django-sitetree Documentation

Release 1.7.0

Igor 'idle sign' Starikov

Apr 14, 2017

Contents

django-sitetree is a reusable application for Django, introducing site tree, menu and breadcrumbs navigation elements.

Site structure in django-sitetree is described through Django admin interface in a so called site trees. Every item of such a tree describes a page or a set of pages through the relation of URI or URL to human-friendly title. Eg. using site tree editor in Django admin:

```
URI             Title
  /             - Site Root
  |_users/      - Site Users
    |_users/13/ - Definite User
```

Alas the example above makes a little sense if you have more than just a few users, that's why django-sitetree supports Django template tags in item titles and Django named URLs in item URIs.

If we define a named URL for user personal page in urls.py, for example, 'users-personal', we could change a scheme in the following way:

```
URI                           Title
  /                           - Site Root
  |_users/                    - Site Users
    |_users-personal user.id  - User Called {{ user.first_name }}
```

After setting up site structure as a sitetree you should be able to use convenient and highly customizable site navigation means (menus, breadcrumbs and full site trees).

User access to certain sitetree items can be restricted to authenticated users or more accurately with the help of Django permissions system (Auth contrib package).

CHAPTER 1

Requirements

1. Python 2.6+, 3.3+
2. Django 1.7+
3. Auth Django contrib package
4. Admin site Django contrib package (optional)

CHAPTER 2

Table of Contents

Getting started

1. Add the **sitetree** application to INSTALLED_APPS in your settings file (usually 'settings.py').
2. Check that *django.core.context_processors.request* is added to TEMPLATE_CONTEXT_PROCESSORS in your settings file. For Django 1.8+: *django.template.context_processors.request* should be defined in `TEMPLATES/OPTIONS/context_processors`.
3. Check that *django.contrib.auth.context_processors.auth* is enabled in TEMPLATE_CONTEXT_PROCESSORS too.
4. Run `./manage.py syncdb` to install sitetree tables into database (`./manage.py migrate` for Django 1.7+).

   Warning: Those, who are using South <1.0 for migrations with Django <1.7, add this into settings file:

   ```
   SOUTH_MIGRATION_MODULES = {
       'sitetree': 'sitetree.south_migrations',
   }
   ```

5. Go to Django Admin site and add some trees and tree items (see *Making tree* section).
6. Add *{% load sitetree %}* tag to the top of a template.

Now you can use the following template tags:

- *sitetree_menu* - to render menu based on sitetree;
- *sitetree_breadcrumbs* - to render breadcrumbs path based on sitetree;
- *sitetree_tree* - to render site tree;
- *sitetree_page_title* - to render current page title resolved against definite sitetree.

- *sitetree_page_description* - to render current page description resolved against definite sitetree.

Making tree

Taken from StackOverflow.

In this tutorial we create sitetree that could handle URI like */categoryname/entryname*.

To create a tree:

1. Go to site administration panel;
2. Click +Add near 'Site Trees';
3. Enter alias for your sitetree, e.g. 'maintree'. You'll address your tree by this alias in template tags;
4. Push 'Add Site Tree Item';
5. Create first item:

```
Parent - As it is root item that would have no parent.
Title - Let it be 'My site'.
URL - This URL is static, so put here '/'.
```

6. Create second item (that one would handle 'categoryname' from your 'categoryname/entryname'):

```
Parent - Choose 'My site' item from step 5.
Title - Put here 'Category #{{ category.id }}'.
URL - Put named URL 'category-detailed category.name'.

In 'Additional settings': check 'URL as Pattern' checkbox.
```

7. Create third item (that one would handle 'entryname' from your 'categoryname/entryname'):

```
Parent - Choose 'Category #{{ category.id }}' item from step 6.
Title - Put here 'Entry #{{ entry.id }}'.
URL - Put named URL 'entry-detailed category.name entry.name'.

In 'Additional settings': check 'URL as Pattern' checkbox.
```

8. Put '{% load sitetree %}' into yor template to have access to sitetree tags.
9. Put '{% sitetree_menu from "maintree" include "trunk" %}' into your template to render menu from tree trunk.
10. Put '{% sitetree_breadcrumbs from "maintree" %}' into your template to render breadcrumbs.

