django-internationalflavor Documentation

Release 0.1

Ralph Broenink

Apr 04, 2022

Contents

`django-internationalflavor` is born to complement the `django-localflavor` package. While localflavor is awesome when you are making a localized app, the reality is that you often need to accommodate for users from multiple countries. While Django has great support for internationalization and localization, there is no package that helps you store data from all over the world. This package aims to fill the gap and provides fields that are designed for use in almost every country, while enforcing consistent data types.

Warning: This module is far from complete and may or may not break your existing installation. I'm still working on it, so please bear with me. Pull requests are welcome!

CHAPTER 1

Available data types

1.1 Countries

Although for the most countries, sovereignty is not disputed, there are numerous cases where the definition of country is not clear. Using the full ISO 3166-1 country list may seem the most political correct method, but this would also include uninhabited wasteland such as Antartica and Norfolk Island. Furthermore, some territories are more or less part of another country, such as Greenland that is part of Denmark.

To accommodate for these differences, this module chooses to include the following countries by default:

- All UN member states
- All UN disputed states (except for when no ISO 3166-1 country code is available)
- All UN observer states
- All UN non-self-governing states (this list is rather arbitrary, unfortunately)

Both fields provided by this module provide ways to manually add or exclude items from this list, as long as they are valid ISO 3166-1 alpha-2 codes.

The list of ISO 3166-1 countries included with this module uses the common names of countries, rather than using their official names.

Note: While this module tries to provide some sort of equal ground most people would agree on, the choices that have been made may be grounds for disputes on sovereignty. First of all, I'm sorry if the choices that have been made make you feel uncomfortable. Second of all, you are not required to use the provided default list and you can easily include or exclude countries to your liking. And finally, you are welcome to open a ticket (or pull request) on Github, but please keep it civilized and try to maintain a unbiased position.

---

class internationalflavor.countries.models.**CountryField**(*countries=None*, *exclude=None*, **args*, ***kwargs*)

A model field that allows users to choose their country. By default, it lists all countries recognized by the UN, but

using the `countries` attribute you can specify your own set of allowed countries. Use `exclude` to exclude specific countries.

class internationalflavor.countries.forms.**CountryFormField**(*countries=None*, *exclude=None*, **args*, ***kwargs*)

A form field that allows users to choose their country. By default, it lists all countries recognized by the UN, but using the `countries` attribute you can specify your own set of allowed countries. Use `exclude` to exclude specific countries.

1.1.1 Data constants

You can use the following constants to specify your own set of available countries.

internationalflavor.countries.data.**UN_MEMBER_STATES = (...)**

List of UN Member States Source: https://www.un.org/en/member-states/index.html

internationalflavor.countries.data.**UN_OBSERVER_STATES = (...)**

List of UN Observer States Source: http://www.un.org/en/members/nonmembers.shtml

internationalflavor.countries.data.**UN_DISPUTED_STATES = (...)**

Disputed UN states.

Source: https://en.wikipedia.org/wiki/List_of_sovereign_states#List_of_states

No ISO 3166-1 code has yet been assigned, and thus not included, for:

- Abkhazia
- Nagorno-Karabakh
- Northern Cyprus
- Sahrawi Arab Democratic Republic
- Somaliland
- South Ossetia
- Transnistria

Although Kosovo has no ISO 3166-1 code either, it is generally accepted to be XK temporarily; see http://ec.europa.eu/budget/contracts_grants/info_contracts/inforeuro/inforeuro_en.cfm or the CLDR

internationalflavor.countries.data.**UN_NON_SELF_GOVERNING_STATES = (...)**

List of the (rather arbitrary) UN non-self-governing states

Source: https://www.un.org/dppa/decolonization/en/nsgt

internationalflavor.countries.data.**UN_RECOGNIZED_COUNTRIES = (...)**

Combined list of all UN_* data constants.

internationalflavor.countries.data.**IOC_RECOGNIZED_COUNTRIES = (...)**

List of countries as defined by IOC.

