# Flask-WTF Documentation

***Release 0.14.2.dev20170813***

**Dan Jacob**

**Aug 13, 2017**

# Contents

Simple integration of Flask and WTForms, including CSRF, file upload, and reCAPTCHA.

CHAPTER 1

# Features

- Integration with WTForms.
- Secure Form with CSRF token.
- Global CSRF protection.
- reCAPTCHA support.
- File upload that works with Flask-Uploads.
- Internationalization using Flask-Babel.

CHAPTER 2

# 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.

## 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.

### 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.

### 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
```

# 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.

## Creating Forms

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

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

class MyForm(FlaskForm):
    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>
```

If your form has multiple hidden fields, you can render them in one block using *hidden_tag()*.

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

## 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.

# Creating Forms

## Secure Form

Without any configuration, the *`FlaskForm`* 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 = FlaskForm(csrf_enabled=False)
```

You can disable it globally—though you really shouldn't—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'
```

## File Uploads

The *`FileField`* provided by Flask-WTF differs from the WTForms-provided field. It will check that the file is a non-empty instance of `FileStorage`, otherwise `data` will be `None`.

```
from flask_wtf import FlaskForm
from flask_wtf.file import FileField, FileRequired
from werkzeug.utils import secure_filename

class PhotoForm(FlaskForm):
    photo = FileField(validators=[FileRequired()])

@app.route('/upload', methods=['GET', 'POST'])
def upload():
    if form.validate_on_submit():
        f = form.photo.data
        filename = secure_filename(f.filename)
        f.save(os.path.join(
            app.instance_path, 'photos', filename
        ))
        return redirect(url_for('index'))
```

```
    return render_template('upload.html', form=form)
```

Remember to set the `enctype` of the HTML form to `multipart/form-data`, otherwise `request.files` will be empty.

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

Flask-WTF handles passing form data to the form for you. If you pass in the data explicitly, remember that `request.form` must be combined with `request.files` for the form to see the file data.

```
form = PhotoForm()
# is equivalent to:

from flask import request
from werkzeug.datastructures import CombinedMultiDict
form = PhotoForm(CombinedMultiDict((request.files, request.form)))
```

### Validation

Flask-WTF supports validating file uploads with *FileRequired* and *FileAllowed*. They can be used with both Flask-WTF's and WTForms's `FileField` classes.

*FileAllowed* works well with Flask-Uploads.

```
from flask_uploads import UploadSet, IMAGES
from flask_wtf import FlaskForm
from flask_wtf.file import FileField, FileAllowed, FileRequired

images = UploadSet('images', IMAGES)

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

It can be used without Flask-Uploads by passing the extensions directly.

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

## Recaptcha

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

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

class SignupForm(FlaskForm):
```

```
    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_PARAMETERS | **optional** A dict of JavaScript (api.js) parameters. |
| RECAPTCHA_DATA_ATTRS | **optional** A dict of data attributes options. https://developers.google.com/recaptcha/docs/display |

Example of RECAPTCHA_PARAMETERS, and RECAPTCHA_DATA_ATTRS:

```
RECAPTCHA_PARAMETERS = {'hl': 'zh', 'render': 'explicit'}
RECAPTCHA_DATA_ATTRS = {'theme': 'dark'}
```

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.

## CSRF Protection

Any view using *FlaskForm* to process the request is already getting CSRF protection. If you have views that don't use `FlaskForm` or make AJAX requests, use the provided CSRF extension to protect those requests as well.

### Setup

To enable CSRF protection globally for a Flask app, register the *CSRFProtect* extension.

```
from flask_wtf.csrf import CSRFProtect

csrf = CSRFProtect(app)
```

Like other Flask extensions, you can apply it lazily:

```
csrf = CSRFProtect()

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

**Note:** CSRF protection requires a secret key to securely sign the token. By default this will use the Flask app's `SECRET_KEY`. If you'd like to use a separate token you can set `WTF_CSRF_SECRET_KEY`.

