

---

# **Django Concurrency Documentation**

*Release 2.4*

**Stefano Apostolico**

**Aug 04, 2022**



# CONTENTS

<b>1 Overview</b>	<b>1</b>
<b>2 How it works</b>	<b>3</b>
<b>3 Table Of Contents</b>	<b>5</b>
<b>4 Links</b>	<b>29</b>
<b>Index</b>	<b>31</b>



## OVERVIEW

django-concurrency is an optimistic locking library for Django Models

It prevents users from doing concurrent editing in Django both from UI and from a django command.

- easy to add to existing Models (just add *VersionField* )
- can be added with Django internal models (ie *auth.User* or *auth.Group*)
- handle http post and standard python code (ie. django management commands)
- complete test suite (*Test suite*)
- Admin integration. Handle *actions* and *list\_editable* (solves issue #11313)
- can handle external updates (see *TriggerVersionField*)



## HOW IT WORKS

### 2.1 Overview

**django-concurrency** works adding a `concurrency.fields.VersionField` to each model, each time a record is saved the version number changes (the algorithm used depends on the implementation of `concurrency.fields.VersionField` used (see *Fields*).

Each update is converted in the following SQL clause like:

```
UPDATE mymodel SET version=NEW_VERSION, ... WHERE id = PK AND version = VERSION_NUMBER
```



## TABLE OF CONTENTS

### 3.1 Install

Using pip:

```
pip install django-concurrency
```

Go to <https://github.com/saxix/django-concurrency> if you need to download a package or clone the repo.

**django-concurrency** does not need to be added into `INSTALLED_APPS` unless you want to run the tests or use the `templatetags` and/or `admin` integration

#### 3.1.1 Test suite

**django-concurrency** comes with a set of tests that can simulate different scenarios

- basic versioned model
- inherited model
- inherited model from abstract model
- inherited model from external project model
- django User model
- models with custom save
- proxy models
- admin actions

#### How to run the tests

```
$ pip install tox  
$ tox
```

## 3.2 Fields

- *VersionField*
- *IntegerVersionField*
- *AutoIncVersionField*
- *TriggerVersionField*
  - `trigger_name`
  - `triggers management command`
- *ConditionalVersionField*

### 3.2.1 VersionField

```
class concurrency.fields.VersionField(*args, **kwargs)
    Base class
```

### 3.2.2 IntegerVersionField

```
class concurrency.fields.IntegerVersionField(*args, **kwargs)
    Version Field that returns a “unique” version number for the record.
```

The version number is produced using `time.time() * 1000000`, to get the benefits of microsecond if the system clock provides them.

### 3.2.3 AutoIncVersionField

```
class concurrency.fields.AutoIncVersionField(*args, **kwargs)
    Version Field increment the revision number each commit
```

### 3.2.4 TriggerVersionField

```
class concurrency.fields.TriggerVersionField
```

This field use a database trigger to update the version field. Using this you can control external updates (ie using tools like phpMyAdmin, pgAdmin, SQLDeveloper). The trigger is automatically created during `syncdb()` or you can use the *triggers* management command.

Changed in version 1.0.

**Warning:** Before **django-concurrency** 1.0 two triggers per field were created, if you are upgrading you must manually remove old triggers and recreate them using *triggers* management command

### *trigger\_name*

New in version 1.0.

TriggerVersionField.**trigger\_name**

Starting from 1.0 you can customize the name of the trigger created. Otherwise for each *TriggerVersionField* will be created two triggers named:

```
'concurrency_[DBTABLENAME]_[FIELDNAME]'
```

**Warning:** Any name will be automatically prefixed with `concurrency_`

### *triggers management command*

Helper command to work with triggers:

- `list` : list existing triggers for each database
- `drop` : drop existing triggers
- `create` : create required triggers

example

```
sax@: (concurrency) django-concurrency [feature/triggers*] $ ./demo/manage.py
↳triggers create
DATABASE          TRIGGERS
default           concurrency_concurrency_triggerconcurrentmodel_u
```

## 3.2.5 ConditionalVersionField

New in version 1.1.

