example `pip`:

```
$ pip install Flask-Via
```

4.1.2 Basic Application

Once we have installed `Flask-Via` we need to perform the following steps:

1. Create some view functions
2. Create a list of routes
3. Initialise `flask_via.Via` and call `flask_via.Via.init_app()`

The following example code performs the above steps with key lines emphasised.

```
from flask import Flask
from flask.ext.via import Via
from flask.ext.via.routers.default import Functional

app = Flask(__name__)

def foo(bar=None):
    return 'Foo View!'

routes = [
    Functional('/foo', foo),
    Functional('/foo/<bar>', foo, endpoint='foo2'),
]

via = Via()
via.init_app(app, route_module='path.to.here')

if __name__ == "__main__":
    app.run(debug=True)
```

Lines 10-13 show how routes are defined in a list using the basic flask router class (flask_via.routers.default.Functional).

Line 16 shows how we Flask-Via looks for where routes are defined, this can be set as we have done above or using the VIA_ROUTES_MODULE application configuration variable.

4.2 Configuration

The following configuration variables can be set in your flask application config.

VIA_ROUTES_MODULE	This should be a string with the value of a python dotted path to your root module which contains routes, e.g: VIA_ROUTES_MODULE = 'yourapp.routes'
VIA_ROUTES_NAME	By default Via will look for a variable called routes within the routes module, if you want to call it something different then use this config variable, e.g: VIA_ROUTES_NAME = 'urls'

4.3 Route Discovery

Routes can live anywhere you want them too, as long as they are importable.

You can tell Flask-Via where to find routes in a couple of ways:

1. VIA_ROUTES_MODULE application configuration variable
2. routes_module argument passed into init_app in your application factory method.

You can use which ever you prefer.

4.3.1 Using Application Config

```
from flask import Flask
from flask.ext.via import Via

app = Flask(__name__)
app.config['VIA_ROUTES_MODULE'] = 'yourapp.routes'

via = Via()
via.init_app(app)

if __name__ == "__main__":
    app.run(debug=True)
```

4.3.2 Using init_app setting routes_module

```
from flask import Flask
from flask.ext.via import Via

app = Flask(__name__)
```

```
via = Via()
via.init_app(app, routes_module='yourapp.routes')

if __name__ == "__main__":
    app.run(debug=True)
```

4.3.3 Route Module

The routes module should define a `list` of routes, by default this list is called `routes`:

```
routes = [
    Functional('/', home),
    Functional('/about', about),
]
```

You can configure `Flask-Via` to look for any variable name of your choosing, this is done by passing an argument named `routes_name` into `init_app`, for example:

```
via = Via()
via.init_app(app, routes_name='urls')
```

You can also make this setting permanent by using the `VIA_ROUTES_NAME` configuration variable:

```
app = Flask(__name__)
app.config['VIA_ROUTES_MODULE'] = 'yourapp.routes'
app.config['VIA_ROUTES_NAME'] = 'urls'

via = Via()
via.init_app(app)
```

Note: If you set `VIA_ROUTES_NAME` overriding this using `routes_name` is still possible however this does not propagate over any routes which are included.

4.3.4 Application Example

Assume we have the following application structure:

```
/path/to/foo
    - __init__.py
    - routes.py
    - views.py
    - app.py
```

Within `views.py` we have:

```
def home():
    return 'Hello world!'

def about():
    return 'The world is big'
```

Within `routes.py` we have:

```
from flask.ext.via.routers import default
```

```
urls = [
    default.Functional('/', home),
    default.Functional('/about', about),
]
```

Within `app.py` we have:

```
from flask import Flask
from flask.ext.via import Via

app = Flask(__name__)
app.config['VIA_ROUTES_MODULE'] = 'foo.routes'

via = Via()
via.init_app(app, routes_name='urls')

if __name__ == "__main__":
    app.run(debug=True)
```

You will see we used `routes_name` when calling `via.init_app` to tell `Via` what variable to look for within the routes module.

