flask-wtf Documentation

Release 0.9.5

Dan Jacob

October 03, 2016

Contents

Flask-WTF offers simple integration with WTForms.

CHAPTER 1

Current Version

The current version of Flask-WTF is 0.9.5.

View other versions of documentation at Read the Docs.

CHAPTER 2

Features

- Integration with wtforms.
- Secure Form with csrf token.
- Global csrf protection.
- Recaptcha supporting.
- File upload that works with Flask-Uploads.
- Internationalization integration.

CHAPTER 3

User's Guide

This part of the documentation, which is mostly prose, begins with some background information about Flask-WTF, then focuses on step-by-step instructions for getting the most out of Flask-WTF.

3.1 Installation

This part of the documentation covers the installation of Flask-WTF. The first step to using any software package is getting it properly installed.

3.1.1 Distribute & Pip

Installing Flask-WTF is simple with pip:

```
$ pip install Flask-WTF
```

or, with easy_install:

```
$ easy_install Flask-WTF
```

But, you really shouldn't do that.

3.1.2 Get the Code

Flask-WTF is actively developed on GitHub, where the code is always available.

You can either clone the public repository:

```
git clone git://github.com/lepture/flask-wtf.git
```

Download the tarball:

```
$ curl -OL https://github.com/lepture/flask-wtf/tarball/master
```

Or, download the zipball:

```
$ curl -OL https://github.com/lepture/flask-wtf/zipball/master
```

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages easily:

```
$ python setup.py install
```

3.2 Quickstart

Eager to get started? This page gives a good introduction to Flask-WTF. It assumes you already have Flask-WTF installed. If you do not, head over to the Installation section.

3.2.1 Creating Forms

Flask-WTF provides your Flask application integration with WTForms. For example:

```
from flask_wtf import Form
from wtforms import StringField
from wtforms.validators import DataRequired

class MyForm(Form):
    name = StringField('name', validators=[DataRequired()])
```

Note: From version 0.9.0, Flask-WTF will not import anything from wtforms, you need to import fields from wtforms.

In addition, a CSRF token hidden field is created automatically. You can render this in your template:

```
<form method="POST" action="/">
    {{ form.csrf_token }}
    {{ form.name.label }} {{ form.name(size=20) }}
    <input type="submit" value="Go">
</form>
```

However, in order to create valid XHTML/HTML the Form class has a method hidden_tag which renders any hidden fields, including the CSRF field, inside a hidden DIV tag:

```
<form method="POST" action="/">
    {{ form.hidden_tag() }}
    {{ form.name.label }} {{ form.name(size=20) }}
    <input type="submit" value="Go">
</form>
```

3.2.2 Validating Forms

Validating the request in your view handlers:

```
@app.route('/submit', methods=('GET', 'POST'))
def submit():
    form = MyForm()
    if form.validate_on_submit():
        return redirect('/success')
    return render_template('submit.html', form=form)
```

Note that you don't have to pass `request.form` to Flask-WTF; it will load automatically. And the convenience `validate_on_submit` will check if it is a POST request and if it is valid.

Heading over to Creating Forms to learn more skills.

3.3 Creating Forms

This part of the documentation covers the Form parts.

3.3.1 Secure Form

Without any configuration, the *Form* will be a session secure form with csrf protection. We encourage you do nothing.

But if you want to disable the csrf protection, you can pass:

```
form = Form(csrf_enabled=False)
```

If you want to disable it globally, which you really shouldn't. But if you insist, it can be done with the configuration:

```
WTF_CSRF_ENABLED = False
```

In order to generate the csrf token, you must have a secret key, this is usually the same as your Flask app secret key. If you want to use another secret key, config it:

```
WTF_CSRF_SECRET_KEY = 'a random string'
```

3.3.2 File Uploads

Flask-WTF provides you a *FileField* to handle file uploading, it will automatically draw data from `flask.request.files` if the form is posted. The `data` attribute of *FileField* will be an instance of Werkzeug FileStorage.

