wagtailtrans Documentation

Release 0.1.0

LUKKIEN

Oct 31, 2017

Contents

Wagtailtrans is a package that can be used to facilitate multi language sites in Wagtail. We developed wagtailtrans with two implementations in mind.

- A **synchronised** tree
- A **freeform** tree

Synchronised tree is a method used to keep the tree structure in all languages the same. Any changes in the tree of the default language will also automatically be done in the language tree(s).

See also:

The documentation about *Synchronized translation trees*

Freeform trees allows the customization of the language trees. Pages can still be copied with content into the preferred language but this is not automatic nor mandatory. moderators can decide how the page structure works for every language.

See also:

The documentation about *Freeform translation trees*

CHAPTER 1

Table of contents

Getting started

To start using wagtailtrans in your project, take the following steps:

Installation

1. Install Wagtailtrans via `pip`

```
$ pip install wagtailtrans
```

2. Add `wagtailtrans` and `wagtail.contrib.modeladmin` to your `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    # ...
    'wagtail.contrib.modeladmin',
    'wagtailtrans',
    # ...
]
```

Note: As of Wagtailtrans 1.0.3 the custom Language management views are replaced with with `wagtail.contrib.modeladmin` This needs to be added to `INSTALLED_APPS` as well.

3. Add `wagtailtrans.middleware.TranslationMiddleware` to your `MIDDLEWARE_CLASSES`:

```
MIDDLEWARE_CLASSES = [
    # ...
    'django.contrib.sessions.middleware.SessionMiddleware',
    'wagtail.wagtailcore.middleware.SiteMiddleware',
    'wagtailtrans.middleware.TranslationMiddleware',
    'django.middleware.common.CommonMiddleware',
```