This field allow to configure which fields trigger the version increment so to limit the scope of the concurrency checks.

```
class User(models.Model):
    version = ConditionalVersionField()
    username = models.CharField(...)
    password = models.PasswordField(...)

    class ConcurrencyMeta:
        check_fields = ('username',)
```

## 3.3 ConcurrencyMiddleware

You can globally intercept *RecordModifiedError* adding *ConcurrencyMiddleware* to your `MIDDLEWARE_CLASSES`. Each time a *RecordModifiedError* is raised it goes up to the *ConcurrencyMiddleware* and the handler defined in `CONCURRENCY_HANDLER409` is invoked.

### Example

```
settings.py
```

```
MIDDLEWARE_CLASSES=('django.middleware.common.CommonMiddleware',
                    'concurrency.middleware.ConcurrencyMiddleware',
                    'django.contrib.sessions.middleware.SessionMiddleware',
                    'django.middleware.csrf.CsrfViewMiddleware',
                    'django.contrib.auth.middleware.AuthenticationMiddleware',
                    'django.contrib.messages.middleware.MessageMiddleware')

CONCURRENCY_HANDLER409 = 'demoproject.demoapp.views.conflict'
CONCURRENCY_POLICY = 2 # CONCURRENCY_LIST_EDITABLE_POLICY_ABORT_ALL
```

views.py

```
from diff_match_patch import diff_match_patch
from concurrency.views import ConflictResponse
from django.template import loader
from django.utils.safestring import mark_safe
from django.template.context import RequestContext

def get_diff(current, stored):
    data = []
    dmp = diff_match_patch()
    fields = current._meta.fields
    for field in fields:
        v1 = getattr(current, field.name, "")
        v2 = getattr(stored, field.name, "")
        diff = dmp.diff_main(unicode(v1), unicode(v2))
        dmp.diff_cleanupSemantic(diff)
        html = dmp.diff_prettyHtml(diff)
        html = mark_safe(html)
        data.append((field, v1, v2, html))
    return data

def conflict(request, target=None, template_name='409.html'):
    template = loader.get_template(template_name)
    try:
        saved = target.__class__._default_manager.get(pk=target.pk)
        diff = get_diff(target, saved)
    except target.__class__.DoesNotExist:
        saved = None
        diff = None

    ctx = RequestContext(request, {'target': target,
                                   'diff': diff,
                                   'saved': saved,
                                   'request_path': request.path})

    return ConflictResponse(template.render(ctx))
```

409.html

```
{% load concurrency %}
<table>
  <tr>
    <th>
      Field
    </th>
    <th>
      Current
```

(continues on next page)

(continued from previous page)

```

</th>
<th>
    Stored
</th>
<th>
    Diff
</th>

</tr>
<tr>
    {% for field, current, stored, entry in diff %}
        {% if not field.primary_key and not field|is_version %}
            <tr>
                <td>
                    {{ field.verbose_name }}
                </td>
                <td>
                    {{ current }}
                </td>
                <td>
                    {{ stored }}
                </td>
                <td>
                    {{ entry }}
                </td>
            </tr>
        {% endif %}
    {% endfor %}
</tr>
</table>

```

If you want to use `ConcurrencyMiddleware` in the admin and you are using `concurrency.admin.ConcurrentModelAdmin` remember to set your `ModelAdmin` to NOT use `concurrency.forms.ConcurrentForm`

```

from django import forms

class MyModelAdmin(ConcurrencyMiddleware):
    form = forms.ModelForm # overrides default ConcurrentForm

```

### 3.4 Admin Integration

- *Handle `list_editable`*
- *Check admin's action execution for concurrency*
- *Update existing actions templates to be managed by concurrency*

### 3.4.1 Handle `list_editable`

New in version 0.6.

**django-concurrency** is able to handle conflicts in the admin's changelist view when `ModelAdmin.list_editable` is enabled. To enable this feature simply extend your `ModelAdmin` from `ConcurrentModelAdmin` or use `ConcurrencyListEditableMixin`

**See also:**

*POLICY*

### 3.4.2 Check admin's action execution for concurrency

New in version 0.6.

Extend your `ModelAdmin` with `ConcurrencyActionMixin` or use `ConcurrentModelAdmin`

### 3.4.3 Update existing actions templates to be managed by concurrency

New in version 0.6.