For example:

```
from werkzeug import secure_filename
from flask_wtf.file import FileField

class PhotoForm(Form):
    photo = FileField('Your photo')

@app.route('/upload/', methods=('GET', 'POST'))
def upload():
    form = PhotoForm()
    if form.validate_on_submit():
        filename = secure_filename(form.photo.data.filename)
        form.photo.data.save('uploads/' + filename)
    else:
        filename = None
    return render_template('upload.html', form=form, filename=filename)
```

Note: Remember to set the `enctype` of your HTML form to `multipart/form-data`, which means:

```
<form action="/upload/" method="POST" enctype="multipart/form-data">
    ....
</form>
```

More than that, Flask-WTF supports validation on file uploading. There are *FileRequired* and *FileAllowed*.

The *FileAllowed* works well with Flask-Uploads, for example:

```
from flask.ext.uploads import UploadSet, IMAGES
from flask_wtf import Form
from flask_wtf.file import FileField, FileAllowed, FileRequired

images = UploadSet('images', IMAGES)

class UploadForm(Form):
    upload = FileField('image', validators=[
        FileRequired(),
        FileAllowed(images, 'Images only!')
    ])
```

It can work without Flask-Uploads too. You need to pass the extensions to *FileAllowed*:

```
class UploadForm(Form):
    upload = FileField('image', validators=[
        FileRequired(),
        FileAllowed(['jpg', 'png'], 'Images only!')
    ])
```

3.3.3 HTML5 Widgets

Note: HTML5 widgets and fields are builtin of wtforms since 1.0.5. You should consider import them from wtforms if possible.

We will drop html5 module in next release 0.9.3.

You can import a number of HTML5 widgets from `wtforms`:

```
from wtforms.fields.html5 import URLField
from wtforms.validators import url

class LinkForm(Form):
    url = URLField(validators=[url()])
```

3.3.4 Recaptcha

Flask-WTF also provides Recaptcha support through a `RecaptchaField`:

```
from flask_wtf import Form, RecaptchaField
from wtforms import TextField

class SignupForm(Form):
    username = TextField('Username')
    recaptcha = RecaptchaField()
```

This comes together with a number of configuration, which you have to implement them.

RECAPTCHA_PUBLIC_KEY	**required** A public key.
RECAPTCHA_PRIVATE_KEY	**required** A private key.
RECAPTCHA_API_SERVER	**optional** Specify your Recaptcha API server.
RECAPTCHA_OPTIONS	**optional** A dict of configuration options. https://www.google.com/recaptcha/admin/create

For testing your application, if `app.testing` is `True`, recaptcha field will always be valid for you convenience.

And it can be easily setup in the templates:

```
<form action="/" method="post">
    {{ form.username }}
    {{ form.recaptcha }}
</form>
```

We have an example for you: recaptcha@github.

3.4 CSRF Protection

This part of the documentation covers the CSRF protection.

3.4.1 Why CSRF

Flask-WTF form is already protecting you from CSRF, you don't have to worry about that. However, you have views that contain no forms, and they still need protection.

For example, the POST request is sent by AJAX, but it has no form behind it. You can't get the csrf token prior 0.9.0 of Flask-WTF. That's why we created this CSRF for you.

3.4.2 Implementation

To enable CSRF protection for all your view handlers, you need to enable the *CsrfProtect* module:

```
from flask_wtf.csrf import CsrfProtect

CsrfProtect(app)
```

Like any other Flask extensions, you can load it lazily:

```
from flask_wtf.csrf import CsrfProtect

csrf = CsrfProtect()

def create_app():
    app = Flask(__name__)
    csrf.init_app(app)
```

Note: You need to setup a secret key for CSRF protection. Usually, this is the same as your Flask app SECRET_KEY.