Source: https://en.wikipedia.org/wiki/Comparison_of_IOC,_FIFA,_and_ISO_3166_country_codes

1.1.2 Comparison with other packages

django-countries This module has a more elaborate `CountryField`. It returns `Country` objects instead of ISO 3166-1 alpha-2 codes that allow easy access to the full name of a country or its country flag. However,

it only provides the basic ISO country code list, with official country names (rather than using their common names).

See also:

UN member states List of UN member states.

UN observer states List of UN observer states

UN non-self-governing states List of UN non-self-governing states

Wikpedia: List of ISO 3166-1 alpha-2 codes List of all ISO 3166-1 alpha-2 codes.

Wikipedia: List of sovereign states List of disputed sovereign states.

Wikipedia: Comparison of IOC ... country codes List of countries as recognized by the IOC.

1.2 IBAN / BIC

Most countries over the world use IBAN for international payments. Starting at August 1, 2014, the European Union has mandated that all its member countries must use IBAN for domestic and international transactions. Even if your country does not require IBAN for domestic transactions, it may be a good idea to use and store IBANs anyway. This allows you to handle bank account numbers from different countries.

1.2.1 IBAN

class internationalflavor.iban.validators.**IBANValidator**(*countries=None*, *exclude=None*, *sepa_only=False*, *accept_experimental=False*)

Validator for checking whether a given IBAN is valid. An IBAN consists of up to 34 alphanumeric characters, where the first two characters indicate a country code, the third and fourth indicate a checksum and the rest of the IBAN are localized characters (the so-called BBAN).

Parameters

- **countries** – If set, the list of source countries will be limited to the provided list. Otherwise, all available IBANs are included (with the exception of experimental IBANs if `accept_experimental` is not set).
- **exclude** – You can use this parameter to exclude items from the list of countries.
- **sepa_only** (*bool*) – By default, all countries are allowed. If you want to reduce the list of countries to the list of SEPA countries (i.e. Single European Payments Area), for instance if you are an European company wanting to perform direct debits, you can set this to True. This is equivalent to setting the exclude list to all countries without SEPA.
- **accept_experimental** (*bool*) – By default, this validator will validate any IBAN that is recognized by the SWIFT organization, but SWIFT has specified a few additional IBAN formats and defined them as 'experimental'. By setting this parameter to True, these extensions are also allowed.

> **Warning:** The validation of the experimental numbers may be wrong for some countries, as only their length is published by the SWIFT organization

class internationalflavor.iban.models.**IBANField**(*countries=None*, *exclude=None*, *sepa_only=False*, *accept_experimental=False*, **args*, ***kwargs*)

A model field that applies the *validators.IBANValidator* and is represented by a *forms.IBANFormField*. The arguments are equal to those of the validator.

Example:

```
from django.db import models
from internationalflavor.iban import IBANField

class MyModel(models.Model):
    iban = IBANField(countries=['NL', 'BE'])
```

This field is an extension of a CharField.

class internationalflavor.iban.forms.**IBANFormField**(*countries=None*, *exclude=None*, *sepa_only=False*, *accept_experimental=False*, **args*, ***kwargs*)

A form field that applies the *validators.IBANValidator*. The arguments are equal to those of the validator.

This field represents the data in 4-character blocks, but stores it internally without any formatting.

1.2.2 BIC

class internationalflavor.iban.validators.**BICValidator**

class internationalflavor.iban.models.**BICField**(**args*, ***kwargs*)

A model field that applies the *validators.BICValidator* and is represented by a *forms.BICFormField*.

This field is an extension of a CharField.

class internationalflavor.iban.forms.**BICFormField**(**args*, ***kwargs*)

A form field that applies the *validators.BICValidator*.

1.2.3 Comparison with other packages

localflavor Both an `IBANField` and a `BICField` are provided by this module. Although `internationalflavor` and `localflavor` have different approaches to validation, if you are already using `localflavor` and do not need any of the other fields provided by `internationalflavor`, you may be better off choosing `localflavor` (and vice versa).

django-iban The validation in this module is equal to the *localflavor* validation. The author of this package is seeking to retire his package, so it may be best to not use this package in new projects.