4.4 Routers

Here you will find the documentation for each bundled router provided by `Flask-Via`.

4.4.1 Flask Routers

These routers are designed to work with standard flask functional and class based pluggable views.

Functional Router

The `flask_via.routers.default.Functional` router handles basic functional based view routing.

Arguments:

- `url`: The url for this route, e.g: `/foo`
- `func`: The view function

Keyword Arguments:

- `endpoint`: (Optional) A custom endpoint name, by default flask uses the view function name.

Example

```
from flask.ext.via.routers.default import Functional

def foo(bar=None):
    return 'foo'

routes = [
    Functional('/', foo),
```

```
    Functional('/<bar>', foo, endpoint='foobar'),
]
```

Pluggable Router

The `flask_via.routers.default.Pluggable` router handles views created using Flasks pluggable views.

Arguments:

- `url`: The url for this route, e.g: `/foo`
- `class`: The Flask Pluggable View Class
- `name`: The name of the view, aka: endpoint

Keyword Arguments:

- `**kwargs`: Arbitrary keyword arguments, for example `methods`

Example

```
from flask.views import MethodView
from flask.ext.via.routers.default import Pluggable

class FooView(MethodView):

    def get(self, bar=None):
        return 'foo'

routes = [
    Plugganle('/', FooView, 'foo'),
    Plugganle('/<bar>', FooView, 'foobar'),
]
```

4.4.2 `Flask-Restful` Routers

`Flask-Restful` is an awesome framework for building REST API's in Flask but has it's own way of adding routes to the Flask application, so tere is a little bit of extra work required when bootstrapping your application:

```
from flask import Flask
from flask.ext import restful
from flask.ext.via import Via

app = Flask(__name__)
api = restful.Api(app)

via = Via()
via.init_app(
    app,
    routes_module='yourapp.routes',
    restful_api=api)

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

Note that on line 12 we passed a keyword argument called `restful_api` with the value being the `Flask-Restful` api object into `via.init_app`. This will allow the `flask_via.routers.restful.Resource` router to add resouce routes to the api.

Resouce Router

> **Warning:** Before using this router be sure you have read the section directly above.

The `flask_via.routers.restful.Resource` router allows us to register `Flask-Restful` resources to your application.

Arguments:

- `url`: The url for this route, e.g: `/foo`
- `resource`: A `Flask-Restful` `Resource` class

Keyword Arguments:

- `endpoint`: (Optional) A custom endpoint name

Example

```
class FooResource(restful.Resource):

    def get(self, bar=None):
        return {'hello': 'world'}

routes = [
    Resource('/', FooResource)
    Resource('/<bar>', FooResource, endpoint='foobar')
]
```

4.4.3 `Flask-Admin` Routers

As with the `Flask-Restful` router you need to pass an extra argument to `via.init_app` called `flask_admin` which should hold the `Flask-Admin` instance.

```
1  from flask import Flask
2  from flask.ext.admin import Admin
3  from flask.ext.via import Via
4
5  app = Flask(__name__)
6
7  admin = Admin(name='Admin')
8  admin.init_app(app)
9
10 via = Via()
11 via.init_app(
12     app,
13     routes_module='flask_via.examples.admin',
14     flask_admin=admin)
15
16 if __name__ == '__main__':
17     app.run(debug=True)
```

Note that line `14` is where the instantiated `Flask-Admin` instance gets passed into `via.init_app`.

Admin Router

Warning: Before using this router be sure you have read the section directly above.

The `flask_via.routers.admin.AdminRoute` router allows us to register `Flask-Admin` views to your application. `Flask-Admin` handles defining urls for its views so a `url` argument is not requied, all is required is the `Flask-Admin` view class.

Arguments:

- `view`: An instantiated `Flask-Admin` view

Example

```
class FooAdminView(BaseView):