Steps 6 and 7 clarifications:

- In titles we use Django template variables, which would be resolved just like they do in your templates.

  E.g.: You made your view for 'categoryname' (let's call it 'detailed_category') to pass category object into template as 'category' variable. Suppose that category object has 'id' property. In your template you use '{{ category.id }}' to render id. And we do just the same for site tree item in step 6.

- In URLs we use Django's named URL patterns (documentation). That is almost idential to usage of Django 'url' tag in templates.

  Your urls configuration for steps 6, 7 supposed to include:

```
url(r'^(?P<category_name>\S+)/(?P<entry_name>\S+)/$', 'detailed_entry', name=
↪'entry-detailed'),
url(r'^(?P<category_name>\S+)/$', 'detailed_category', name='category-detailed'),
```

Consider 'name' argument values of 'url' function.

So, putting 'entry-detailed category.name entry.name' in step 7 into URL field we tell sitetree to associate that sitetree item with URL named 'entry-detailed', passing to it category_name and entry_name parameters.

SiteTree template tags

To use template tags available in SiteTree you should add **{% load sitetree %}** tag to the top of chosen template.

Tree tag argument (part in double quotes, following '**from**' word) of SiteTree tags should containt tree alias.

Hints:

- Tree tag argument could be a template variable (do not use quotes for those).
- Optional **template** argument could be supplied to all SitetTree tags except *sitetree_page_title* to render using different templates. It should contain path to template file.

  Examples:

```
{% sitetree_menu from "mytree" include "trunk,topmenu" template "mytrees/mymenu.
↪html" %}
{% sitetree_breadcrumbs from "mytree" template "mytrees/mybreadcrumbs.html" %}
```

sitetree_menu

This tag renders menu based on sitetree.

Usage example:

```
{% sitetree_menu from "mytree" include "trunk,topmenu" %}
```

This command renders as a menu sitetree items from tree named 'mytree', including items **under** 'trunk' and 'topmenu' aliased items. That means that 'trunk' and 'topmenu' themselves won't appear in a menu, but rather all their ancestors. If you need item filtering behaviour please use *tree hooks*.

Aliases are given to items through Django's admin site.

Note that there are some reserved aliases. To illustrate how do they work, take a look at the sample tree:

```
Home
|-- Users
|    |-- Moderators
|    |-- Ordinary
|
|-- Articles
|    |-- About cats
|    |      |-- Good
|    |      |-- Bad
|    |      |-- Ugly
|    |
|    |-- About dogs
|    |-- About mice
```

```
|
|-- Contacts
|    |-- Russia
|    |      |-- Web
|    |      |      |-- Public
|    |      |      |-- Private
|    |      |
|    |      |-- Postal
|    |
|    |-- Australia
|    |-- China
Exit
```

Note: As it mentioned above, basic built-in templates won't limit the depth of rendered tree, if you need to render the limited number of levels, you ought to *override the built-in templates*. For brevity rendering examples below will show only top level rendered for each alias.

- **trunk** - get hierarchy under trunk, i.e. root item(s) - items without parents:

  Renders:

  ```
  Home
  Exit
  ```

- **this-children** - get items under item resolved as current for the current page;

  Considering that we are now at *Articles* renders:

  ```
  About cats
  About dogs
  About mice
  ```

- **this-siblings** - get items under parent of item resolved as current for the current page (current item included);

  Considering that we are now at *Bad* renders:

  ```
  Good
  Bad
  Ugly
  ```

- **this-parent-siblings** - items under parent item for the item resolved as current for the current page.

  Considering that we are now at *Public* renders:

  ```
  Web
  Postal
  ```

- **this-ancestor-children** - items under grandparent item (closest to root) for the item resolved as current for the current page.

  Considering that we are now at *Public* renders all items under *Home* (which is closest to the root).

Thus in the template tag example above *trunk* is reserved alias, and *topmenu* alias is given to an item through admin site.

Sitetree items could be addressed not only by aliases but also by IDs:

```
{% sitetree_menu from "mytree" include "10" %}
```

sitetree_breadcrumbs

This tag renders breadcrumbs path (from tree root to current page) based on sitetree.