See also:

IBAN Registry The official IBAN format registry of SWIFT.

IBAN Structure Additional IBAN formats listed as experimental

Wikipedia: International Bank Account Number More information on IBAN

1.3 Language

Django already has excellent support for internationalization, but there's currently no built-in field to store the user's language (though there is middleware that can take the user's locale from the browser). This module provides a field that allows you to store the user's language in his profile.

All languages defined in the CLDR are defined in this module. Note that CLDR will use names such as `zh-Hans`, though we normalize that to all-lowercase, i.e. `zh-hans`, as this is how Django stores language codes.

class internationalflavor.language.models.**LanguageField**(*languages=None*, *exclude=None*, **args*, ***kwargs*)

A model field that allows users to choose their language. By default, all languages that are defined in your settings file are available. Use the `langagues` argument to specify your own langagues, and use `exclude` to exclude specific languages.

class internationalflavor.language.forms.**LanguageFormField**(*languages=None*, *exclude=None*, **args*, ***kwargs*)

1.3.1 Middlware

As a convenience, this module provides a middleware class that ensures your user's language is applied across your app. If you customize your user model to include a `language` field, e.g.:

```
class MyUser(auth_base.AbstractBaseUser, auth.PermissionsMixin)::
    language = LanguageField()
```

You can use it by adding `internationalflavor.language.middleware.UserLanguageMiddleware` to your `MIDDLEWARE` setting:

```
MIDDLEWARE = [
    ...
    'django.middleware.locale.LocaleMiddleware',
    'internationalflavor.language.middleware.UserLanguageMiddleware',
]
```

1.4 Names

This module contains some utilities for storing names. It is not a one-size-fits-all solution, as it is simply not possible to capture all the world's name formats in a single module.

1.4.1 Utilities

internationalflavor.names.utils.**split_name**(*name*, *scheme=None*, ***kwargs*)

Splits a name in several parts. Useful for when you are working with applications that store names in separate parts, and with applications that don't.

This utility method works on several schemes, the default being None, simply splitting a name. The result depends on the scheme you choose. Additionally, you can specify whether you want honorific titles returned. This is not supported by all schemes.

> **Warning:** There is no fool-proof method to split names that works in all edge cases correctly. If you need to have names split, simply ask them to be entered in split form. This method is best-effort only and provides the guarantee that it will make mistakes.

A scheme may support additional arguments, these are specified below.

The following schemes are supported:

- **None** [split a name in two parts, simply splitting on the first whitespace, ignoring honorific titles etc.] returns a tuple of (first_name, last_name), when the name can not be split, it is put in the last_name.

  extra kwarg: long_first : if set, a long first name is preferred over a long last name.

- **NL** [split a name in three parts, splitting including the Dutch 'voorvoegsel' or 'tussenvoegsel',] ignores honorific titles etc. When no such voorvoegsel exists, it falls back to the None scheme.

Using stalemate you can set whether you prefer longer first names or longer last names.

`internationalflavor.names.utils.`**`join_name`**(**parts*)

Joins a name. This is the inverse of split_name, but x == join_name(split_name(x)) does not necessarily hold.

Joining a name may also be subject to different schemes, but the most basic implementation is just joining all parts with a space.

1.5 Time zones

Time zone support was added in Django 1.4, allowing you to store and handle local dates and times. It is highly recommended to store your time objects timezone-aware. Django handles this for you, but it does not provide a way for a user to save their timezone. This module adds support for this.

This module highly recommends you install `pytz` along with it, but it is not required. By default, the module uses the set of common timezones as reported by `pytz.common_timezones`. If this is not available, the set as provided by the CLDR is used instead.

This module provides two different types to store your timezones. The *`models.TimezoneField`* model field uses the default timezone database and the default form field requests the user to select a timezone based on their location. This is the recommended approach and most consistent with how timezones are stored internally. You are ensured that you will always be in the same timezone, and when a location changes its timezone, it is automatically applied.