## HTML Forms

When using a `FlaskForm`, render the form's CSRF field like normal.

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

If the template doesn't use a `FlaskForm`, render a hidden input with the token in the form.

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

## JavaScript Requests

When sending an AJAX request, add the `X-CSRFToken` header to it. For example, in jQuery you can configure all requests to send the token.

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

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

## Customize the error response

When CSRF validation fails, it will raise a *`CSRFError`*. By default this returns a response with the failure reason and a 400 code. You can customize the error response using Flask's `errorhandler()`.

```
from flask_wtf.csrf import CSRFError

@app.errorhandler(CSRFError)
def handle_csrf_error(e):
    return render_template('csrf_error.html', reason=e.description), 400
```

## Exclude views from protection

We strongly suggest that you protect all your views with CSRF. But if needed, you can exclude some views using a decorator.

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

You can exclude all the views of a blueprint.

```
csrf.exempt(account_blueprint)
```

You can disable CSRF protection in all views by default, by setting `WTF_CSRF_CHECK_DEFAULT` to `False`, and selectively call `protect()` only when you need. This also enables you to do some pre-processing on the requests before checking for the CSRF token.

```
@app.before_request
def check_csrf():
    if not is_oauth(request):
        csrf.protect()
```

# Configuration

| | |
|---|---|
| `WTF_CSRF_ENABLED` | Set to `False` to disable all CSRF protection. |
| `WTF_CSRF_CHECK_DEFAULT` | When using the CSRF protection extension, this controls whether every view is protected by default. Default is `True`. |
| `WTF_CSRF_SECRET_KEY` | Random data for generating secure tokens. If this is not set then `SECRET_KEY` is used. |
| `WTF_CSRF_METHODS` | HTTP methods to protect from CSRF. Default is `{'POST', 'PUT', 'PATCH', 'DELETE'}`. |
| `WTF_CSRF_FIELD_NAME` | Name of the form field and session key that holds the CSRF token. |
| `WTF_CSRF_HEADERS` | HTTP headers to search for CSRF token when it is not provided in the form. Default is `['X-CSRFToken', 'X-CSRF-Token']`. |
| `WTF_CSRF_TIME_LIMIT` | Max age in seconds for CSRF tokens. Default is `3600`. If set to `None`, the CSRF token is valid for the life of the session. |
| `WTF_CSRF_SSL_STRICT` | Whether to enforce the same origin policy by checking that the referrer matches the host. Only applies to HTTPS requests. Default is `True`. |
| `WTF_I18N_ENABLED` | Set to `False` to disable Flask-Babel I18N support. |

## Recaptcha

| | |
|---|---|
| `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. https://www.google.com/recaptcha/admin/create |
| `RECAPTCHA_OPTIONS` | **optional** A dict of configuration options. |

## Logging

CSRF errors are logged at the `INFO` level to the `flask_wtf.csrf` logger. You still need to configure logging in your application in order to see these messages.

CHAPTER 3

# API Documentation

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

## Developer Interface

### Forms and Fields

**class** `flask_wtf.`**`FlaskForm`**(*formdata=<object object>*, ***kwargs*)
Flask-specific subclass of WTForms `Form`.

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

**`hidden_tag`**(**fields*)
Render the form's hidden fields in one call.

A field is considered hidden if it uses the `HiddenInput` widget.

If `fields` are given, only render the given fields that are hidden. If a string is passed, render the field with that name if it exists.

Changed in version 0.13: No longer wraps inputs in hidden div. This is valid HTML 5.

Changed in version 0.13: Skip passed fields that aren't hidden. Skip passed names that don't exist.

**`is_submitted`**()
Consider the form submitted if there is an active request and the method is `POST`, `PUT`, `PATCH`, or `DELETE`.

**`validate_on_submit`**()
Call `validate()` only if the form is submitted. This is a shortcut for `form.is_submitted() and form.validate()`.