    @expose('/')
    def index(self):
        return 'foo'

    @expose('/bar')
    def index(self):
        return 'bar'


routes = [
    AdminRoute(FooAdminView(name='Foo'))
]
```

4.5 Including

Sometimes you don't want to define all your routes in one place, you want to be modular right!? You can do that too with `Flask-Via`.

4.5.1 Include Router

The most basic way of including other routes is to use the `flask_via.routers.Include` router. This is not a intended replacement or implementation of Flask blueprints, just a simple way of putting routes somewhere else in your application.

Arguments:

- `routes_module`: Python dotted path to the route module as a string.

Keyword Arguments:

- `routes_name`: (Optional) If you have not called the list of routes in the moduke `routes` you can set that here, for example `urls`.
- `url_prefix`: (optional) Add a url prefix to all routes included
- `endpoint`: (optional) Add a endpoint prefix to all routes included

Example

Assume the following application structure:

```
/path/to/foo
    - bar/
        - __init__.py
        - routes.py
        - views.py
    - __int__.py
    - routes.py
```

In the top level `routes.py` we would have:

```
from flask.ext.via.routers import Include

routes = [
    Include('foo.bar.routes')
]
```

In the `foo.routes` we would have:

```
from flask.ext.via.routes import default
from foo.bar.views import some_view

routes = [
    default.Functional('/bar', some_view)
]
```

You can see this in action with the Small Application Example.

URL Prefixes

The `flask_via.routers.Include` class also allows you to add a `url_prefix` similar to blueprints.

The following routers support the `url_prefix` being passed to their `add_to_app` methods:

- `flask_via.routers.default.Functional`
- `flask_via.routers.default.Pluggable`
- `flask_via.routers.default.Blueprint`

Example

Assume the same application structure as in the above examples except the top level `routes.py` now looks like this:

```
from flask.ext.via.routers import Include

routes = [
    Include('foo.bar.routes', url_prefix='/foo')
]
```

This will result in the url to the view becoming `/foo/bar` instead of `/bar`.

Endpoints

The `flask_via.routers.Include` router also allows you to add endpoint prefixes to your included routes, much like blueprints. This is supported by:

- `flask_via.routers.default.Functional`
- `flask_via.routers.default.Pluggable`
- `flask_via.routers.default.Blueprint`

Example

We will assume the same application structure as we have in the previous example applications. The top level `routes.py` can be altered as followes:

```
from flask.ext.via.routers import Include

routes = [
    Include('foo.bar.routes', url_prefix='/foo', endpoint='foo')
]
```

We can now call `url_for` with `foo.bar` which would generate `/foo/bar`.

4.5.2 Blueprint Router

Flask Blueprints are also supported allowing `Flask-Via`.

You can either let `Flask-Via` automatically create and register your blueprint or create an instance of your blueprint and pass that to the Blueprint router.

See also:

- `flask_via.routers.default.Blueprint`.

Note: All routes will be added to the blueprint rather than the flask application, this applies to any routes included using the `Include` router.

Arguments:

- `name_or_instance`: A Blueprint name or a Blueprint instance

Keyword Arguments:

- `module`: Python module path to blueprint module, defaults to `None`
- `routes_module_name`: The module `Flask-Via` will look for within the blueprint module which contains the routes, defaults to `routes`
- `routes_name`: If you have not called the list of routes in the module `routes` you can set that here, for example `urls`.
- `static_folder`: Path to static files for blueprint, defaults to `None`
- `static_url_path`: URL path for blueprint static files, defaults to `None`
- `template_folder`: Templates folder name, defaults to `None`
- `url_prefix`: URL prefix for routes served within the blueprint, defaults to `None`
- `subdomain` : Sub domain for blueprint, defaults to `None`

- `url_defaults`: Callback function for URL defaults for this blueprint. It's called with the endpoint and values and should update the values passed in place, defaults to `None`.

Automatic Example

Let us assume we have the following application structure:

```
/path/to/foo
    - bar/
        - templates/
            - foo.html
        - __init__.py
        - routes.py
        - views.py
    - __int__.py
    - routes.py
```

In the above structure `bar` is a Flask blueprint which we wish to add to our flask application, so our top level routes would look like this:

```
from flask.ext.via.routers.default import Blueprint

routes = [
    Blueprint('bar', 'foo.bar', template_folder='templates')
]
```

You will note we give the blueprint a name and pass the top level module path to the blueprint rather than a path to the routes file.

In our blueprints views we can define routes as normal:

```
from flask.ext.via.routes import default
from foo.bar.views import some_view

routes = [
    default.Functional('/bar', some_view)
]
```

Instance Example

If you do not wish `Flask-Via` to automatically create the Blueprint instance you can pass a Blueprint instance as the first and only argument into the.

In the above example we would alter the contents of `/path/to/foo/bar/routes.py` as follows:

```
from flask import Blueprint
from flask.ext.via.routes import default
from foo.bar.views import some_view

blueprint = Blueprint('bar', 'foo.bar', template_folder='templates')

routes = [
    default.Functional('/bar', some_view)
]
```

And now in our `/path/to/foo/routes.py` we would import the blueprint and pass it into the router:

```
from foo.bar.routes import blueprint
from flask.ext.via.routers.default import Blueprint

routes = [
    Blueprint(blueprint)
]
```

Of course you can crate your Blueprint instance where ever you wish.

Including Blueprints

You can use the `flask_via.routers.Include` router to also include blueprints, you can even add `url_prefix` to prefix the blueprints `url_prefix`, crazy eh?

Example

Let us assume we have the same application structure as in the earlier blueprint examples, except our top level `routes.py` now looks like this:

```
from flask.ext.via.routers import default, Include

routes = [
    Include(
        'foo.routes',
        routes_name='api',
        url_prefix='/api/v1',
        endpoint='api.v1')
]

api = [
    default.Blueprint('bar', 'foo.bar', url_prefix='/bar')
    # These don't exist but just for illustraion purposes
    default.Blueprint('baz', 'foo.baz', url_prefix'/baz')
    default.Blueprint('fap', 'foo.fap', url_prefix'/fap')
]
```

Here we will include all the routes defined in the `api` list which are all blueprints, each blueprint will be registered with a `url_prefix` of `/api/v1` as well their url prefixes for the blueprint, so the above blueprints will be accessible on the followibg urls:

- `/api/v1/bar`
- `/api/v1/baz`
- `/api/v1/fap`

If each of these blueprints had a route defined with a url of `/bar` these would be accessed on the following urls:

- `/api/v1/bar/bar`
- `/api/v1/baz/bar`
- `/api/v1/fap/bar`

Hopefully you can see from this that `flask_via.routers.Include` coupled with `flask_via.routers.default.Blueprint` can offer some potentially powerful routing options for your application.

You will also notice we used the `endpoint` keyword agument in the Include. This means our urls can also be reversed using `url_for`, for example:

```
* ``url_for('api.v1.bar.bar')`` would return: ``/api/v1/bar/bar``
* ``url_for('api.v1.baz.bar')`` would return: ``/api/v1/baz/bar``
* ``url_for('api.v1.fap.bar')`` would return: ``/api/v1/fap/bar``
```

4.6 Examples

Here you can find examples of how to use `Flask-Via`. All examples are on GitHub.

- Basic Example
- Pluggable Example
- Mixed Routers Example Example
- Basic Restful Example
- Basic Admin Example
- Small Application Example
- Include Application Example
- Blueprint Application Example

CHAPTER 5

Reference

5.1 API

5.1.1 flask_via

class `flask_via.`**`RoutesImporter`**

Bases: `object`

Handles the import of routes module and obtaining a list of routes from that module as well as loading each route onto the application

New in version 2014.05.06.

`include`(*routes_module*, *routes_name*)

Imports a routes module and gets the routes from within that module and returns them.

Parameters

- **routes_module** (*str*) – Python dotted path to routes module
- **routes_name** (*str*) – Module attribute name to use when attempted to get the routes

Returns List of routes in the module

Return type list

Raises

- `ImportError` – If the route module cannot be imported
- `AttributeError` – If routes do not exist in the moduke

`load`(*app*, *routes*, ***kwargs*)

Loads passed routes onto the application by calling each routes `add_to_app` method which must be implemented by the route class.

class `flask_via.`**`Via`**

Bases: `flask_via.RoutesImporter`

The core class which kicks off the whole registration processes.

New in version 2014.05.06.

Example