The other option is to use metazones. These are defined by CLDR and often span multiple cities, i.e. you don't pick the city and let the system figure out which timezone you are in based on the location, but let the user pick a timezone. This has the advantage that it is more intuitive for the user and results in a much shorter dropdown menu, but has the obvious disadvantage that it does not update automatically when a timezone is changed. It is also unsuitable for accurate historic dates.

The model field uses a `datetime.tzinfo` Python object as representation, unless *use_tzinfo* is set to `False`. If `pytz` is not available, setting *use_tzinfo* to `False` is required, as it is not possible to convert between timezone names and `datetime.tzinfo` objects without it.

`class` `internationalflavor.timezone.models.`**`TimezoneField`**(*timezones=None*, *exclude=None*, *use_tzinfo=True*, **args*, ***kwargs*)

A model field that allows users to choose their timezone. By default, all timezones in the set of common timezones of pytz are available. Use the `timezones` argument to specify your own timezones, and use `exclude` to exclude specific zones.

If `use_tzinfo` is `True`, an instance of `datetime.tzinfo` is returned. This requires `pytz` to be installed. If `use_tzinfo` is `False`, a string is returned instead.

class internationalflavor.timezone.forms.**TimezoneFormField**(*timezones=None*, *exclude=None*, **args*, ***kwargs*)

class internationalflavor.timezone.models.**MetazoneField**(*metazones=None*, *exclude=None*, *use_tzinfo=True*, *display_format='name'*, **args*, ***kwargs*)

A model field that allows users to choose their metazone. By default, all metazones in the CLDR set are available. Use the `metazones` argument to specify your own metazones, and use `exclude` to exclude specific zones.

If `use_tzinfo` is `True`, an instance of `datetime.tzinfo` is returned. This requires `pytz` to be installed. Note, however, that only one exemplar timezone tzinfo is returned for the metazone. The exemplar timezone may change over time as cities change their timezones.

If `use_tzinfo` is `False`, a string is returned instead.

class internationalflavor.timezone.forms.**MetazoneFormField**(*metazones=None*, *exclude=None*, *display_format='name'*, **args*, ***kwargs*)

See also:

Django: Time zones Django documentation on time zones

1.6 VAT numbers

VAT numbers are used in many countries for taxing purposes. In the European Union, organizations are required to use these VAT numbers when conducting intra-Community trade and e.g. for reverse charging. Although there's no US equivalent of a VAT number, these are used in most other countries around the world.

class internationalflavor.vat_number.validators.**VATNumberValidator**(*countries=None*, *exclude=None*, *eu_only=False*, *vies_check=False*)

Validator for checking whether a given VAT number is valid. A VAT number starts with two characters representing the country code, followed by at least 2 characters representing the local VAT number.

Parameters

- **countries** – If set, the list of accepted origin countries will be limited to the provided list. Otherwise, all available VAT number countries are used.
- **exclude** – You can use this parameter to exclude items from the list of countries.
- **eu_only** (*bool*) – By default, all countries are allowed. However, if you are an EU company, you are likely to only want to accept EU VAT numbers.
- **vies_check** (*bool*) – By default, this validator will only validate the syntax of the VAT number. If you need to validate using the EU VAT Information Exchange System (VIES) checker (see http://ec.europa.eu/taxation_customs/vies/), you can set this boolean. Any VAT number in the EU VAT Area will then receive additional validation from the VIES checker, other VAT numbers will be unaffected.

The VIES check may use two different methods to obtain the result. If the `suds` module is installed, the VIES check uses this module to reach the VIES WSDL services (you could use the `suds-jurko` fork for Py3k compatibility). If this module is not available, a bare-bones native method is used instead. Both methods should give similar results, although using `suds` should be more reliable.

Note: If the VIES service can not be reached, this part of the validation will succeed.

Note: If regulations require you to validate against the VIES service, you probably also want to set `eu_only`. You probably can't accept any other VAT number in that case.

Note: All valid VAT Numbers are ISO 3166-1 country-2 codes followed by the number, except for Greece, where EL is used. You can specify GR and EL as both valid country codes, but only EL-prefixed values are accepted.