You can use the `identity` filter to pass both `pk` and `version` to your `ModelAdmin`. Each time you use `{{ obj.pk }}` simply change to `{{ obj|identity }}`. So in the `admin/delete_selected_confirmation.html` will have:

```
{% for obj in queryset %}
<input type="hidden" name="{{ action_checkbox_name }}" value="{{ obj|identity }}" />
{% endfor %}
```

## 3.5 API

- *Forms*
  - *ConcurrentForm*
  - *VersionWidget*
- *Exceptions*
  - *VersionChangedError*
  - *RecordModifiedError*
  - *InconsistencyError*
  - *VersionError*
- *Admin*
  - *ConcurrentModelAdmin*
  - *ConcurrencyActionMixin*
  - *ConcurrencyListEditableMixin*
- *Middleware*

- *ConcurrencyMiddleware*
- *concurrency.views.conflict*
- *Helpers*
  - `apply_concurrency_check()`
  - `disable_concurrency()`
    - \* *examples*
  - `concurrency_disable_increment()`
- *Templatetags*
  - `identity`
  - `version`
  - `is_version`
- *Triggers*
  - *TriggerFactory*
- *Test Utilities*
  - *ConcurrencyTestMixin*
- *Signing*

### 3.5.1 Forms

#### ConcurrentForm

```
class concurrency.forms.ConcurrentForm(data=None, files=None, auto_id='id_%s', prefix=None, initial=None, error_class=<class 'django.forms.utils.ErrorList'>, label_suffix=None, empty_permitted=False, instance=None, use_required_attribute=None, renderer=None)
```

Simple wrapper to `ModelForm` that try to mitigate some concurrency error. Note that is always possible have a `RecordModifiedError` in `model.save()`. Statistically `form.clean()` should catch most of the concurrent editing, but is good to catch `RecordModifiedError` in the view too.

#### VersionWidget

```
class concurrency.forms.VersionWidget(attrs=None)
```

Widget that show the revision number using `<div>`

Usually `VersionField` use `HiddenInput` as `Widget` to minimize the impact on the forms, in the Admin this produce a side effect to have the label `Version` without any value, you should use this widget to display the current revision number

## 3.5.2 Exceptions

### VersionChangedError

```
class concurrency.exceptions.VersionChangedError(message, code=None,
                                                  params=None)
```

```
class concurrency.exceptions.RecordModifiedError
```

### RecordModifiedError

```
class concurrency.exceptions.RecordModifiedError(*args, **kwargs)
```

### InconsistencyError

Changed in version 0.7.

**Warning:** removed in 0.7

```
class concurrency.exceptions.InconsistencyError
```

### VersionError

```
class concurrency.exceptions.VersionError(message=None, code=None, params=None,
                                          *args, **kwargs)
```

## 3.5.3 Admin

### ConcurrentModelAdmin

```
class concurrency.admin.ConcurrentModelAdmin(model, admin_site)
```

**Warning:** If you customize `fields` or `fieldsets` remember to add `version` field to the list. (See issue [issue #81](#))

### ConcurrencyActionMixin

```
class concurrency.admin.ConcurrencyActionMixin
```

## ConcurrencyListEditableMixin

```
class concurrency.admin.ConcurrencyListEditableMixin
```

## 3.5.4 Middleware

```
class concurrency.middleware.ConcurrencyMiddleware
```

### ConcurrencyMiddleware

See also:

*ConcurrencyMiddleware*

```
class concurrency.middleware.ConcurrencyMiddleware (get_response=None)
```

Intercept *RecordModifiedError* and invoke a callable defined in `CONCURRENCY_HANDLER409` passing the request and the object.

### concurrency.views.conflict

```
concurrency.views.conflict (request, target=None, template_name='409.html')
409 error handler.
```

#### Parameters

- **request** – Request
- **template\_name** – *409.html*
- **target** – The model to save

## 3.5.5 Helpers

### *apply\_concurrency\_check()*

New in version 0.4.

Changed in version 0.8.

Add concurrency check to existing classes.

---

**Note:** With Django 1.7 and the new migrations management, this utility does not work anymore. To add concurrency management to a external Model, you need to use a migration to add a *VersionField* to the desired Model.

---

---

**Note:** See `demo.auth_migrations` for a example how to add *IntegerVersionField* to `auth.Group`

---

```
operations = [
    # add version to django.contrib.auth.Group
    migrations.AddField(
        model_name='Group',
        name='version',
```

(continues on next page)

(continued from previous page)

```
        field=IntegerVersionField(help_text=b'Version', default=1),
    ),
]
```

and put in your settings.py

```
MIGRATION_MODULES = {
    ...
    ...
    'auth': '<new.migration.package>',
}
```

### *disable\_concurrency()*

New in version 0.6.