Usage example:

```
{% sitetree_breadcrumbs from "mytree" %}
```

This command renders breadcrumbs from tree named 'mytree'.

sitetree_tree

This tag renders entire site tree.

Usage example:

```
{% sitetree_tree from "mytree" %}
```

This command renders sitetree from tree named 'mytree'.

sitetree_page_title

This tag renders current page title resolved against definite sitetree. Title is taken from a sitetree item title resolved as current for the current page.

Usage example:

```
{% sitetree_page_title from "mytree" %}
```

This command renders current page title from tree named 'mytree'.

sitetree_page_description

This tag renders current page description resolved against definite sitetree. Description is taken from a sitetree item description resolved as current for the current page.

That can be useful for meta description for an HTML page.

Usage example:

```
{% sitetree_page_description from "mytree" %}
```

This command renders current page description from tree named 'mytree'.

sitetree_page_hint

This tag is similar to *sitetree_page_description*, but it uses data from tree item *hint* field instead of a *description* fields.

Usage example:

```
{% sitetree_page_hint from "mytree" %}
```

SITETREE_RAISE_ITEMS_ERRORS_ON_DEBUG

DEFAULT: True

There are some rare occasions when you want to turn off errors that are thrown by sitetree even during debug.

Setting SITETREE_RAISE_ITEMS_ERRORS_ON_DEBUG = False will turn them off.

Internationalization

With django-sitetree it is possible to render different trees for different active locales still addressing them by the same alias from a template.

`register_i18n_trees(aliases)` function registers aliases of internationalized sitetrees. Internationalized sitetrees are those, which are dubbed by other trees having locale identifying suffixes in their aliases.

Lets suppose `my_tree` is the alias of a generic tree. This tree is the one that we call by its alias in templates, and it is the one which is used if no i18n version of that tree is found.

Given that `my_tree_en`, `my_tree_ru` and other `my_tree_{locale-id}`-like trees are considered internationalization sitetrees. These are used (if available) in accordance with current locale used in project.

Example:

```
# This code usually belongs to urls.py (or `ready` method of a user defined
# sitetree application config if Django 1.7+).

# First import the register function.
from sitetree.sitetreeapp import register_i18n_trees


# Now register i18n trees.
register_i18n_trees(['my_tree', 'my_another_tree'])

# After that you need to create trees for languages supported
# in your project, e.g.: `my_tree_en`, `my_tree_ru`, `my_tree_pt-br`.

# Then when we address ``my_tree`` from a template django-sitetree will render
# an appropriate tree for locale currently active in your project.
# See ``activate`` function from ``django.utils.translation``
# and https://docs.djangoproject.com/en/dev/topics/i18n/
# for more information.
```

Shipping sitetrees with your apps

SiteTree allows you to define sitetrees within your apps.

Defining a sitetree

Let's suppose you have *books* application and want do define a sitetree for it.

- First create *sitetrees.py* in the directory of *books* app.
- Then define a sitetree with the help of *tree* and *item* functions from *sitetree.utils* module and assign it to *sitetrees* module attribute

```
from sitetree.utils import tree, item

# Be sure you defined `sitetrees` in your module.
sitetrees = (
  # Define a tree with `tree` function.
  tree('books', items=[
      # Then define items and their children with `item` function.
      item('Books', 'books-listing', children=[
          item('Book named "{{ book.title }}"', 'books-details', in_menu=False, in_
↪sitetree=False),
          item('Add a book', 'books-add'),
          item('Edit "{{ book.title }}"', 'books-edit', in_menu=False, in_
↪sitetree=False)
      ])
  ]),
  # ... You can define more than one tree for your app.
)
```

Please see *tree* and *item* signatures for possible options.

Note: If you added extra fields to the Tree and TreeItem models, then you can specify their values when instantiating *item* see *Sitetree definition with custom models*

Export sitetree to DB

Now when your app has a defined sitetree you can use *sitetree_resync_apps* management command to instantly move sitetrees from every (or certain) applications into DB:

```
python manage.py sitetree_resync_apps
```

Or solely for *books* application:

```
python manage.py sitetree_resync_apps books
```

Dynamic sitetree structuring

Optionally you can structure app-defined sitetrees into existing or new trees runtime.

Basically one should compose a dynamic tree with *compose_dynamic_tree()* and register it with *register_dynamic_trees()*.