If the template has a form, you don't need to do any thing. It is the same as before:

```
<form method="post" action="/">
    {{ form.csrf_token }}
</form>
```

But if the template has no forms, you still need a csrf token:

```
<form method="post" action="/">
    <input type="hidden" name="csrf_token" value="{{ csrf_token() }}" />
</form>
```

Whenever a CSRF validation fails, it will return a 400 response. You can customize the error response:

```
@csrf.error_handler
def csrf_error(reason):
    return render_template('csrf_error.html', reason=reason), 400
```

We strongly suggest that you protect all your views from CSRF. But there is a chance that you might exclude some view handlers, it can be done:

```
@csrf.exempt
@app.route('/foo', methods=('GET', 'POST'))
def my_handler():
    # ...
    return 'ok'
```

3.4.3 AJAX

Sending POST requests via AJAX is possible where there is no forms at all. This feature is only available since 0.9.0.

Assumimg you have done `CsrfProtect(app)`, you can get the csrf token via `{{ csrf_token() }}`. This method is available in every templates, that way you don't have to worry if there are no forms for rendering the csrf token field.

The suggested way is that you render the token in a `<meta>` tag:

```
<meta name="csrf-token" content="{{ csrf_token() }}">
```

And it is also possible to render it in the `<script>` tag:

```
<script type="text/javascript">
    var csrftoken = "{{ csrf_token() }}"
</script>
```

We will take the `<meta>` way for example, the `<script>` way is far more easier, you don't have to worry if there is no example for it.

Whenever you send a AJAX POST request, add the `X-CSRFToken` for it:

```
var csrftoken = $('meta[name=csrf-token]').attr('content')

$.ajaxSetup({
    beforeSend: function(xhr, settings) {
        if (!/^(GET|HEAD|OPTIONS|TRACE)$/i.test(settings.type) && !this.crossDomain) {
            xhr.setRequestHeader("X-CSRFToken", csrftoken)
        }
    }
})
```

3.4.4 Troubleshooting

When you define your forms, if you make the mistake of importing `Form` from `wtforms` instead of from `flask.ext.wtf`, most features besides CSRF protection will work (aside from `form.validate_on_submit()`), but CSRF protection will fail. Upon submitting forms, you'll get `Bad Request/CSRF token missing or incorrect` (and the `form.csrf_token` in your template will produce no output). The problem is in your broken import statements, not your configuration.

3.5 Configuration

Here is the full table of all configurations.

3.5.1 Forms and CSRF

The full list of configuration for Flask-WTF. Usually, you don't need to configure any of them. It just works.

WTF_CSRF_ENABLED	Disable/enable CSRF protection for forms. Default is True.
WTF_I18N_ENABLED	Disable/enable I18N support. This should work together with Flask-Babel. Default is True.
WTF_CSRF_HEADERS	CSRF token HTTP headers checked. Default is **['X-CSRFToken', 'X-CSRF-Token']**
WTF_CSRF_SECRET_KEY	A random string for generating CSRF token. Default is the same as SECRET_KEY.
WTF_CSRF_TIME_LIMIT	CSRF token expiring time. Default is **3600** seconds.
WTF_CSRF_SSL_STRICT	Strictly protection on SSL. This will check the referrer, validate if it is from the same origin. Default is True.
WTF_CSRF_METHODS	CSRF protection on these request methods. Default is **['POST', 'PUT', 'PATCH']**
WTF_HIDDEN_TAG	HTML tag name of the hidden tag wrapper. Default is **div**
WTF_HIDDEN_TAG_ATTRS	HTML tag attributes of the hidden tag wrapper. Default is **{'style': 'display:none;'}**

3.5.2 Recaptcha

You have already learned these configuration at *Recaptcha*. This table is only designed for a convience.