```
    # ...
]
```

Note: Keep in mind `wagtailtrans.middleware.TranslationMiddleware` is a replacement for `django.middleware.locale.LocaleMiddleware`.

Note: It relies on `wagtail.wagtailcore.middleware.SiteMiddleware`, which should come before it. See http://docs.wagtail.io/en/latest/getting_started/integrating_into_django.html#settings for more information.

Configuration

Before we start incorporating wagtailtrans in your project, you'll need to configure wagtailtrans for the behavior that best suits the need of your project. The required settings to consider here are:

- `WAGTAILTRANS_SYNC_TREE`
- `WAGTAILTRANS_LANGUAGES_PER_SITE`

Both settings are mandatory but provided with a default value, so if you want *synchronized* trees and no languages per site, you're good to go from here.

See also:

Complete reference about available settings: *Settings*

Incorporating

To start using wagtailtrans we first need to create a translation home page. This page will route the requests to the homepage in the right language. We can create a translation site root page by creating the `wagtailtrans.models.TranslatableSiteRootPage` as the first page under the root page.

In this example we will also make a `HomePage` which will be translatable. This is done by implementing the `wagtailtrans.models.TranslatablePage` next to Wagtail's `Page`

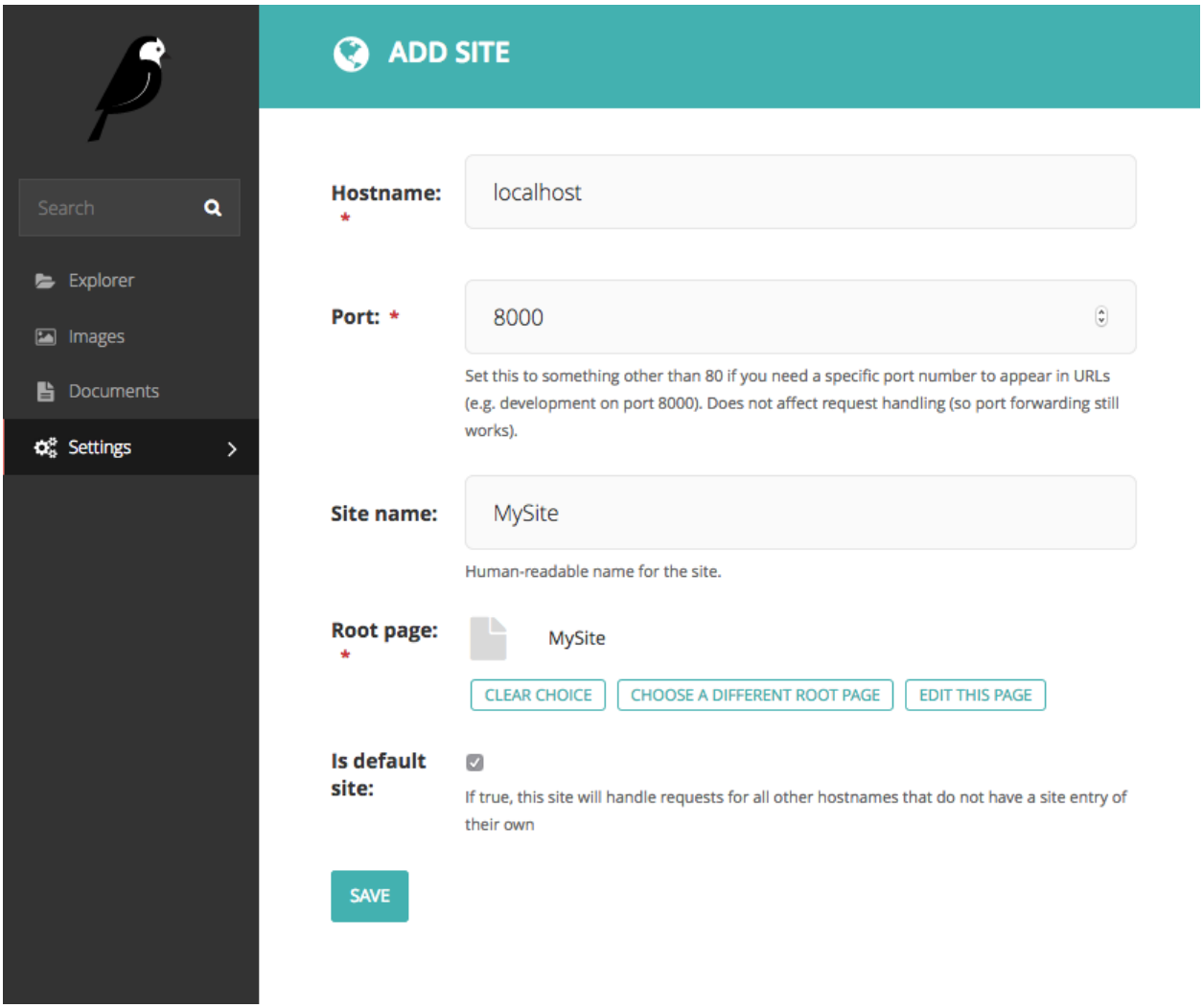
```
from wagtail.wagtailcore.models import Page
from wagtailtrans.models import TranslatablePage


class HomePage(TranslatablePage, Page):
    body = RichTextField(blank=True, default="")
    image = models.ForeignKey('wagtailimages.Image', null=True, blank=True, on_
↪delete=models.SET_NULL, related_name='+')

    content_panels = Page.content_panels + [
        FieldPanel('body'),
        ImageChooserPanel('image')
    ]

    subpage_types = [
        # Your subpage types.
    ]
```

This will create our first translatable page. To start using it we first need to migrate our database

```
$ python manage.py makemigrations
$ python manage.py migrate
```

Now run the server and under the page `Root` create a `TranslatableSiteRootPage` (MySite).

Next we need to create a site and point it's `root_page` to our `TranslatableSiteRootPage` (MySite).



We now have the basics for a Translatable Site.

Migrate your existing Wagtail site

Migrating of already existing Wagtail content to `Wagtailtrans` can be quite difficult. Since there is no way to do this automatically, we've put some examples below to point you in the right direction.

> **Danger:** Examples below contain custom database migrations, make sure you've created a back-up of your database before you start this migration process.

Non Wagtailtrans site

1. Install wagtailtrans

```
$ pip install wagtailtrans
```

2. Add wagtailtrans to `INSTALLED_APPS`

```
INSTALLED_APPS = [
    # ...
    'wagtailtrans',
    # ...
]
```

3. Add `wagtailtrans.models.TranslatablePage` to the existing `wagtail.wagtailcore.models.Page` models.