```
from flask import Flask
from flask.ext.via import Via
from flask.ext.via.routers.flask import Basic

app = Flask(__name__)

def foo(bar=None):
    return 'Foo View!'

routes = [
    Basic('/foo', foo),
    Basic('/foo/<bar>', foo, endpoint='foo2'),
]

via = Via()
via.init_app(app, routes_module='path.to.here')

if __name__ == "__main__":
    app.run(debug=True)
```

init_app(*app*, *routes_module=None*, *routes_name=None*, ***kwargs*)
Initialises Flask extension. Bootstraps the automatic route registration process.

Changed in version 2014.05.19: Replace `NotImplementedError` with `ImproperlyConfigured` `routes_name` keyword argument default value set to `None` `routes_name` can now be configured using `VIA_ROUTES_NAME` app configuration variable. If `routes_name` keyword argument and `VIA_ROUTES_NAME` are not configured the default will be routes.

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

Keyword Arguments

- **route_module** (*str, optional*) – Python dotted path to where routes are defined, defaults to `None`
- **routes_name** (*str, optional*) – Within the routes module look for a variable of this name, defaults to `None`
- ****kwargs** – Arbitrary keyword arguments passed to `add_url_rule`

Raises `ImproperlyConfigured` – If `VIA_ROUTES_MODULE` is not configured in application config and `route_module` keyword argument has not been provided.

5.1.2 flask_via.exceptions

Custom exceptions which can be thrown by `Flask-Via`.

exception `flask_via.exceptions.`**ImproperlyConfigured**
Bases: `exceptions.Exception`

Raised in the event `Flask-Via` has not been properly configured

5.1.3 flask_via.routers

Base router classes and utilities.

class `flask_via.routers.`**BaseRouter**
Bases: `object`

Base router class all routers should inherit from providing common router functionality.

New in version 2014.05.06.

Example

```
from flask.ext.via.routers import BaseRouter

class MyRouter(BaseRouter):

    def __init__(self, arg):
        ...

    def add_to_app(self, app):
        ...
```

__init__()
Constructor should be overridden to accept specific arguments for the router.

Raises `NotImplementedError` – If method not implemented

add_to_app(*app*, ***kwargs*)
Method all routers require, which handles adding the route to the application instance.

Raises `NotImplementedError` – If method not implemented

class `flask_via.routers.`**Include**(*routes_module*, *routes_name=None*, *url_prefix=None*, *endpoint=None*)
Bases: `flask_via.routers.BaseRouter`, `flask_via.RoutesImporter`

Adds the ability to include routes from other modules, this can be handy when you want to break out your routes into separate files for sanity.

New in version 2014.05.06.

Note: This is not a implementation of Flask blueprints

__init__(*routes_module*, *routes_name=None*, *url_prefix=None*, *endpoint=None*)
Constructor for Include router, taking the passed arguments and storing them on the instance.

Changed in version 2014.05.08: `url_prefix` argument added

Changed in version 2014.05.19: `routes_name` keyword argument default value set to `None` `endpoint` keyword argument added

Parameters **routes_module** (*str*) – Python dotted path to the routes module

Keyword Arguments

- **routes_name** (*str, optional*) – Name of the variable holding the routes in the module, defaults to `routes`
- **url_prefix** (*str, optional*) – Adds a url prefix to all routes included by the router, defaults to `None`

- **endpoint** (*str, optional*) – Prefix an endpoint to all routes included, defaults to `None`

add_to_app(*app*, ***kwargs*)

Instead of adding a route to the flask application this will include and load routes similar, same as in the `flask_via.Via` class.abs

Changed in version 2014.05.08: `url_prefix` now injected into kwargs when loading in routes

Changed in version 2014.05.19: `endpoint` now injects into kwargs when loading in routes

•Parameters

- **app** (*flask.app.Flask*) – Flask application instance
- ****kwargs** – Arbitrary keyword arguments passed in to `init_app`

5.1.4 flask_via.routers.default

A set of flask specific router classes to be used when defining routes.

Example