**class** `flask_wtf.`**`Form`**(...)
Deprecated since version 0.13: Renamed to *`FlaskForm`*.

**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.fields.FileField.

**has_file**()
Return True if self.data is a FileStorage object.

Deprecated since version 0.14.1: data is no longer set if the input is not a non-empty FileStorage. Check form.data is not None instead.

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

**Parameters**

- **upload_set** – A list of extensions or an UploadSet
- **message** – error message

You can also use the synonym file_allowed.

**class** flask_wtf.file.**FileRequired**(*message=None*)
Validates that the data is a Werkzeug FileStorage object.

**Parameters** **message** – error message

You can also use the synonym file_required.

## CSRF Protection

**class** flask_wtf.csrf.**CSRFProtect**(*app=None*)
Enable CSRF protection globally for a Flask app.

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

Checks the csrf_token field sent with forms, or the X-CSRFToken header sent with JavaScript requests. Render the token in templates using {{ csrf_token() }}.

See the *CSRF Protection* documentation.

**error_handler**(*view*)
Register a function that will generate the response for CSRF errors.

Deprecated since version 0.14: Use the standard Flask error system with @app.errorhandler(CSRFError) instead. This will be removed in version 1.0.

The function will be passed one argument, reason. By default it will raise a *CSRFError*.

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

Due to historical reasons, the function may either return a response or raise an exception with flask.abort().

**exempt**(*view*)
Mark a view or blueprint to be excluded from CSRF protection.

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

```
bp = Blueprint(...)
csrf.exempt(bp)
```

**class** flask_wtf.csrf.**CsrfProtect**(...)
Deprecated since version 0.14: Renamed to *CSRFProtect*.

**class** flask_wtf.csrf.**CSRFError**(*description=None*, *response=None*)
Raise if the client sends invalid CSRF data with the request.

Generates a 400 Bad Request response with the failure reason by default. Customize the response by registering a handler with flask.Flask.errorhandler().

flask_wtf.csrf.**generate_csrf**(*secret_key=None*, *token_key=None*)
Generate a CSRF token. The token is cached for a request, so multiple calls to this function will generate the same token.

During testing, it might be useful to access the signed token in g.csrf_token and the raw token in session['csrf_token'].

**Parameters**

- **secret_key** – Used to securely sign the token. Default is WTF_CSRF_SECRET_KEY or SECRET_KEY.
- **token_key** – Key where token is stored in session for comparision. Default is WTF_CSRF_FIELD_NAME or 'csrf_token'.

flask_wtf.csrf.**validate_csrf**(*data*, *secret_key=None*, *time_limit=None*, *token_key=None*)
Check if the given data is a valid CSRF token. This compares the given signed token to the one stored in the session.

**Parameters**

- **data** – The signed CSRF token to be checked.
- **secret_key** – Used to securely sign the token. Default is WTF_CSRF_SECRET_KEY or SECRET_KEY.
- **time_limit** – Number of seconds that the token is valid. Default is WTF_CSRF_TIME_LIMIT or 3600 seconds (60 minutes).
- **token_key** – Key where token is stored in session for comparision. Default is WTF_CSRF_FIELD_NAME or 'csrf_token'.

**Raises** **ValidationError** – Contains the reason that validation failed.

Changed in version 0.14: Raises ValidationError with a specific error message rather than returning True or False.

CHAPTER 4

# Additional Notes

Legal information and changelog are here.