```
from wagtail.wagtailcore.models import Page
from wagtailtrans.models import TranslatablePage


class HomePage(TranslatablePage, Page):
    # ...
```

4. Create a database migration file, when `makemigrations` is asking for a one-off default value for `translatablepage_ptr` you can fill in a fake value, since we're going to change this later.

```
$ python manage.py makemigrations <appname>
```

5. Update migrations file to add the newly `translatablepage_ptr_id` field in the required table.

Note: We've made some assumptions when creating this migration file, for example a default language `en`. Please make sure you've checked all the database queries and adjusted them according to your own setup before executing.

```
class Migration(migrations.Migration):

    dependencies = [
        ('wagtailtrans', '0006_auto_20161212_2020'),
        ('pages', '0002_auto_20160930_1042'),
    ]

    operations = [
        migrations.RunSQL(
            """
            BEGIN;

            -- Remove constraints so we can edit our table
            ALTER TABLE pages_homepage DROP CONSTRAINT pages_homepage_pkey CASCADE;

            -- Add ``translatablepage_ptr_id`` field and copy the ``page_ptr_id``␣
↪content
            ALTER TABLE pages_homepage ADD COLUMN translatablepage_ptr_id INTEGER␣
↪UNIQUE;
            UPDATE pages_homepage SET translatablepage_ptr_id=page_ptr_id;

            -- Insert the required values in ``wagtailtrans`` table
```

```
            INSERT INTO wagtailtrans_language (code, is_default, position, live)␣
↪SELECT 'en', 't', 0, 't' WHERE NOT EXISTS (SELECT code FROM wagtailtrans_language␣
↪WHERE code='en');
            INSERT INTO wagtailtrans_translatablepage (translatable_page_ptr_id,␣
↪canonical_page_id, language_id) SELECT translatablepage_ptr_id, NULL, 1 FROM pages_
↪homepage;

            -- Add required indexes and constraints
            ALTER TABLE pages_homepage ADD CONSTRAINT pages_homepage_translatablepage_
↪ptr_id_e5b77cf7_fk_wagtailtrans_translatable_page_id FOREIGN KEY (translatablepage_
↪ptr_id) REFERENCES wagtailtrans_translatablepage (translatable_page_ptr_id)␣
↪DEFERRABLE INITIALLY DEFERRED;
            ALTER TABLE pages_homepage ALTER COLUMN translatablepage_ptr_id SET NOT␣
↪NULL;
            ALTER TABLE pages_homepage ADD PRIMARY KEY (translatablepage_ptr_id);

            COMMIT;
            """,
            state_operations=[
                migrations.AddField(
                    model_name='homepage',
                    name='translatablepage_ptr',
                    field=models.OneToOneField(auto_created=True, on_delete=django.db.
↪models.deletion.CASCADE, parent_link=True, primary_key=True, serialize=False, to=
↪'wagtailtrans.TranslatablePage'),
                    preserve_default=False,
                ),
                migrations.AlterField(
                    model_name='homepage',
                    name='page_ptr',
                    field=models.OneToOneField(auto_created=True, on_delete=django.db.
↪models.deletion.CASCADE, parent_link=True, to='wagtailcore.Page'),
                ),
            ]
        ),
    ]
```

Pre 0.1 Wagtailtrans site

Before the 0.1 final release we've made a backwards incompatible change by defining a custom parent_link, this is done to ease the process of migrate an existing Wagtail site to Wagtailtrans.

Migrating can be done by following these steps:

1. Update code where necessary, models inheriting from `wagtailtrans.models.TranslatablePage` should also inherit from `wagtail.wagtailcore.models.Page`

```
from wagtail.wagtailcore.models import Page
from wagtailtrans.models import TranslatablePage


class HomePage(TranslatablePage, Page):
    # ....
```

2. Create a database migration file, when `makemigrations` is asking for a one-off default value for `page_ptr` you can fill in a fake value, since we're going to change this later.