Let's suppose the following code is in *setting.py* (for Django < 1.7; or for Djagno >= 1.7 somewhere where app registry is already created, e.g. *urls.py*) of your project.

```
from sitetree.sitetreeapp import register_dynamic_trees, compose_dynamic_tree
from sitetree.utils import tree, item


register_dynamic_trees(
```

```
    # Gather all the trees from `books`,
    compose_dynamic_tree('books'),

    # or gather all the trees from `books` and attach them to `main` tree root,
    compose_dynamic_tree('books', target_tree_alias='main'),

    # or gather all the trees from `books` and attach them to `for_books` aliased␣
↪item in `main` tree,
    compose_dynamic_tree('books', target_tree_alias='main', parent_tree_item_alias=
↪'for_books'),

    # or even define a tree right at the process of registration.
    compose_dynamic_tree((
        tree('dynamic', items=(
            item('dynamic_1', 'dynamic_1_url', children=(
                item('dynamic_1_sub_1', 'dynamic_1_sub_1_url'),
            )),
            item('dynamic_2', 'dynamic_2_url'),
        )),
    )),

    # Line below tells sitetree to drop and recreate cache, so that all newly␣
↪registered
    # dynamic trees are rendered immediately.
    reset_cache=True
)
```

Management commands

SiteTree comes with two management commands which can facilitate development and deployment processes.

sitetreedump

Sends sitetrees from database as a fixture in JSON format to output.

Output all trees and items into *treedump.json* file example:

```
python manage.py sitetreedump > treedump.json
```

You can export only trees that you need by supplying their aliases separated with spaces:

```
python manage.py sitetreedump my_tree my_another_tree > treedump.json
```

If you need to export only tree items without trees use `--items_only` command switch:

```
python manage.py sitetreedump --items_only my_tree > items_only_dump.json
```

Use `--help` command switch to get quick help on the command:

```
python manage.py sitetreedump --help
```

sitetreeload

This command loads sitetrees from a fixture in JSON format into database.

> **Warning:** *sitetreeload* won't even try to restore permissions for sitetree items, as those should probably be tuned in production rather than exported from dev.
>
> If required you can use Django's *loaddata* management command with *sitetreedump* created dump, or the *dumpscript* from *django-extensions* to restore the permissions.

Command makes use of `--mode` command switch to control import strategy.

a) *append* (default) mode should be used when you need to extend sitetree data that is now in DB with that from a fixture.

Note: In this mode trees and tree items identifiers from a fixture will be changed to fit existing tree structure.

b) *replace* mode should be used when you need to remove all sitetree data existing in DB and replace it with that from a fixture.

Warning: Replacement is irreversible. You should probably dump sitetree data if you think that you might need it someday.

Using *replace* mode:

```
python manage.py sitetreeload --mode=replace treedump.json
```

Import all trees and items from *treedump.json* file example:

```
python manage.py sitetreeload treedump.json
```

Use `--items_into_tree` command switch and alias of target tree to import all tree items from a fixture there. This will not respect any trees information from fixture file - only tree items will be considered. **Keep in mind** also that this switch will automatically change *sitetreeload* commmand into *append* mode:

```
python manage.py sitetreeload --items_into_tree=my_tree items_only_dump.json
```

Use `--help` command switch to get quick help on the command:

```
python manage.py sitetreeload --help
```

Notes on built-in templates

Default templates shipped with SiteTree created to have as little markup as possible in a try to fit most common website need.

Styling built-in templates

Use CSS to style default templates for your needs. Templates are deliberately made simple, and only consist of *ul*, *li* and *a* tags.

Nevertheless pay attention that menu template also uses two CSS classes marking tree items:

- **current_item** — marks item in the tree, corresponding to current page;

- **current_branch** — marks all ancestors of current item, and current item itself.

Overriding built-in templates

To customize visual representation of navigation elements you should override the built-in SiteTree templates as follows:

1. Switch to sitetree folder
2. Switch further to 'templates/sitetree'
3. There among others you'll find the following templates:
   - breadcrumbs.html (basic breadcrumbs)
   - breadcrumbs-title.html (breadcrumbs that can be put inside html *title* tag)
   - menu.html (basic menu)
   - tree.html (basic tree)
4. Copy whichever of them you need into your project templates directory and feel free to customize it.
5. See *Advanced SiteTree tags section* for clarification on two advanced SiteTree template tags.