```
from flask.ext.via.routes.flask import Basic, Pluggable
from yourapp.views import BarView, foo_view

routes = [
    Basic('/foo', 'foo', foo_view),
    Pluggable('/bar', BarView, 'bar'),
]
```

class `flask_via.routers.default.`**Basic**(*url*, *func*, *endpoint=None*)

Bases: `flask_via.routers.default.Functional`

This is deprecated and will be removed in the next release. Please use `Functional`.

New in version 2014.05.06.

Deprecated since version 2014.05.19.

class `flask_via.routers.default.`**Blueprint**(*name_or_instance*, *module=None*, *routes_module_name='routes'*, *routes_name=None*, *static_folder=None*, *static_url_path=None*, *template_folder=None*, *url_prefix=None*, *subdomain=None*, *url_defaults=None*)

Bases: `flask_via.routers.BaseRouter`, `flask_via.RoutesImporter`

Registers a flask blueprint and registers routes to that blueprint, similar to `flask_via.routers.Include`.

New in version 2014.05.06.

Example

Auto creates Blueprint instance*

```
from flask.ext.via.routers import default

routes = [
    default.Blueprint('foo', 'flask_via.examples.blueprints.foo')
]
```

Pass existing Blueprint instance*

```
from flask import Blueprint
from flask.ext.via.routers import default

blueprint = Blueprint('foo', __name__)

routes = [
    default.Blueprint(blueprint)
]
```

__init__(*name_or_instance*, *module=None*, *routes_module_name='routes'*, *routes_name=None*, *static_folder=None*, *static_url_path=None*, *template_folder=None*, *url_prefix=None*, *subdomain=None*, *url_defaults=None*)

Constructor for blueprint router.

Changed in version 2014.05.19: Replaced `name` with `name_or_instance` argument which allows the router to take an already instantiated blueprint instance. `module` argument optional when instance is passed as the first argument `routes_name` keyword argument default value set to `None`

Parameters **name** (*str, flask.blueprints.Blueprint*) – Blueprint name or a Blueprint class instance

Keyword Arguments

- **module** (*str*) – Python dotted path to the blueprint module
- **routes_module_name** (*str, optional*) – The module `Flask-Via` will look for within the blueprint module which contains the routes, defaults to `routes`
- **routes_name** (*str, optional*) – Name of the variable holding the routes in the module, defaults to `None`
- **static_folder** (*str, optional*) – Path to static files for blueprint, defaults to `None`
- **static_url_path** (*str, optional*) – URL path for blueprint static files, defaults to `None`
- **template_folder** (*str, optional*) – Templates folder name, defaults to `None`
- **url_prefix** (*str, optional*) – URL prefix for routes served within the blueprint, defaults to `None`
- **subdomain** (*str, optional*) – Sub domain for blueprint, defaults to `None`
- **url_defaults** (*function, optional*) – Callback function for URL defaults for this blueprint. It's called with the endpoint and values and should update the values passed in place, defaults to `None`.

add_to_app(*app*, ***kwargs*)

Creates a Flask blueprint and registers routes with that blueprint, this means any routes defined will be added to the blueprint rather than the application.

Parameters

- **app** (*flask.app.Flask*) – Flask application instance
- ****kwargs** – Arbitrary keyword arguments passed in to `init_app`

blueprint(***kwargs*)
Returns a Flask Blueprint instance, either one provided or created here.

Changed in version 2014.05.19: Renamed method from `create_blueprint` to `blueprint` If `instance` attribute exists, use this is as the blueprint else create the blueprint. Support for endpoint prefixing

•**Returns** An instantiated Flask Blueprint instance

Return type flask.blueprints.Blueprint

routes_module
Generates the routes module path, this is built from `self.module` and `self.routes_module_name`.

Returns Python dotted path to the routes module containing routes.

Return type str

class `flask_via.routers.default.`**Functional**(*url*, *func*, *endpoint=None*)
Bases: `flask_via.routers.BaseRouter`

A basic Flask router, used for the most basic form of flask routes, namely functionally based views which would normally use the `@route` decorator.

New in version 2014.05.19.

Example