Context manager to temporary disable concurrency checking.

Changed in version 0.9.

Starting from version 0.9, *disable\_concurrency* can disable both at Model level or instance level, depending on the passed object. Passing Model is useful in django commands, load data or fixtures, where instance should be used by default

Changed in version 1.0.

Is now possible use *disable\_concurrency* without any argument to disable concurrency on any Model. This features has been developed to be used in django commands

Changed in version 1.1.

Does not raise an exception if a model not under concurrency management is passed as argument.

### examples

```
@disable_concurrency()
def recover_view(self, request, version_id, extra_context=None):
    return super().recover_view(request,
                                version_id,
                                extra_context)
```

```
def test_recover():
    deleted_list = revisions.get_deleted(ReversionConcurrentModel)
    delete_version = deleted_list.get(id=5)

    with disable_concurrency(ReversionConcurrentModel):
        deleted_version.revert()
```

### *concurrency\_disable\_increment()*

New in version 1.1.

Context manager to temporarily disable version increment. Concurrent save is still checked but no version increment is triggered, this creates ‘shadow saves’.

It accepts both a Model or an instance as target.

## 3.5.6 Templatetags

### *identity*

`concurrency.templatetags.concurrency.identity(obj)`  
returns a string representing “<pk>,<version>” of the passed object

### *version*

`concurrency.templatetags.concurrency.version(obj)`  
returns the value of the VersionField of the passed object

### *is\_version*

`concurrency.templatetags.concurrency.is_version(field)`  
returns True if passed argument is a VersionField instance

## 3.5.7 Triggers

### TriggerFactory

New in version 2.3.

**class** `concurrency.triggers.TriggerFactory(connection)`  
Abstract Factory class to create triggers. Implementations need to set the following attributes  
*update\_clause*, *drop\_clause* and *list\_clause*

Those will be formatted using standard python *format()* as:

```
self.update_clause.format(trigger_name=field.trigger_name,
                           opts=field.model._meta,
                           field=field)
```

So as example:

```
update_clause = """CREATE TRIGGER {trigger_name}
AFTER UPDATE ON {opts.db_table}
BEGIN UPDATE {opts.db_table}
SET {field.column} = {field.column}+1
WHERE {opts.pk.column} = NEW.{opts.pk.column};
END;
"""
```

See also:

`TRIGGERS_FACTORY`

## 3.5.8 Test Utilities

### ConcurrencyTestMixin

**class** `concurrency.utils.ConcurrencyTestMixin`

Mixin class to test Models that use *VersionField*

this class offer a simple test scenario. Its purpose is to discover some conflict in the *save()* inheritance:

```
from concurrency.utils import ConcurrencyTestMixin
from myproject.models import MyModel

class MyModelTest(ConcurrencyTestMixin, TestCase):
    concurrency_model = TestModel0
    concurrency_kwargs = {'username': 'test'}
```

## 3.5.9 Signining

New in version 0.5.

*VersionField* is ‘displayed’ in the Form using an `django.forms.HiddenInput` widget, anyway to be sure that the version is not tampered with, its value is *signed*. The default *VersionSigner* is `concurrency.forms.VersionFieldSigner` that simply extends `django.core.signing.Signer`. If you want change your Signer you can set `CONCURRENCY_FIELD_SIGNER` in your settings

`mysigner.py`

```
class DummySigner():
    """ Dummy signer that simply returns the raw version value. (Simply do,
    ↪not sign it) """
    def sign(self, value):
        return smart_str(value)

    def unsign(self, signed_value):
        return smart_str(signed_value)
```

`settings.py`

```
CONCURRENCY_FIELD_SIGNER = "myapp.mysigner.DummySigner"
```

## 3.6 Settings

Here’s a full list of all available settings, in alphabetical order, and their default values.

---

**Note:** Each entry **MUST** have the prefix `CONCURRENCY_` when used in your settings.py

---

### 3.6.1 AUTO\_CREATE\_TRIGGERS

New in version 2.3.

Default: `True`

If `true` automatically create triggers. To manually create triggers set `CONCURRENCY_AUTO_CREATE_TRIGGERS=False` and use *triggers management command* management command or create them manually using your DB client.

Note:: This flag deprecates `MANUAL_TRIGGERS`

### 3.6.2 ENABLED

New in version 0.10.