RECAPTCHA_USE_SSL	Enable/disable recaptcha through ssl. Default is False.
RECAPTCHA_PUBLIC_KEY	**required** A public key.
RECAPTCHA_PRIVATE_KEY	**required** A private key.
RECAPTCHA_OPTIONS	**optional** A dict of configuration options. https://www.google.com/recaptcha/admin/create

CHAPTER 4

API Documentation

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

4.1 Developer Interface

This part of the documentation covers all interfaces of Flask-WTF.

4.1.1 Forms and Fields

class `flask_wtf.`**`Form`**(*formdata=<class 'flask_wtf.form._Auto'>*, *obj=None*, *prefix=''*, *csrf_context=None*, *secret_key=None*, *csrf_enabled=None*, **args*, ***kwargs*)

Flask-specific subclass of WTForms **SecureForm** class.

If formdata is not specified, this will use flask.request.form. Explicitly pass formdata = None to prevent this.

Parameters

- **`csrf_context`** – a session or dict-like object to use when making CSRF tokens. Default: flask.session.
- **`secret_key`** – a secret key for building CSRF tokens. If this isn't specified, the form will take the first of these that is defined:
  - SECRET_KEY attribute on this class
  - WTF_CSRF_SECRET_KEY config of flask app
  - SECRET_KEY config of flask app
  - session secret key
- **`csrf_enabled`** – whether to use CSRF protection. If False, all csrf behavior is suppressed. Default: WTF_CSRF_ENABLED config value

`hidden_tag`(**fields*)

Wraps hidden fields in a hidden DIV tag, in order to keep XHTML compliance.

New in version 0.3.

Parameters **`fields`** – list of hidden field names. If not provided will render all hidden fields, including the CSRF field.

`is_submitted`()

Checks if form has been submitted. The default case is if the HTTP method is **PUT** or **POST**.

validate_csrf_data(*data*)
Check if the csrf data is valid.

Parameters **data** – the csrf string to be validated.

validate_on_submit()
Checks if form has been submitted and if so runs validate. This is a shortcut, equivalent to `form.is_submitted() and form.validate()`

class flask_wtf.**RecaptchaField**(*label=''*, *validators=None*, ***kwargs*)

class flask_wtf.**Recaptcha**(*message=None*)
Validates a ReCaptcha.

class flask_wtf.**RecaptchaWidget**

class flask_wtf.file.**FileField**(*label=None*, *validators=None*, *filters=()*, *description=''*, *id=None*, *default=None*, *widget=None*, *render_kw=None*, *_form=None*, *_name=None*, *_prefix=''*, *_translations=None*, *_meta=None*)
Werkzeug-aware subclass of **wtforms.FileField**

Provides a *has_file()* method to check if its data is a FileStorage instance with an actual file.

has_file()
Return True iff self.data is a FileStorage with file data

class flask_wtf.file.**FileAllowed**(*upload_set*, *message=None*)
Validates that the uploaded file is allowed by the given Flask-Uploads UploadSet.

Parameters

- **upload_set** – A list/tuple of extention names or an instance of `flask.ext.uploads.UploadSet`
- **message** – error message

You can also use the synonym **file_allowed**.

class flask_wtf.file.**FileRequired**(*message=None*)
Validates that field has a file.

Parameters **message** – error message

You can also use the synonym **file_required**.

class flask_wtf.html5.**SearchInput**(*input_type=None*)
Renders an input with type "search".

class flask_wtf.html5.**SearchField**(*label=None*, *validators=None*, *filters=()*, *description=''*, *id=None*, *default=None*, *widget=None*, *render_kw=None*, *_form=None*, *_name=None*, *_prefix=''*, *_translations=None*, *_meta=None*)
Represents an `<input type="search">`.

class flask_wtf.html5.**URLInput**(*input_type=None*)
Renders an input with type "url".

class flask_wtf.html5.**URLField**(*label=None*, *validators=None*, *filters=()*, *description=''*, *id=None*, *default=None*, *widget=None*, *render_kw=None*, *_form=None*, *_name=None*, *_prefix=''*, *_translations=None*, *_meta=None*)
Represents an `<input type="url">`.