Templates for Foundation Framework

Information about Foundation Framework is available at http://foundation.zurb.com

The following templates are bundled with SiteTree:

- *sitetree/breadcrumbs_foundation.html*

  This template can be used to construct a breadcrumb navigation from a sitetree.

- *sitetree/menu_foundation.html*

  This template can be used to construct Foundation Nav Bar (classic horizontal top menu) from a sitetree.

  Note: The template renders no more than two levels of a tree with hover dropdowns for root items having children.

- *sitetree/menu_foundation-vertical.html*

  This template can be used to construct a vertical version of Foundation Nav Bar, suitable for sidebar navigation.

  Note: The template renders no more than two levels of a tree with hover dropdowns for root items having children.

- *sitetree/sitetree/menu_foundation_sidenav.html*

  This template can be used to construct a Foundation Side Nav.

  Note: The template renders only one tree level.

You can take a look at Foundation navigation elements examples at http://foundation.zurb.com/docs/navigation.php

Templates for Bootstrap Framework

Information about Bootstrap Framework is available at http://getbootstrap.com

The following templates are bundled with SiteTree:

- *sitetree/breadcrumbs_bootstrap.html*

  This template can be used to construct a breadcrumb navigation from a sitetree.

- *sitetree/breadcrumbs_bootstrap3.html*

  The same as above but for Bootstrap version 3.

- *sitetree/menu_bootstrap.html*

  This template can be used to construct *menu contents* for Bootstrap Navbar.

  Warning: To widen the number of possible use-cases for which this template can be applied, it renders only menu contents, but not Navbar container itself.

  This means that one should wrap *sitetree_menu* call into the appropriately styled divs (i.e. having classes *navbar*, *navbar-inner*, etc.).

  Example:

```
<div class="navbar">
    <a class="brand" href="/">My Site</a>
    <div class="navbar-inner">
        {% sitetree_menu from "main" include "topmenu" template "sitetree/menu_
↪bootstrap.html" %}
     </div>
</div>
```

  Please see Bootstrap Navbar documentation for more information on subject.

  Note: The template renders no more than two levels of a tree with hover dropdowns for root items having children.

- *sitetree/menu_bootstrap3.html*

  The same as above but for Bootstrap version 3.

- *sitetree/menu_bootstrap_dropdown.html*

  One level deep dropdown menu.

- *sitetree/menu_bootstrap3_dropdown.html*

  The same as above but for Bootstrap version 3.

- *sitetree/menu_bootstrap_navlist.html*

  This template can be used to construct a Bootstrap Nav list.

  Note: The template renders only one tree level.

- *sitetree/menu_bootstrap3_navpills.html*

  Constructs nav-pills Bootstrap 3 horizontal navigation.

- *sitetree/menu_bootstrap3_navpills-stacked.html*

  Constructs nav-pills Bootstrap 3 vertical navigation similar to navlist from Bootstrap 2.

You can find Bootstrap navigation elements examples at http://getbootstrap.com/components/

Templates for Semantic UI Framework

Information about Semantic UI Framework is available at http://semantic-ui.com/

The following templates are bundled with SiteTree:

- *sitetree/breadcrumbs_semantic.html*

  This template can be used to construct a breadcrumb navigation from a sitetree.

- *sitetree/menu_semantic.html*

  This template can be used to construct Semantic Menu (classic horizontal top menu) from a sitetree.

  Warning: To widen the number of possible use-cases for which this template can be applied, it renders only menu contents, but not the UI Menu container itself.

  This means that one should wrap *sitetree_menu* call into the appropriately styled divs (i.e. having *ui menu* classes).

  Example:

```
<div class="ui menu">
    <a class="item" href="/">MY SITE</a>
    {% sitetree_menu from "main" include "topmenu" template "sitetree/menu_
↪semantic.html" %}
</div>
```

  Please see Semantic UI Menu documentation for more information on subject.

  Note: The template renders no more than two levels of a tree with hover dropdowns for root items having children.