```
from flask.ext.via.routes import default
from yourapp.views import foo_view, bar_view

routes = [
    default.Functional('/foo', 'foo', foo_view),
    default.Functional('/bar', 'bar', bar_view),
]
```

__init__(*url*, *func*, *endpoint=None*)
Basic router constructor, stores passed arguments on the instance.

Parameters

- **url** (*str*) – The url to use for the route
- **func** (*function*) – The view function to connect the route with

Keyword Arguments **endpoint** (*str, optional*) – Optional endpoint string, by default flask will use the view function name as the endpoint name, use this argument to change the endpoint name.

add_to_app(*app*, ***kwargs*)
Adds the url route to the flask application object.mro

Changed in version 2014.05.08: `url_prefix` can now be prefixed if present in kwargs

Changed in version 2014.05.19: `endpoint` can now be prefixed if present in kwargs

•**Parameters**

- **app** (*flask.app.Flask*) – Flask application instance
- ****kwargs** – Arbitrary keyword arguments passed in to `init_app`

class flask_via.routers.default.**Pluggable**(*url*, *view*, *endpoint*, ***kwargs*)
Bases: flask_via.routers.BaseRouter

Pluggable View router class, allows Flask pluggable view routes to be added to the flask application.

New in version 2014.05.06.

Example

```
from flask.ext.via.routers import flask
from flask.views import MethodView

class FooView(MethodView):
    def get(self):
        return 'foo view'

class BarView(MethodView):
    def get(self):
        return 'bar view'

routes = [
    flask.Pluggable('/', FooView, 'foo')
    flask.Pluggable('/', BarView, 'bar')
]
```

__init__(*url*, *view*, *endpoint*, ***kwargs*)
Pluggable router constructor, stores passed arguments on instance.

Changed in version 2014.05.19: Added `view` argument Added `endpoint` argument

•Parameters

- **url** (*str*) – The url to use for the route
- **view** (*class*) – The Flask pluggable view class, for example: * `flask.views.View` * `flask.views.MethodView`
- **endpoint** (*str*) – The Flask endpoint name for the view, this is required for Flask pluggable views.
- ****kwargs** – Arbitrary keyword arguments for `add_url_rule`

add_to_app(*app*, ***kwargs*)
Adds the url route to the flask application object.

Changed in version 2014.05.19: Updated `add_url_rule` to support endpoint prefixing and support new way of defining Pluggable views

Parameters

- **app** (*flask.app.Flask*) – Flask application instance
- ****kwargs** – Arbitrary keyword arguments passed in to `init_app`

5.1.5 flask_via.routers.restful

Routers for the Flask-Restful framework.

class flask_via.routers.restful.**Resource**(*url*, *resource*, *endpoint=None*)
Bases: flask_via.routers.BaseRouter

The Resource router allows you to define Flask-Restful routes and have those API resources added to the application automatically. For this to work you must at `init_app` time pass a optional keyword argument `restful_api` to `init_app` with its value being the restful api extension instance.

New in version 2014.05.06.

Example