Default: `True`

enable/disable concurrency

### 3.6.3 CALLBACK

Changed in version 0.7.

Default: `concurrency.views.callback`

Handler invoked when a conflict is raised. The default implementation simply raise *RecordModifiedError*

Can be used to display the two version of the record and let the user to force the update or merge the values.

### 3.6.4 FIELD\_SIGNER

New in version 0.5.

Default: `concurrency.forms.VersionFieldSigner`

Class used to sign the version numbers.

**See also:**

*Signining*

### 3.6.5 HANDLER409

New in version 0.6.

Default: `concurrency.views.conflict`

Handler to intercept *RecordModifiedError* into *ConcurrencyMiddleware*. The default implementation (`concurrency.views.conflict`) renders `409.html` while passing into the context the object that is going to be saved (target)

**See also:**

*ConcurrencyMiddleware*

### 3.6.6 IGNORE\_DEFAULT

New in version 1.2.

Changed in version 1.5.

Default: `True`

**See also:**

`VERSION_FIELD_REQUIRED`

### 3.6.7 VERSION\_FIELD\_REQUIRED

New in version 1.5.

Default: `True`

Determines whether version number is mandatory in any save operation. Setting this flag to `False` can cause omitted version numbers to pass concurrency checks.

### 3.6.8 MANUAL\_TRIGGERS

New in version 1.0.

Deprecated since version 2.3.

Default: `False`

If false do not automatically create triggers, you can create them using *triggers management command* management command or manually using your DB client.

### 3.6.9 POLICY

Changed in version 0.7.

Default: `CONCURRENCY_LIST_EDITABLE_POLICY_SILENT`

#### `CONCURRENCY_LIST_EDITABLE_POLICY_SILENT`

Used by admin's integrations to handle `list_editable` conflicts. Do not save conflicting records, continue and save all non-conflicting records, show a message to the user

#### `CONCURRENCY_LIST_EDITABLE_POLICY_ABORT_ALL`

Used by admin's integrations to handle `list_editable`. Stop at the first conflict and raise *RecordModifiedError*. Note that if you want to use *ConcurrencyMiddleware* based conflict management you must set this flag.

**See also:**

*Handle list\_editable, ConcurrencyMiddleware*

### 3.6.10 TRIGGERS\_FACTORY

New in version 2.3.

Default:

```
{'postgresql': "concurrency.triggers.PostgreSQL",
 'mysql': "concurrency.triggers.MySQL",
 'sqlite3': "concurrency.triggers.Sqlite3",
 'sqlite': "concurrency.triggers.Sqlite3",
 }
```

dict to customise *TriggerFactory*. Use this to customise the SQL clause to create triggers.

## 3.7 Cookbook

- *Unable to import data ?*
- *Add version management to new models*
- *Add version management to Django and/or plugged in applications models*
- *Test Utilities*
- *Recover deleted record with django-reversion*

### 3.7.1 Unable to import data ?

Sometimes you need to temporary disable concurrency (ie during data imports)

**Temporary disable per Model**

```
from concurrency.api import disable_concurrency

with disable_concurrency(instance):
    Model.object
```

### 3.7.2 Add version management to new models

models.py

```
from concurrency.fields import IntegerVersionField

class ConcurrentModel( models.Model ):
    version = IntegerVersionField( )
```

tests.py

```
a = ConcurrentModel.objects.get(pk=1)
b = ConcurrentModel.objects.get(pk=1)
a.save()
b.save() # this will raise ``RecordModifiedError``
```

### 3.7.3 Add version management to Django and/or plugged in applications models

Changed in version 0.8.

Concurrency can work even with existing models, anyway if you are adding concurrency management to an existing database remember to edit the database's tables:

`your_app.models.py`

```
from django.contrib.auth import User
from concurrency.api import apply_concurrency_check

apply_concurrency_check(User, 'version', IntegerVersionField)
```

If used with Django>=1.7 remember to create a custom migration.

### 3.7.4 Test Utilities

*ConcurrencyTestMixin* offer a very simple test function for your existing models

```
from concurrency.utils import ConcurrencyTestMixin
from myproject.models import MyModel

class MyModelTest(ConcurrencyTestMixin, TestCase):
    concurrency_model = TestModel0
    concurrency_kwargs = {'username': 'test'}
```

### 3.7.5 Recover deleted record with django-reversion

Recovering deleted records with [django-reversion](#) produces a `RecordModifiedError`, because both *pk* and *version* are present in the object, and **django-concurrency** tries to load the record (that does not exist), which raises `RecordModifiedError` then.