- *sitetree/menu_semantic-vertical.html*

  This template can be used to construct a vertical version of Semantic UI Menu, suitable for sidebar navigation.

  Note: The template renders no more than two levels of a tree with hover dropdowns for root items having children.

Advanced SiteTree tags

SiteTree introduces two advanced template tags which you have to deal with in case you override the built-in sitetree templates.

sitetree_children

Implements down the tree traversal with rendering.

Usage example:

```
{% sitetree_children of someitem for menu template "sitetree/mychildren.html" %}
```

Used to render child items of specific sitetree item 'someitem' for 'menu' navigation type, using template "sitetree/mychildren.html".

Allowed navigation types: 1) *menu*; 2) *sitetree*.

Basically template argument should contain path to current template itself.

sitetree_url

Resolves site tree item's url or url pattern.

Usage example:

```
{% sitetree_url for someitem params %}
```

This tag is much the same as Django built-in 'url' tag. The difference is that after 'for' it should get site tree item object.

It can cast the resolved URL into a context variable when using *as* clause just like *url* tag.

Tree hooks

What to do if a time comes and you need some fancy stuff done to tree items that django-sitetree does not support? It might be that you need some special tree items ordering in a menu, or you want to render in a huge site tree with all articles titles that are described by one tree item in Django admin, or god knowns what else.

django-sitetree can facilitate on that as it comes with `register_items_hook(callable)` function which registers a hook callable to process tree items right before they are passed to templates.

Note that callable should be able to:

1. **handle `tree_items` and `tree_sender` key params.** `tree_items` will contain a list of extended TreeItem objects ready to pass to template.

   `tree_sender` will contain navigation type identifier (e.g.: *menu*, *sitetree*, *breadcrumbs*, *menu.children*, *sitetree.children*)

2. return a list of extended TreeItems objects to pass to template.

Example:

```
# First import the register function.
from sitetree.sitetreeapp import register_items_hook


# The following function will be used as items processor.
def my_items_processor(tree_items, tree_sender):
    # Suppose we want to process only menu child items.
    if tree_sender == 'menu.children':
        # Lets add 'Hooked: ' to resolved titles of every item.
```

```
        for item in tree_items:
            item.title_resolved = 'Hooked: %s' % item.title_resolved
    # Return items list mutated or not.
    return tree_items

# And we register items processor.
register_items_hook(my_items_processor)
```

Note: You might also be interested in the notes on *Overriding SiteTree Admin representation*.

Overriding SiteTree Admin representation

SiteTree allows you to override tree and tree item representation in Django Admin interface.

That could be used not only for the purpose of enhancement of visual design but also for integration with other applications, using admin inlines. The following functions from *sitetree.admin* could be used to override tree and tree item representation:

- *override_tree_admin()* is used to customize tree representation.
- *override_item_admin()* is used to customize tree item representation.

Example:

```
# Supposing we are in admin.py of your own application.

# Import two helper functions and two admin models to inherit our custom model from.
from sitetree.admin import TreeItemAdmin, TreeAdmin, override_tree_admin, override_
↪item_admin

# This is our custom tree admin model.
class CustomTreeAdmin(TreeAdmin):
    exclude = ('title',)  # Here we exclude `title` field from form.

# And our custom tree item admin model.
class CustomTreeItemAdmin(TreeItemAdmin):
    # That will turn a tree item representation from the default variant
    # with collapsible groupings into a flat one.
    fieldsets= None

# Now we tell the SiteTree to replace generic representations with custom.
override_tree_admin(CustomTreeAdmin)
override_item_admin(CustomTreeItemAdmin)
```

Note: You might also be interested in using *Tree hooks*.

Inlines override example

In the example below we'll use django-seo application from https://github.com/willhardy/django-seo

According to django-seo documentation it allows an addition of custom metadata fields to your models, so we use it to connect metadata to sitetree items.

That's how one might render django-seo inline form on sitetree item create and edit pages:

```
from rollyourown.seo.admin import get_inline
from sitetree.admin import TreeItemAdmin, TreeAdmin, override_tree_admin, override_
↪item_admin
# Let's suppose our application contains seo.py with django-seo metadata class␣
↪defined.
from myapp.seo import CustomMeta


class CustomTreeItemAdmin(TreeItemAdmin):
    inlines = [get_inline(CustomMeta)]

override_item_admin(CustomTreeItemAdmin)
```

SiteTree Forms and Fields

Ocasionally you may want to link some site entities (e.g. Polls, Articles) to certain sitetree items (as to categorize them). You can achieve it with the help of generic forms and fields shipped with SiteTree.