Warning: The validation of non-EU VAT numbers may be incomplete or wrong in some cases. Please issue a pull request if you feel there's an error.

class internationalflavor.vat_number.models.**VATNumberField**(*countries=None*, *exclude=None*, *eu_only=False*, *vies_check=False*, **args*, ***kwargs*)

A model field that applies the *validators.VATNumberValidator* and is represented by a *forms.VATNumberFormField*. The arguments are equal to those of the validator.

Example:

```
from django.db import models
from internationalflavor.vat_number import VATNumberField

class MyModel(models.Model):
    vat_number = VATNumberField(countries=['NL', 'BE'])
```

This field is an extension of a CharField.

class internationalflavor.vat_number.forms.**VATNumberFormField**(*countries=None*, *exclude=None*, *eu_only=False*, *vies_check=False*, **args*, ***kwargs*)

A form field that applies the *validators.VATNumberValidator*. The arguments are equal to those of the validator.

1.7 Changelog

This file contains the changelog for the project.

1.7.1 0.4.3 (2022-04-04)

- Updates CLDR data to v40
- Support for Django 4.x

1.7.2 0.4.2 (2021-11-06)

- Updates CLDR data to v39
- Support for Django 3.x
- Fixes for Brexit meaning change in VAT numbers

1.7.3 0.4.1 (2020-09-14)

Small release containing updates to translation files. Mostly re-releasing to ensure compatibility with the new Pip version resolver. Note that the CLDR update to v37 failed because the npm repositories have not been updated with correct tagging.

1.7.4 0.4.0 (2020-05-04)

- Updates Django support, removing 1.x support
- Updates Python support, removing 2.x support
- Updated CLDR data to v36
- Updated IBAN data, renaming Nordea extensions to experimental countries (based off SWIFT list)
- Updated VAT data, fixing the validation for Dutch VAT numbers
- Added a language field

Due to a packaging error, v0.4.0 as re-uploaded to PyPI without changes to the source itself.

1.7.5 0.3.1 (2017-01-28)

- Fix Django 1.11 bug in the SortedSelect widget. This affects the sorting of optgroups for all versions, they are now always sorted above other choices.
- Updated CLDR data from v29 to v30.0.3
- `timezone`: Add Metazone fields, but it is not sorted yet properly :).
- `vat_number`: Additional cleanup for CH VAT numbers and validation for RU VAT numbers
- `names`: Add two utilties for joining and splitting (parts of) names. It is best-effort, and not intended to be more than that.

1.7.6 0.3.0 (2016-08-20)

- Fixes a Django 1.10 bug (I fixed it before, but it never made it into a release. . .)
- Updated CLDR and IBAN data. Note: UA got an official IBAN, so this means the Nordea alternative got dropped (SC also got a IBAN, but it was never in the Nordea set).

1.7.7 0.2.1 (2015-02-09)

- Fixes a Python 3 bug discovered when releasing 0.2.0

1.7.8 0.2.0 (2015-02-09)

- `vat_number` and `iban`: Some consistency issues resolved; changed argument order and `include_countries` is now simply `countries`.
- `vat_number`: Do not imply `eu_only` when using `vies_check`.
- `vat_number`: Fallback to a native check if suds is not available.

1.7.9 0.1.2 (2014-12-18)

- Important packaging fixes

1.7.10 0.1.1 (2014-12-08)

- `iban`: Added support for IBANs from Kosovo and Timor-Leste, and Nordea extensions from Republic of Congo, Egypt and Gabon.

1.7.11 0.1 (2014-12-01)

- Initial release
- Added modules `countries`, `iban`, `timezone` and `vat_number`

(Expected soon: telephone numbers)

CHAPTER 2

Basic principles

All validators enforce one specific format and generally do not allow any additional white-spacing, dashes or other readability marks. These should not be present in your database, as readability is not a property of your data. However, the provided model and form fields will strip these characters out and allow for a more seamless experience for your users.

Most validators rely on data present in this module, but such data is likely to change over time. Trying to keep this module up-to-date is one of the primary aims of this project, but from time to time an update may be missed. Please send your pull requests for such oversights, preferably including a link to an official resource confirming the change.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index