To avoid this simply disable concurrency, by using a mixin:

```
class ConcurrencyVersionAdmin(reversion.admin.VersionAdmin):

    @disable_concurrency()
    def revision_view(self, request, object_id, version_id, extra_context=None):
        return super().revision_view(request, object_id, version_id, extra_
↪context=None)

    @disable_concurrency()
    def recover_view(self, request, version_id, extra_context=None):
        return super().recover_view(request, version_id, extra_context)
```

## 3.8 Changelog

This section lists the biggest changes done on each release.

- *Release 2.4*
- *Release 2.3*
- *Release 2.2*
- *Release 2.1.1*
- *Release 2.1 ( not released on pypi)*
- *Release 2.0*
- *Release 1.4 (13 Sep 2016)*
- *Release 1.3.2 (10 Sep 2016)*
- *Release 1.3.1 (15 Jul 2016)*
- *Release 1.3 (15 Jul 2016)*
- *Release 1.2 (05 Apr 2016)*
- *Release 1.1 (13 Feb 2016)*
- *Release 1.0.1*
- *Release 1.0*
- *Release 0.9*
- *Release 0.8.1*
- *Release 0.8*
- *Release 0.7.1*
- *Release 0.7*
- *Release 0.6.0*
- *Release 0.5.0*
- *Release 0.4.0*
- *Release 0.3.2*

### 3.8.1 Release 2.4

- add support Django 4
- add support Python 3.10

### 3.8.2 Release 2.3

- Removes code producing `DeprecationError`
- add `AUTO_CREATE_TRIGGERS` and deprecate `MANUAL_TRIGGERS`
- add support for Postgres 13
- add ability to customise SQL to create triggers `TRIGGERS_FACTORY`

### 3.8.3 Release 2.2

- drop support django<3.0
- drop support Python<3.6

### 3.8.4 Release 2.1.1

- fixes packaging

### 3.8.5 Release 2.1 ( not released on pypi)

- drop support Python < 3.5
- add support Django 2.2 / 3.0
- drop support Django < 1.11

### 3.8.6 Release 2.0

- drop official support to Django < 1.10
- add support Django 2.1
- removed deprecated api `concurrency_check`
- **BACKWARD INCOMPATIBLE:** version field is now mandatory in any save operation. Use `VERSION_FIELD_REQUIRED=False` to have the old behaviour.
- `disable_concurrency` now has `start()`, `finish()` to be called as command

### 3.8.7 Release 1.4 (13 Sep 2016)

- closes [issue #81](#). Add docs and check.
- fixes [issue #80](#). (thanks Naddiseo for the useful support)
- Django 1.11 compatibility
- some minor support for Django 2.0

### 3.8.8 Release 1.3.2 (10 Sep 2016)

- fixes bug in `ConditionalVersionField` that produced 'maximum recursion error' when a model had a `ManyToManyField` with a field to same model (self-relation)

### 3.8.9 Release 1.3.1 (15 Jul 2016)

- just packaging

### 3.8.10 Release 1.3 (15 Jul 2016)

- drop support for Python < 3.3
- add support `Django>=1.10`
- change license
- fixes [issue #36](#). (thanks claytondaley)
- new `IGNORE_DEFAULT` to ignore default version number

### 3.8.11 Release 1.2 (05 Apr 2016)

- better support for django 1.9 (`TemplateDoesNotExist` is now in `django.template.exceptions`)
- improved error message in `ConcurrencyListEditableMixin` [issue #63](#) [issue #64](#)
- fixes [issue #61](#). Error in `ConditionalVersionField` (thanks ticosax)
- fixes `skipif` test in `pypy`

### 3.8.12 Release 1.1 (13 Feb 2016)

- drop support for `django<1.7`
- add support for `pypy`
- new `concurrency.fields.ConditionalVersionField`
- new decorator `concurrency.api.concurrency_disable_increment`
- `concurrency.api.disable_concurrency` is now a noop if applied to a model not under concurrency management

### 3.8.13 Release 1.0.1

- fixes [issue #56](#) “Can’t upgrade django-concurrency to 1.0” (thanks oppianmatt).