TreeItemForm

You can inherit from that form to have a dropdown with tree items for a certain tree:

```
from sitetree.forms import TreeItemForm


class MyTreeItemForm(TreeItemForm):
    """We inherit from TreeItemForm to allow user link some title to sitetree item.
    This form besides `title` field will have `tree_item` dropdown.

    """

    title = forms.CharField()

# We instruct our form to work with `main` aliased sitetree.
# And we set tree item with ID = 2 as initial.
my_form = MyTreeItemForm(tree='main', tree_item=2)
```

You can also use a well known *initial={'tree_item': 2}* approach to set an initial sitetree item.

After that deal with that form just as usual.

TreeItemChoiceField

TreeItemChoiceField is what *TreeItemForm* uses internally to represent sitetree items dropdown, and what used in Admin contrib on sitetree item create/edit pages.

You can inherit from it (and customized it) or use it as it is in your own forms:

```
from sitetree.fields import TreeItemChoiceField


class MyField(TreeItemChoiceField):
```

```
    # We override template used to build select choices.
    template = 'my_templates/tree_combo.html'
    # And override root item representation.
    root_title = '-** Root item **-'
```

SiteTree Models

SiteTree comes with Tree and Tree item built-in models to store sitetree data.

Models customization

Now let's pretend you are not satisfied with SiteTree built-in models and want to customize them.

1. First thing you should do is to define your own *tree* and *tree item* models inherited from *TreeBase* and *TreeItemBase* classes respectively:

```
# Suppose you have `myapp` application.
# In its `models.py` you define your customized models.
from sitetree.models import TreeItemBase, TreeBase


class MyTree(TreeBase):
    """This is your custom tree model.
    And here you add `my_tree_field` to all fields existing in `TreeBase`.

    """
    my_tree_field = models.CharField('My tree field', max_length=50, null=True,␣
↪blank=True)


class MyTreeItem(TreeItemBase):
    """And that's a tree item model with additional `css_class` field."""
    css_class = models.CharField('Tree item CSS class', max_length=50)
```

2. Now when *models.py* in your *myapp* application has the definitions of custom sitetree models, you need to instruct Django to use them for your project instead of built-in ones:

```
# Somewhere in your settings.py do the following.
# Here `myapp` is the name of your application, `MyTree` and `MyTreeItem`
# are the names of your customized models.

SITETREE_MODEL_TREE = 'myapp.MyTree'
SITETREE_MODEL_TREE_ITEM = 'myapp.MyTreeItem'
```

3. Run *manage.py syncdb* to install your customized models into DB.

Note: As you've added new fields to your models, you'll probably need to tune their Django Admin representation. See *Overriding SiteTree Admin representation* for more information.

Sitetree definition with custom models

Given the example model given above, you can now use the extra fields when defining a sitetree programmatically:

```
from sitetree.utils import tree, item

# Be sure you defined `sitetrees` in your module.
sitetrees = (
  # Define a tree with `tree` function.
  tree('books', items=[
      # Then define items and their children with `item` function.
      item('Books', 'books-listing', children=[
          item('Book named "{{ book.title }}"',
               'books-details',
               in_menu=False,
               in_sitetree=False,
               css_class='book-detail'),
          item('Add a book',
               'books-add',
               css_class='book-add'),
          item('Edit "{{ book.title }}"',
               'books-edit',
               in_menu=False,
               in_sitetree=False,
               css_class='book-edit')
      ])
  ]),
  # ... You can define more than one tree for your app.
)
```

Models referencing

You can reference sitetree models (including customized) from other models, with the help of *MODEL_TREE*, *MODEL_TREE_ITEM* settings:

```
from sitetree.settings import MODEL_TREE, MODEL_TREE_ITEM

# As taken from the above given examples
# MODEL_TREE will contain `myapp.MyTree`, MODEL_TREE_ITEM - `myapp.MyTreeItem`
```