```
$ python manage.py makemigrations <appname>
```

3. Alter the migration file to add the `page_ptr_id` field to the database, update it with the right values, create the required indexes and constraints and update the ORM state with a seperate state operation.

Note: We've made some assumptions when creating this migration file. Please make sure you've checked all the database queries and adjusted them according your own setup before executing.

```
class Migration(migrations.Migration):

    dependencies = [
        ('wagtailcore', '0029_unicode_slugfield_dj19'),
        ('pages', '0002_auto_20160930_1042'),
        ('wagtailtrans', '0006_auto_20161212_2020'),
    ]

    operations = [
        migrations.RunSQL(
            """
            BEGIN;

            -- Add the ``page_ptr_id`` field in the DB.
            ALTER TABLE pages_homepage ADD COLUMN page_ptr_id INTEGER UNIQUE;
            UPDATE pages_homepage SET page_ptr_id=translatablepage_ptr_id;
            ALTER TABLE pages_homepage ALTER COLUMN page_ptr_id DROP DEFAULT;
            ALTER TABLE pages_homepage ALTER COLUMN page_ptr_id SET NOT NULL;
            ALTER TABLE pages_homepage ADD CONSTRAINT pages_homepage_page_ptr_id_
↪5b805d74_fk_wagtailcore_page_id FOREIGN KEY (page_ptr_id) REFERENCES wagtailcore_
↪page (id) DEFERRABLE INITIALLY DEFERRED;