### 3.8.14 Release 1.0

- **BACKWARD INCOMPATIBLE::** dropped support for Django prior 1.6
- code clean
- fixes [issue #54](#) “Incorrect default for IntegerVersionField” (thanks vmspike).
- fixes [issue #53](#). updates Documentation
- `disable_concurrency()` can now disable concurrency in any model
- `disable_concurrency()` is now also a decorator
- **BACKWARD INCOMPATIBLE::** removed custom backends. `TriggerVersionField` can be used with standard Django
- new way to create triggers (thanks Naddiseo)
- new trigger code
- new `TriggerVersionField.check`.
- new `TriggerVersionField.trigger_name`.
- new `CONCURRENCY_ENABLED` to fully disable concurrency
- new `CONCURRENCY_MANUAL_TRIGGERS` to disable triggers auto creation fixes [issue #41](#) (thanks Naddiseo)

### 3.8.15 Release 0.9

- Django 1.8 compatibility
- python 3.4 compatibility
- **BACKWARD INCOMPATIBLE** `disable_concurrency()` works differently if used with classes or instances
- better support for external Models (models that are part of plugged-in applications)
- fixes issue with `TriggerVersionField` and Proxy Models (thanx Richard Eames)

### 3.8.16 Release 0.8.1

- avoid to use concurrency when selecting all items (`select_across`)

### 3.8.17 Release 0.8

- Django 1.7 compatibility
- fixes typo in `delete_selected_confirmation.html` template
- python 3.2/3.3 compatibility

### 3.8.18 Release 0.7.1

- **backward compatibility updates. Do not check for concurrency if 0 is passed as version value** (ie. no value provided by the form)

### 3.8.19 Release 0.7

- new `concurrency.fields.TriggerVersionField`
- start using pytest
- moved tests outside main package
- new protocol see:ref:protocols
- it's now possible disable concurrency in Models that extends concurrency enabled models
- fixed [issue #23](#) (thanks matklad)
- new `USE_SELECT_FOR_UPDATE`

### 3.8.20 Release 0.6.0

- new `disable_concurrency()` context manager
- added documentation for `concurrency.middleware.ConcurrencyMiddleware`
- **BACKWARD INCOMPATIBLE** Fixed typo: `CONCURRECY_SANITY_CHECK` now `CONCURRENCY_SANITY_CHECK`
- added `disable_sanity_check` context manager
- added configuration
- check admin actions for concurrent deletion
- added concurrency check for admin's `Handle list_editable`

### 3.8.21 Release 0.5.0

- python 3.x compatibility
- new `CONCURRENCY_FIELD_SIGNER`

### 3.8.22 Release 0.4.0

- start deprecation of `concurrency.core.VersionChangedError`, `concurrency.core.RecordModifiedError`, `concurrency.core.InconsistencyError`, moved in `concurrency.exceptions`
- start deprecation of `concurrency.core.apply_concurrency_check`, `concurrency.core.concurrency_check` moved in `concurrency.api`
- added `CONCURRENCY_SANITY_CHECK` settings entry
- signing of version number to avoid tampering (*ConcurrentForm*)
- added *ConcurrencyTestMixin* to help test on concurrency managed models
- changed way to add concurrency to existing models (*apply\_concurrency\_check()*)
- fixed [issue #4](#) (thanks FrankBie)
- removed `RandomVersionField`
- new *concurrency\_check*
- added *ConcurrentForm* to mitigate some concurrency conflict
- `select_for_update` now executed with `nowait=True`
- removed some internal methods, to avoid unlikely but possible name clashes

### 3.8.23 Release 0.3.2

- fixed [issue #3](#) (thanks pombredanne)
- fixed [issue #1](#) (thanks mbrochh)

## 3.9 FAQ

- *I use Django-Rest-Framework and `django-concurrency` seems do not work*
- *Just added `django-concurrency` to existing project and it does not work*
- *South support ?*
- *How is managed `update_fields`*

### 3.9.1 I use Django-Rest-Framework and `django-concurrency` seems do not work

Use `CONCURRENCY_IGNORE_DEFAULT` accordingly or be sure that serializer does not set `0` as initial value

### 3.9.2 Just added django-concurrency to existing project and it does not work

Check that your records do not have 0 as version number and use `CONCURRENCY_IGNORE_DEFAULT` accordingly

### 3.9.3 South support ?

South support has been removed after version 1.0 when Django <1.6 support has been removed as well.