If you need to get current *tree* or *tree item* classes use *get_tree_model* and *get_tree_item_model* functions:

```
from sitetree.utils import get_tree_model, get_tree_item_model

current_tree_class = get_tree_model()  # MyTree from myapp.models (from the example␣
↪above)
current_tree_item_class = get_tree_item_model()  # MyTreeItem from myapp.models (from␣
↪the example above)
```

Performance notes

To avoid performance hits on large sitetrees try to simplify them, and/or reduce number of sitetree items:

- Restructure (unify) sitetree items where appropriate. E.g.:

```
Home
|-- Category "Photo"
|   |-- Item "{{ item.title  }}"
|
|-- Category "Audio"
|   |-- Item "{{ item.title  }}"
|
|-- etc.
```

could be restructured into:

```
Home
|-- Category "{{ category.title }}"
|   |-- Item "{{ item.title  }}"
|
|-- etc.
```

- Do not use `URL as Pattern` sitetree item option. Instead you may use hardcoded URLs.
- Do not use access permissions restrictions (access rights) where not required.
- Use Django templates caching machinery.
- Use fast Django cache backend.

Note: Sitetree uses Django cache framework to store trees data, but keep in mind that Django's default is Local-memory caching that is known not playing well with multiple processes (which will eventually cause sitetree to render navigation in different states for different processes), so you're advised to use the other choices.

Thirdparty addons

Here are listed addons, helpers, tools that work in a conjunction with SiteTree.

Skip through, maybe you find there something interesting.

django-nav-tree

Application providing a better way to set sitetree item URLs in Django Admin using popup window.

https://github.com/ikresoft/django-nav-tree

Thirdparty applications support

Here belongs some notes on thirdparty Django applications support in SiteTree.

django-smuggler

https://pypi.python.org/pypi/django-smuggler/

Smuggler dump and load buttons will be available on trees listing page if this app is installed allowing to dump and load site trees and items right from your browser.

django-modeltranslation

https://pypi.python.org/pypi/django-modeltranslation/

If you do not want to use the built-in *sitetree* Internationalization machinery, with *modeltranslation* you can localize your tree items into different languages. This requires some work though.

1. Create a custom sitetree item model:

```
# models.py of some of your apps (e.g. myapp).
from sitetree.models import TreeItemBase


class MyTranslatableTreeItem(TreeItemBase):
    """This model will be used by modeltranslation."""
```

2. Instruct Django to use your custom model:

```
# setting.py of your project.
SITETREE_MODEL_TREE_ITEM = 'myapp.MyTreeItem'
```

3. Tune up Admin contrib to handle translatable tree items:

```
# admin.py of your application with translatable tree item model.
from modeltranslation.admin import TranslationAdmin
from sitetree.admin import TreeItemAdmin, override_item_admin


class CustomTreeItemAdmin(TreeItemAdmin, TranslationAdmin):
    """This allows admin contrib to support translations for tree items."""

override_item_admin(CustomTreeItemAdmin)
```

4. Instruct *modeltranslation* how to handle your tree item model:

```
# translation.py of your application.
from modeltranslation.translator import translator, TranslationOptions

from .models import MyTranslatableTreeItem


class TreeItemTranslationOptions(TranslationOptions):

    # These fields are for translation.
    fields = ('title', 'hint', 'description')


translator.register(MyTreeItem, TreeItemTranslationOptions)
```

That's how you made *sitetree* work with *modeltranslation*.

Read *django-modeltranslation* documentation for more information on tuning.

CHAPTER 3

Get involved into django-sitetree

Submit issues. If you spotted something weird in application behavior or want to propose a feature you can do that at https://github.com/idlesign/django-sitetree/issues

Write code. If you are eager to participate in application development, fork it at https://github.com/idlesign/django-sitetree, write your code, whether it should be a bugfix or a feature implementation, and make a pull request right from the forked project page.

Translate. If want to translate the application into your native language use Transifex: https://www.transifex.net/projects/p/django-sitetree/.

Spread the word. If you have some tips and tricks or any other words in mind that you think might be of interest for the others — publish it.

CHAPTER 4

Also

If the application is not what you want for site navigation, you might be interested in considering the other choices — http://djangopackages.com/grids/g/navigation/