class flask_wtf.html5.**EmailInput**(*input_type=None*)
Renders an input with type "email".

class flask_wtf.html5.**EmailField**(*label=None*, *validators=None*, *filters=()*, *description=''*, *id=None*, *default=None*, *widget=None*, *render_kw=None*, *_form=None*, *_name=None*, *_prefix=''*, *_translations=None*, *_meta=None*)
Represents an <input type="email">.

class flask_wtf.html5.**TelInput**(*input_type=None*)
Renders an input with type "tel".

class flask_wtf.html5.**TelField**(*label=None*, *validators=None*, *filters=()*, *description=''*, *id=None*, *default=None*, *widget=None*, *render_kw=None*, *_form=None*, *_name=None*, *_prefix=''*, *_translations=None*, *_meta=None*)
Represents an <input type="tel">.

class flask_wtf.html5.**NumberInput**(*step=None*, *min=None*, *max=None*)
Renders an input with type "number".

class flask_wtf.html5.**IntegerField**(*label=None*, *validators=None*, ***kwargs*)
Represents an <input type="number">.

class flask_wtf.html5.**DecimalField**(*label=None*, *validators=None*, *places=<unset value>*, *rounding=None*, ***kwargs*)
Represents an <input type="number">.

class flask_wtf.html5.**RangeInput**(*step=None*)
Renders an input with type "range".

class flask_wtf.html5.**IntegerRangeField**(*label=None*, *validators=None*, ***kwargs*)
Represents an <input type="range">.

class flask_wtf.html5.**DecimalRangeField**(*label=None*, *validators=None*, *places=<unset value>*, *rounding=None*, ***kwargs*)
Represents an <input type="range">.

4.1.2 CSRF Protection

class flask_wtf.csrf.**CsrfProtect**(*app=None*)
Enable csrf protect for Flask.

Register it with:

```
app = Flask(__name__)
CsrfProtect(app)
```

And in the templates, add the token input:

```
<input type="hidden" name="csrf_token" value="{{ csrf_token() }}"/>
```

If you need to send the token via AJAX, and there is no form:

```
<meta name="csrf_token" content="{{ csrf_token() }}" />
```

You can grab the csrf token with JavaScript, and send the token together.

error_handler(*view*)
A decorator that set the error response handler.

It accepts one parameter *reason*:

```
@csrf.error_handler
def csrf_error(reason):
    return render_template('error.html', reason=reason)
```

By default, it will return a 400 response.

exempt (*view*)
A decorator that can exclude a view from csrf protection.

Remember to put the decorator above the *route*:

```
csrf = CsrfProtect(app)

@csrf.exempt
@app.route('/some-view', methods=['POST'])
def some_view():
    return
```

flask_wtf.csrf.**generate_csrf**(*secret_key=None*, *time_limit=None*)
Generate csrf token code.

Parameters

- **secret_key** – A secret key for mixing in the token, default is Flask.secret_key.
- **time_limit** – Token valid in the time limit, default is 3600s.

flask_wtf.csrf.**validate_csrf**(*data*, *secret_key=None*, *time_limit=None*)
Check if the given data is a valid csrf token.

Parameters

- **data** – The csrf token value to be checked.
- **secret_key** – A secret key for mixing in the token, default is Flask.secret_key.
- **time_limit** – Check if the csrf token is expired. default is True.

CHAPTER 5

Additional Notes

Legal information and changelog are here.

5.1 Upgrading to Newer Releases

Flask-WTF itself is changing like any software is changing over time. Most of the changes are the nice kind, the kind where you don't have to change anything in your code to profit from a new release.

However every once in a while there are changes that do require some changes in your code or there are changes that make it possible for you to improve your own code quality by taking advantage of new features in Flask-WTF.

This section of the documentation enumerates all the changes in Flask-WTF from release to release and how you can change your code to have a painless updating experience.