```
app = Flask(__name__)
api = restful.Api(app)

class FooResource(restful.Resource):

    def get(self):
        return {'hello': 'world'}

routes = [
    Resource('/foo', FooResource)
]

via = Via()
via.init_app(
    app,
    routes_module='flask_via.examples.restful',
    restful_api=api)

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

__init__(*url*, *resource*, *endpoint=None*)
Constructor for flask restful resource router.

Parameters

- **url** (*str*) – The url to use for the route
- **resource** – A flask `restful.Resource` resource class

Keyword Arguments **endpoint** (*str, optional*) – Optional, override `Flask-Restful` automatic endpoint naming

add_to_app(*app*, ***kwargs*)
Adds the restul api resource route to the application.

Parameters

- **app** (*flask.app.Flask*) – Flask application instance, this is ignored.
- ****kwargs** – Arbitrary keyword arguments

Raises `NotImplementedError` – If `restful_api` is not provided

5.1.6 flask_via.routers.admin

Routers for the Flask-Admin framework.

class flask_via.routers.admin.**AdminRoute**(*view*)

Bases: flask_via.routers.BaseRouter

The Admin router allows you to define Flask-Admin routes and have those views added to the application automatically. For this to work you must at init_app time pass a optional keyword argument flask_admin to init_app with its value being the Flask-Aadmin extension instance.

New in version 2014.05.08.

Note: Flask-Admin has its own way of handling defining urls so this router literally only requires the Flask-Admin view class.

Example

```
app = Flask(__name__)

admin = Admin(name='Admin')
admin.init_app(app)

class FooAdminView(BaseView):

    @expose('/')
    def index(self):
        return 'foo'

routes = [
    AdminRoute(FooAdminView(name='Foo'))
]

via = Via()
via.init_app(
    app,
    routes_module='flask_via.examples.admin',
    flask_admin=admin)

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

__init__(*view*)

Admin route constructor, this router handles adding Flask-admin views to the application.

Parameters **view** (*flask_admin.base.AdminViewMeta*) – The Flask Admin View Class

add_to_app(*app*, ***kwargs*)

Adds the Flask-Admin View to the Flask the application.

Parameters

- **app** (*flask.app.Flask*) – Flask application instance, this is ignored.
- ****kwargs** – Arbitrary keyword arguments

Raises NotImplementedError – If flask_admin is not provided

5.2 Change Log

5.2.1 2014.05.19

- Feature: `Include` now supports `endpoint` prefixing
- Feature: Blueprint router can now take a blueprint instance
- Feature: Added support for `VIA_ROUTES_NAME` to set a common routes name
- Deprecated: `Basic` Router in favour of the `Functional` router
- Improved: `Pluggable` Router API is now cleaner
- Improved: Test Suite now uses PyTest
- Improved: `ImproperlyConfigured` now raised if routes module is not defined in either `init_app` or in application configuration via `VIA_ROUTES_MODULE`

5.2.2 2014.05.08

- Feature: Flask Admin Router
- Feature: Include `url_prefix` option

5.2.3 2014.05.06

- Feature: Flask extension initialisation
- Feature: Basic and Pluggable Flask Routers
- Feature: Flask-Restful Router
- Feature: Ability to include other routes
- Feature: Ability to register blueprints

5.3 Contributors

Without the work of these people or organisations this project would not be possible, we salute you.

- Soon London: http://thisissoon.com | @thisissoon
- Chris Reeves: @krak3n
- Greg Reed: @peeklondon
- Jack Saunders: @jackqu7

CHAPTER 6

Indices and tables

- *genindex*
- *modindex*
- *search*

f