## 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
```

### 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.

# Flask-WTF Changelog

## Version 0.14.2

Released 2017-01-10

- Fix bug where `FlaskForm` assumed `meta` argument was not `None` if it was passed. (#278)

## Version 0.14.1

Released 2017-01-10

- Fix bug where the file validators would incorrectly identify an empty file as valid data. (#276, #277)
  - `FileField` is no longer deprecated. The data is checked during processing and only set if it's a valid file.
  - `has_file` *is* deprecated; it's now equivalent to `bool(field.data)`.
  - `FileRequired` and `FileAllowed` work with both the Flask-WTF and WTForms `FileField` classes.
  - The `Optional` validator now works with `FileField`.

## Version 0.14

Released 2017-01-06

- Use itsdangerous to sign CSRF tokens and check expiration instead of doing it ourselves. (#264)
  - All tokens are URL safe, removing the `url_safe` parameter from `generate_csrf`. (#206)
  - All tokens store a timestamp, which is checked in `validate_csrf`. The `time_limit` parameter of `generate_csrf` is removed.
- Remove the `app` attribute from `CsrfProtect`, use `current_app`. (#264)
- `CsrfProtect` protects the `DELETE` method by default. (#264)
- The same CSRF token is generated for the lifetime of a request. It is exposed as `request.csrf_token` for use during testing. (#227, #264)
- `CsrfProtect.error_handler` is deprecated. (#264)
  - Handlers that return a response work in addition to those that raise an error. The behavior was not clear in previous docs.
  - (#200, #209, #243, #252)
- Use `Form.Meta` instead of deprecated `SecureForm` for CSRF (and everything else). (#216, #271)
  - `csrf_enabled` parameter is still recognized but deprecated. All other attributes and methods from `SecureForm` are removed. (#271)
- Provide `WTF_CSRF_FIELD_NAME` to configure the name of the CSRF token. (#271)
- `validate_csrf` raises `wtforms.ValidationError` with specific messages instead of returning `True` or `False`. This breaks anything that was calling the method directly. (#239, #271)
  - CSRF errors are logged as well as raised. (#239)

- `CsrfProtect` is renamed to `CSRFProtect`. A deprecation warning is issued when using the old name. `CsrfError` is renamed to `CSRFError` without deprecation. (#271)
- `FileField` is deprecated because it no longer provides functionality over the provided validators. Use `wtforms.FileField` directly. (#272)

## Version 0.13.1

Released 2016/10/6

- Deprecation warning for `Form` is shown during `__init__` instead of immediately when subclassing. (#262)
- Don't use `pkg_resources` to get version, for compatibility with GAE. (#261)

## Version 0.13

Released 2016/09/29

- `Form` is renamed to `FlaskForm` in order to avoid name collision with WTForms's base class. Using `Form` will show a deprecation warning. (#250)
- `hidden_tag` no longer wraps the hidden inputs in a hidden div. This is valid HTML5 and any modern HTML parser will behave correctly. (#217, #193)
- `flask_wtf.html5` is deprecated. Import directly from `wtforms.fields.html5`. (#251)
- `is_submitted` is true for `PATCH` and `DELETE` in addition to `POST` and `PUT`. (#187)
- `generate_csrf` takes a `token_key` parameter to specify the key stored in the session. (#206)
- `generate_csrf` takes a `url_safe` parameter to allow the token to be used in URLs. (#206)
- `form.data` can be accessed multiple times without raising an exception. (#248)
- File extension with multiple parts (`.tar.gz`) can be used in the `FileAllowed` validator. (#201)

## Version 0.12

Released 2015/07/09

- Abstract protect_csrf() into a separate method
- Update reCAPTCHA configuration
- Fix reCAPTCHA error handle

## Version 0.11

Released 2015/01/21

- Use the new reCAPTCHA API via #164.

## 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

## Version 0.10.2

Released 2014/09/03

- Update translation for reCaptcha via #146.

## 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.

## 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.

## Version 0.9.5

Released 2014/03/21

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

## 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

## Version 0.9.3

Released 2013/10/02

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

## 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.

## Version 0.9.1

Released 2013/8/21

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

## 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

## 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.

### 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)

### 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.

## Authors

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

### Contributors

People who send patches and suggestions:

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

Find more contributors on GitHub.

## BSD License

# Python Module Index

## f

# Index