If you want to use the easy_install command to upgrade your Flask-WTF installation, make sure to pass it the -U parameter:

```
$ pip install -U Flask-WTF
```

5.1.1 Version 0.9.0

Dropping the imports of wtforms is a big change, it may be lots of pain for you, but the imports are hard to maintain. Instead of importing `Fields` from Flask-WTF, you need to import them from the original wtforms:

```
from wtforms import TextField
```

Configuration name of `CSRF_ENABLED` is changed to `WTF_CSRF_ENABLED`. There is a chance that you don't need to do anything if you haven't set any configuration.

This version has many more features, if you don't need them, they will not break any code of yours.

5.2 Flask-WTF Changelog

Full list of changes between each Flask-WTF release.

5.2.1 Version 0.11

Released 2015/01/21

- Use the new reCAPTCHA API via #164.

5.2.2 Version 0.10.3

Released 2014/11/16

- Add configuration: WTF_CSRF_HEADERS via #159.
- Support customize hidden tags via #150.
- And many more bug fixes

5.2.3 Version 0.10.2

Released 2014/09/03

- Update translation for reCaptcha via #146.

5.2.4 Version 0.10.1

Released 2014/08/26

- Update RECAPTCHA API SERVER URL via #145.
- Update requirement Werkzeug>=0.9.5
- Fix CsrfProtect exempt for blueprints via #143.

5.2.5 Version 0.10.0

Released 2014/07/16

- Add configuration: WTF_CSRF_METHODS
- Support WTForms 2.0 now
- Fix csrf validation without time limit (time_limit=False)
- CSRF exempt supports blueprint #111.

5.2.6 Version 0.9.5

Released 2014/03/21

- `csrf_token` for all template types #112.
- Make FileRequired a subclass of InputRequired #108.

5.2.7 Version 0.9.4

Released 2013/12/20

- Bugfix for csrf module when form has a prefix
- Compatible support for wtforms2
- Remove file API for FileField

5.2.8 Version 0.9.3

Released 2013/10/02

- Fix validation of recaptcha when app in testing mode #89.
- Bugfix for csrf module #91

5.2.9 Version 0.9.2

Released 2013/9/11

- Upgrade wtforms to 1.0.5.
- No lazy string for i18n #77.
- No DateInput widget in html5 #81.
- PUT and PATCH for CSRF #86.

5.2.10 Version 0.9.1

Released 2013/8/21

This is a patch version for backward compitable for Flask<0.10 #82.

5.2.11 Version 0.9.0

Released 2013/8/15

- Add i18n support (issue #65)
- Use default html5 widgets and fields provided by wtforms
- Python 3.3+ support
- Redesign form, replace SessionSecureForm
- CSRF protection solution
- Drop wtforms imports
- Fix recaptcha i18n support
- Fix recaptcha validator for python 3
- More test cases, it's 90%+ coverage now
- Redesign documentation

5.2.12 Version 0.8.4

Released 2013/3/28

- Recaptcha Validator now returns provided message (issue #66)
- Minor doc fixes
- Fixed issue with tests barking because of nose/multiprocessing issue.

5.2.13 Version 0.8.3

Released 2013/3/13

- Update documentation to indicate pending deprecation of WTForms namespace facade
- PEP8 fixes (issue #64)
- Fix Recaptcha widget (issue #49)

5.2.14 Version 0.8.2 and prior

Initial development by Dan Jacob and Ron Duplain. 0.8.2 and prior there was not a change log.

5.3 Authors

Flask-WTF is created by Dan Jacob, and now is maintained by Hsiaoming Yang.

5.3.1 Contributors

People who send patches and suggestions:

- Dan Jacob
- Ron DuPlain
- Daniel Lepage
- Anthony Ford
- Hsiaoming Yang

Find more contributors on GitHub.

5.4 BSD License

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- The names of the contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

f