            COMMIT;
            """,
            state_operations=[
                migrations.AddField(
                    model_name='homepage',
                    name='page_ptr',
                    field=models.OneToOneField(auto_created=True, on_delete=django.db.
↪models.deletion.CASCADE, parent_link=True, to='wagtailcore.Page'),
                    preserve_default=False,
                ),
            ]
        ),
    ]
```

Settings

There are a few settings which can be used to configure wagtailtrans to suit your needs, these settings need to be configured in your django settings module. All wagtailtrans settings are prefixed with `WAGTAILTRANS_` to avoid conflicts with other packages used.

WAGTAILTRANS_SYNC_TREE

Default `True`

If set to `False` wagtailtrans will work with `Freeform` trees.

See also:

The documentation about *Synchronized translation trees*

See also:

The documentation about *Freeform translation trees*

WAGTAILTRANS_LANGUAGES_PER_SITE

Default `False`

If set to `True` wagtailtrans will allow you to define a default language and additional languages per site. This is mostly used in a `multi site` setup and allowes you to define the languages per site, this way they can differ for all available sites.

Reference

Synchronized translation trees

Before you can start using syncronized trees, please be sure you followed all steps in: *Getting started*.

If you specified the *WAGTAILTRANS_SYNC_TREE* In your settings as *True* you will be using the syncronized trees. This means that every change in your 'canonical' tree will also be done in the translated trees. To start using this we first need to create a default language (canonical). In your wagtail admin page in settings, select languages.

Add any other language you will be using in your site but don't check *is default*. for demonstration purposes we add German and Spanish.