If needed add these lines to your `models.py`:

```
from south.modelsinspector import add_introspection_rules
add_introspection_rules([], [ "^concurrency\.fields\.IntegerVersionField" ])
```

### 3.9.4 How is managed `update_fields`

It is possible to use `save(update_fields=...)` parameter without interfere with the concurrency check algorithm

```
x1 = MyModel.objects.create(name='abc')
x2 = MyModel.objects.get(pk=x1.pk)

x1.save()
x2.save(update_fields=['username']) # raise RecordModifiedError
```

anyway this will NOT raise any error

```
x1 = MyModel.objects.create(name='abc')
x2 = MyModel.objects.get(pk=x1.pk)

x1.save(update_fields=['username']) # skip update version number
x2.save() # saved
```

---

**Note:** `TriggerVersionField` will be always updated

---



## LINKS

- Project home page: <https://github.com/saxix/django-concurrency>
- Issue tracker: <https://github.com/saxix/django-concurrency/issues?sort>
- Download: <http://pypi.python.org/pypi/django-concurrency/>
- Docs: <http://readthedocs.org/docs/django-concurrency/en/latest/>



## A

AutoIncVersionField (class in *concurrency.fields*),  
6

## C

CONCURECY\_MANUAL\_TRIGGERS  
setting, 18  
concurrency.exceptions.InconsistencyError  
(built-in class), 12  
concurrency.exceptions.RecordModifiedError  
(built-in class), 12  
concurrency.fields.TriggerVersionField  
(built-in class), 6  
concurrency.middleware.ConcurrencyMiddleware  
(built-in class), 13  
CONCURECY\_AUTO\_CREATE\_TRIGGERS  
setting, 16  
CONCURECY\_CALLBACK  
setting, 17  
CONCURECY\_ENABLED  
setting, 17  
CONCURECY\_FIELD\_SIGNER  
setting, 17  
CONCURECY\_HANDLER409  
setting, 17  
CONCURECY\_IGNORE\_DEFAULT  
setting, 17  
CONCURECY\_POLICY  
setting, 18  
CONCURECY\_TRIGGERS\_FACTORY  
setting, 18  
CONCURECY\_VERSION\_FIELD\_REQUIRED  
setting, 18  
ConcurrencyActionMixin (class in *concurrency.admin*), 12  
ConcurrencyListEditableMixin (class in *concurrency.admin*), 13  
ConcurrencyMiddleware (class in *concurrency.middleware*), 13  
ConcurrencyTestMixin (class in *concurrency.utils*), 16  
ConcurrentForm (class in *concurrency.forms*), 11

ConcurrentModelAdmin (class in *concurrency.admin*), 12  
conflict() (in module *concurrency.views*), 13

## I

identity  
template filter, 15  
identity() (in module *concurrency.templatetags.concurrency*), 15  
IntegerVersionField (class in *concurrency.fields*),  
6  
is\_version  
template filter, 15  
is\_version() (in module *concurrency.templatetags.concurrency*), 15

## M

MANUAL\_TRIGGERS  
setting, 18

## R

RecordModifiedError (class in *concurrency.exceptions*), 12

## S

setting  
CONCURECY\_MANUAL\_TRIGGERS, 18  
CONCURECY\_AUTO\_CREATE\_TRIGGERS, 16  
CONCURECY\_CALLBACK, 17  
CONCURECY\_ENABLED, 17  
CONCURECY\_FIELD\_SIGNER, 17  
CONCURECY\_HANDLER409, 17  
CONCURECY\_IGNORE\_DEFAULT, 17  
CONCURECY\_POLICY, 18  
CONCURECY\_TRIGGERS\_FACTORY, 18  
CONCURECY\_VERSION\_FIELD\_REQUIRED,  
18  
MANUAL\_TRIGGERS, 18  
TRIGGERS\_FACTORY, 18

## T

template filter

- identity, 15
- is\_version, 15
- version, 15
- trigger\_name (*TriggerVersionField attribute*), 7
- TriggerFactory (*class in concurrency.triggers*), 15
- TRIGGERS\_FACTORY
  - setting, 18

## V

- version
  - template filter, 15
- version() (*in module concurrency.templatetags.concurrency*), 15
- VersionChangedError (*class in concurrency.exceptions*), 12
- VersionError (*class in concurrency.exceptions*), 12
- VersionField (*class in concurrency.fields*), 6
- VersionWidget (*class in concurrency.forms*), 11