Now we go to out site root page (MySite) and create a new page. called *Homepage* after saving the Homepage copies of this page will be saved for each language. This will happen for every *TranslatablePage* instance created when *WAGTAILTRANS_SYNC_TREE* = *True*. This way language trees will always be synchronized.

With the creation of a language a new translator user group is created. This group will gain edit and publish permissions on each page with corresponding language.

Now any change made in the canonical tree will also be made in the translation trees.

Freeform translation trees

Before you can start using freeform trees, please be sure you followed all the steps in: *Getting started*.

If you specified the `WAGTAILTRANS_SYNC_TREE` in your settings as `False` you will be using the freeform trees. the freeform trees are used if you do not want to keep your language trees the same for all languages. This enables the *translate into* button in the Wagtail admin. This will show all the languages that the page can be translated into.

- Once a language is selected you will be asked if you want to copy the content and where to move the page to.
- With this setting your page tree structure is free for all languages.
- The translator roles and groups will still be created as in the synchronized tree section.

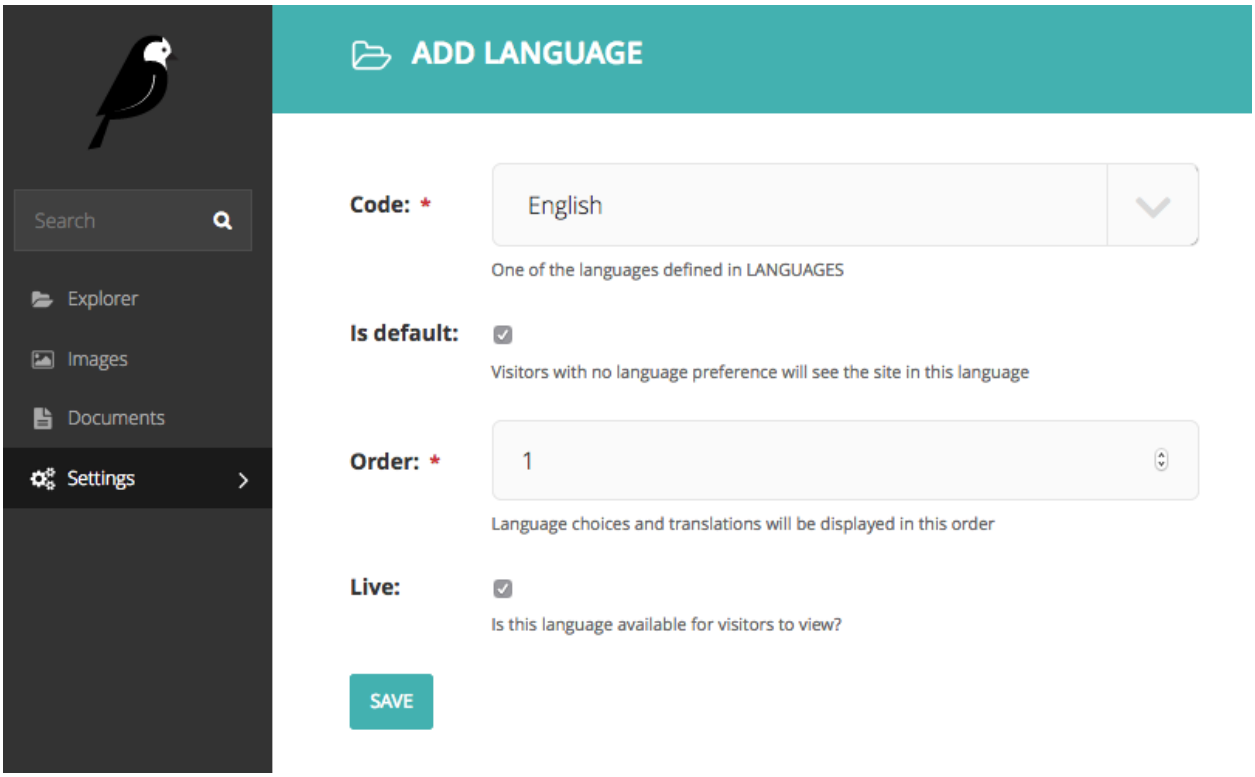
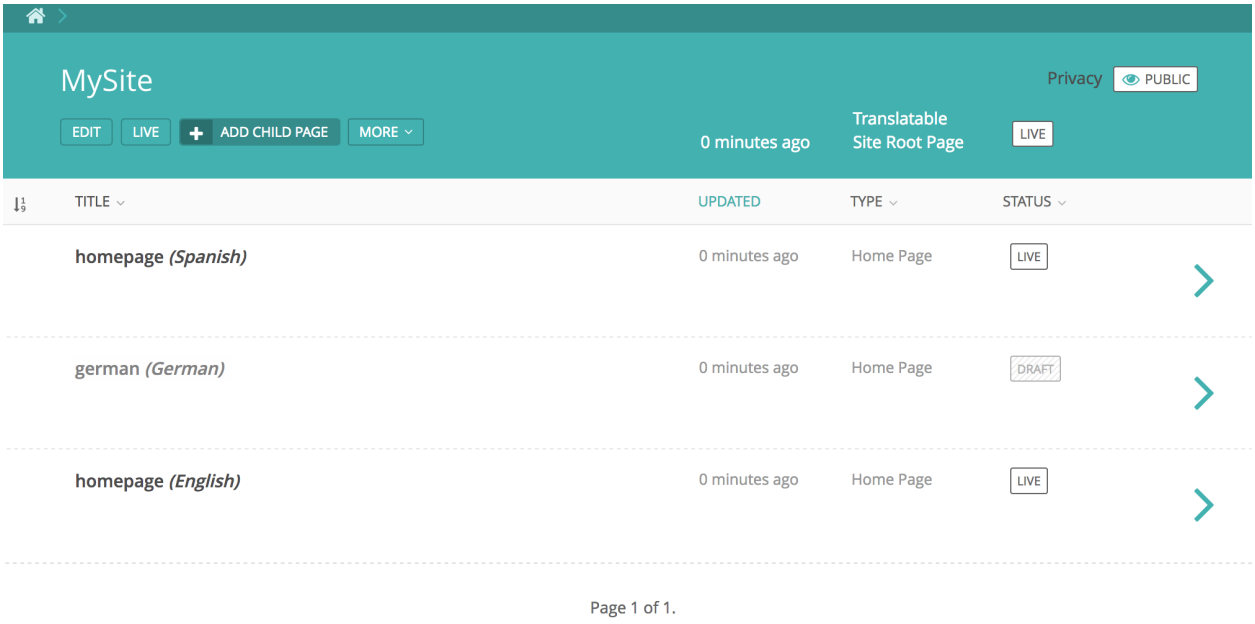
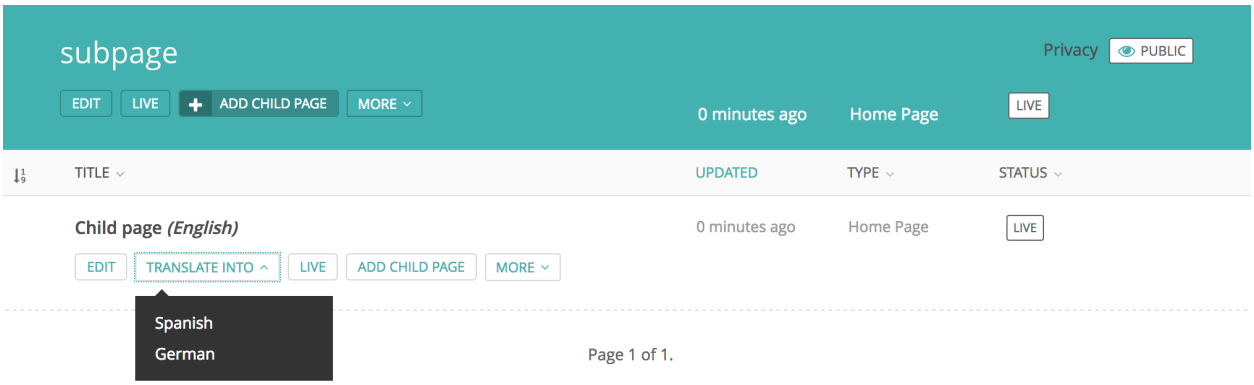


Fig. 1.1: We will be using English as our default language. Please do Note thant only *ONE* default language can be used





Templatetags

Wagtailtrans has a couple of template tags available to make linking in between pages more easy. The template tags can be loaded from `wagtailtrans_tags`:

```
{% load wagtailtrans_tags %}
```

Both template tags are configurable via the same keyword arguments.

homepage_fallback

Default True

By default the template tag will fallback to a homepage if the linked page isn't publlished in the other language(s). This setting will allow you to disable that behavior and leave the page out of the returned result.

include_self

Default True

If set to `False` the requested page won't be included in the result.

get_translations (assignment)

The assignment tag will return a dictionary with language objects as keys and pages as values. For example this can be used to render `<link rel="alternate">` tags.

```
{% get_translations page homepage_fallback=False include_self=False as translations %}

{% for language, page in translations.items %}
<link rel="alternate" href="{{ page.full_url }}" hreflang="{{ language.code }}">
{% endfor %}
```

render_language_selector (inclusion)

This template tag will render a language selector, which renders the template located at: `wagtailtrans/templatetags/language_selector.html`

Release notes

Wagtailtrans 0.1 release notes

What is new

Since this is the first final release, all features are new.

Features

- Implement models following Wagtail RFC9 by Tim Heap
- Force language of child pages to language of parent
- Support storing of translated pages
- Support copying content of canonical pages when creating translations
- Add translation information to the `TranslatablePage.settings_panels`
- Add dropdown page menu for adding translations
- Add Language admin-UI in settings-menu
- Add `WAGTAILTRANS_SYNC_TREE` setting to control which way trees behave
- Add `WAGTAILTRANS_TEMPLATE_DIR` to override the admin template dir (pre Wagtail 1.8)
- Add `WAGTAILTRANS_LANGUAGES_PER_SITE` setting to allow different page languages per site
- Add SiteLanguages as SiteSetting in settings-menu (`WAGTAILTRANS_LANGUAGES_PER_SITE`)
- Add `wagtailtrans.models.TranslatablePage.get_admin_display_title` to display the page language in the admin explorer (Wagtail 1.8+)

Backwards incompatible changes

In the progress of creating this release, we found out that some earlier design choices needed to be reworked in order to have a good maintainable package.

`include_self`

Before wagtailtrans `0.1.0b4` the method `TranslatablePage.get_translations` implemented a `kwarg` to include itself in the QuerySet of translated pages. Since this isn't used within the wagtailtrans codebase we removed it because keeping it there made the codebase better maintainable.

wagtailtrans.models.TranslatablePage.translatable_page_ptr

As of wagtailtrans 0.1 we've refactored the `TranslatablePage` model to have a custom parent_link, this way it's easier to migrate any existing Wagtail site for use with wagtailtrans. This is done by inheriting from `TranslatablePage` and `Page` for implementations of pages.

As for migrating, this is sadly not possible without some custom migrations, a way to do this is documented here: *Migrate your existing Wagtail site*

Wagtailtrans 0.1.1 release notes

What is new

This release mainly implements some minor changes.

Features

- Add `TranslationMiddleware` for earlier language activation, with this middleware error pages will be displayed in the requested language.
- Add `language_id` filter to `wagtailtrans.models.TranslatablePage` so search results can be filterd by language
- Update documentation & typo fixes.

Wagtailtrans 0.1.2 release notes

What is new

This release mainly fixes some annoying bugs and brings full support for wagtail 1.8 and 1.9.

Features

- Add new `wagtailtrans` logo.
- Add admin view for translating with `WAGTAILTRANS_SYNC_TREE = False`, prevents issues with not having a `_meta` on the `TranslationForm`.
- Update `TranslationMiddleware` to also use the `Accept-Language` HTTP header
- Fix: Update middleware to prevent errors on empty database

- Fix: backwards compatibility with Django 1.8 for `wagtailtrans.templatetags`

Backwards incompatible changes

Wagtail 1.6

This release drops support for Wagtail 1.6, most things should probably still work, but it's not supported for bugfixes anymore.

Wagtailtrans 0.1.3 release notes

What is new

This is a minor update generally adding some convenience updates

Features

- Add `include_self=False` kwarg to `TranslatablePage.get_translations()` to have the page return itself as well
- Add language selector template tags
- Update `Language` management to make use of `wagtail.contrib.modeladmin`

Wagtailtrans 0.1.4 release notes

What is new

This release mainly updates the language selector and brings full support for Wagtail 1.11.

Features

- Add support for `Wagtail 1.11`
- Update language selector templatetag to work with pages without language as well
- Update language selector to order language selector based on language positions

Wagtailtrans 0.1.5 release notes

What is new

This release mainly fixes a blocking issue and adds some convenience updates for development.

Features

- Added *appveyor* to the CI to ensure MSSQL compatibility
- Added a settings example file to the sandbox environment to ease the contribution setup
- Added documentation on `Contributing to Wagtailtrans`
- Fix: language admin view throwing an exception with `WAGTAILTRANS_LANGUAGES_PER_SITE` set to `True`

Wagtailtrans 0.2 release notes

What is new

This release is a new major since it will only support Wagtail 1.12 and up. By dropping support for < 1.11 we are keeping inline with Wagtails own release schedule.

Features

- Dropped support for Wagtail versions earlier than 1.12
- Custom `get_admin_display_title` method now uses Wagtails `draft_title`
- Fix: Get default language when `languages_per_site` is enabled now returns the correct default language for the site.

Backwards incompatible changes

This release drops support for Wagtail versions below 1.12.

- References to `WAGTAILTRANS_TEMPLATE_DIR` can be cleaned up since it was a requirement prior to Wagtail 1.8

Contributing to Wagtailtrans

Thank you for considering contributing to Wagtailtrans.

We appreciate all kinds of support, whether it's on bug reports, code, documentation, tests or feature requests.

Issues

Our Github issue tracker is the preferred channel for bug reports, feature requests and submitting pull requests. When creating new issues please provide as much relevant context as possible. Even your detailed comments on existing issues are a big help.

Pull request

You can contribute your code by creating a new pull request. If there are any open issues you think you can help with, you can discuss it on that specific issue and start working on it. If you have any idea for a new feature, please raise it as an issue. Once a core contributor has responded and is happy with your idea and the solution, you can start coding.

You should create your own fork of Wagtailtrans. We recommend you to create separate branches to keep all related changes within the same branch. Once you are done with your changes, you can submit a pull request for review.

Please keep in mind

- Any new feature must be documented.
- New features and bugfixes should have corresponding unit tests.
- If you're not already on the list, add your name to `CONTRIBUTORS.md`.

Development

Wagtailtrans made it very easy to setup a runnable Django project to help with the development. It ships with a Sandbox application that can be availed for this purpose. You need to have some additional packages installed and a PostgreSQL Database on your local machine.

- **Get the codebase**

  Get a copy of the Wagtailtrans codebase. Create your own fork and make changes there. For a brief, take a look at this guideline.

- **Setup database**

  Copy `tests/_sandbox/settings/local_settings.sample` to `tests/_sandbox/settings/local_settings.py` if you would like to use different database settings then we use in our default Sandbox application. This way you can setup your own database settings for Django.

- **Setup local development server**

  1. Activate your virtual environment.
  2. Run following command :

     ```
     $ make sandbox
     ```

     This will install required packages and run the initial data migrations.

- **Run locally**

```
$ ./manage.py runserver
```

Testing

We use pytest for unit testing. To execute the full testsuite, run:

```
$ make qt
```

or for a specific file:

```
$ py.test path/to/file
```

If you want to measure test coverage you can run:

```
$ make coverage
```

Wagtailtrans supports multiple environments which can be tested with `tox`. It takes a bit longer to complete, but you can run it by a simple command: (Please make sure you have a setup with multiple versions of python, in order to run this command.)

```
$ tox
```