Django Stories Documentation

Release 0.3.3beta2

The Washington Times

May 10, 2010

CONTENTS

Contents:

CHAPTER

ONE

INSTALLATION

* Add javascript for advanced and simple modes of editing ### Maybe not

- Move authors to own section and below content.
- Auto-add the user if it is a new story
- Remove pluggable markups from settings and models
- Super Users can see everything!
- new templates for reversion to make it not editable, and show changes (optionally?)

Dependencies:

diff_match_patch (included in lib)

Integration with django-tinymce

Integration with django-reversion

Integration with django-categories

CHAPTER

TWO

SETTINGS

Here are several settings that you can use to customize Stories.

2.1 STORY_STATUS_CHOICES

A story can be in several different states, for example draft vs. live. Your workflow might have several states that a story goes through, but there can only be one choice that is considered "Published".

Choices are specified as a `list` or `tuple` of `integer` - `string` tuples. The `integer` is the code for the choice and the `string` is the description that the user sees.

Defaults:

```
STORY_STATUS_CHOICES = (
    (1, 'DRAFT'),
    (2, 'READY FOR EDITING'),
    (3, 'READY TO PUBLISH'),
    (4, 'PUBLISHED'),
    (5, 'REJECTED'),
    (6, 'UN-PUBLISHED'),
)
```

Draft: A work-in-progress.

Ready for Editing: The story is ready for an editor's touch.

Ready to Publish: The editing is finished and the story is ready to go on to the web site.

Published: The story is on the web site, as long as it is past the story's publish date and time.

Rejected: The editor didn't like something and the author needs to work on it some more.

Un-published: The story has been removed from the site for some reason.

2.2 STORY_DEFAULT_STATUS

When a story is created, what should the the status default to?

Default:

```
STORY_DEFAULT_STATUS = 1 # Draft
```

2.3 STORY_PUBLISHED_STATUS

Which one of the possible statuses is considered "On Site."

Default:

```
STORY_PUBLISHED_STATUS = 4 # Published
```

2.4 STORY_ORIGIN_CHOICES

It is possible that stories could be coming in from several sources, such as a wire service, an editorial front end, or an FTP site. This settings allows you to mark which stories originated from which source, so you can potentially do something different depending on the source. For example, include all stories in the RSS feed, except ones that came from a wire service.

Choices are specified as a `list` or `tuple` of `integer` - `string` tuples. The `integer` is the code for the choice and the `string` is the description that the user sees.

Default:

```
STORY_ORIGIN_CHOICES = (
    (0, 'Admin'),
)
```

2.5 STORY_DEFAULT_ORIGIN

When a story is created from the Django Admin, which choice of origin should it default to?

Default:

```
STORY_DEFAULT_ORIGIN = 0 # Admin
```

2.6 STORY_INCLUDE_PRINT

Should the fields related to print production be included in the database. The fields are `print_pub_date`, `print_section`, and `print_page`.

Default:

```
STORY_INCLUDE_PRINT = False
```

2.7 STORY_RELATION_MODELS

A story can relate to several other things, such as other stories, photographs, photo galleries, and external links. Stories links to the Django Content Types application, which would normally show all sorts of things that don't matter to the author and end users. This setting specifies which specific models are relatable to a story.

The value should be a tuple of *'appname.modelname'* strings.

If this setting is empty or `None`, the story relations are not available in the admin. If at a later time you decide to set this, you must `syncdb` before it will work properly.

Default:

```
STORY_RELATION_MODELS = None # Not enabled
```

2.8 STORY_PAGINATION

Django Stories has a built-in `Paginator` subclass that splits HTML-formatted text into paragraphs for paginating. If `STORY_PAGINATION` is `True`, stories will be paginated in the template. See *Pagination* for more information, and the Django Paginator docs for more about pagination is general.

Default:

```
STORY_PAGINATION = False
```

2.9 STORY_P_PER_PAGE

If `STORY_PAGINATION` is `True`, then this setting sets the number of paragraphs per page for pagination.

Default:

```
STORY_P_PER_PAGE = 20
```

2.10 STORY_ORPHANS

If `STORY_PAGINATION` is `True`, then this setting sets the minimum number of paragraphs allowed on the last page for pagination. This means that with `STORY_P_PER_PAGE = 20` and `STORY_ORPHANS = 4` a story with 24 paragraphs would only have one page, but a story with 25 paragraphs would have two pages.

Default:

```
STORY_ORPHANS = 4
```

CHAPTER

THREE

PAGINATION

Many sites wish to paginate long stories over multiple pages. Paginating text is a bit different from paginating a list of objects. Django Stories has a `Paginator` subclass that takes an HTML-formatted string instead of a `QuerySet`.

The simplest way to use pagination with stories is to set *STORY_PAGINATION* to `True`, which changes the view that handles the story rendering.

3.1 Settings that affect pagination

STORY_PAGINATION Enables or disables pagination altogether. It is disabled by default.

STORY_P_PER_PAGE The number of paragraphs to show on each page. It shows 20 paragraphs by default, but only if story pagination is enabled.

STORY_ORPHANS The minimum number of paragraphs allowed allowed on the last page. It is set to 4 by default (meaning a minimum of 5 paragraphs on a page), but only if story pagination is enabled.

3.2 Using pagination in templates

The default name of the story template is `stories/pag_story.html`. Within the context there are two variables:

story The `Story` object.

story_content The `ParagraphPaginator` class. It contains all the paragraphs of the `Story` that should be on this page. A detailed reference of all the methods is in the Django Paginator docs.

3.2.1 Different heading on first page

```
{% if story_content.has_previous %}
        <h3>{{ story.headline }}</h3>
        <p><em>continued from page {{ story_content.previous_page_number }}</em></p>
{% else %}
        <h1>{{ story.headline }}</h1>
        <h2>{{ story.subheadline }}</h2>
{% endif %}
```

The above template snippet checks to see if this is a page other than 1 (meaning it has a previous page) and displays a small headline with a "continued from page x" below it.

If it is the first page, it displays the headline and subheadline in all their glory.

3.2.2 Looping through the paragraphs

```
{% load add_attribute %}
{% for paragraph in story_content.object_list %}
        {% ifequal story_content.number 1 %}
                {% if forloop.first %}
                        {{ paragraph|add_attribute:"class=dropcap"|safe }}
                {% else %}
                        {{ paragraph|safe }}
                        {% endif %}
        {% endifequal %}
{% endfor %}
```

`add_attribute` is a filter that is included in Django Stories. It adds any attribute to the paragraph. In this example, it checks if it is the first paragraph and adds the attribute `class` with a value of `dropcap` to the `<p>` tag. That part is unnecessary, but allows you some artistic freedom.

Don't forget the `|safe` filter at the end. Django will automatically escape all the tags otherwise.

3.2.3 Leading them to the next page

```
{% if story_content.has_next %}
        <p><a href="?page={{ story_content.next_page_number }}"><em>Story Continues &rarr;</em></a>
{% endif %}
```

Before we hit the typical pagination anchors, it can be nice to add a simple link to the next page, so the reader doesn't have to think about which button to click.

3.2.4 The pagination widget

Django stories includes a template to show a list of pages with previous and next buttons. The template is in `stories/pagination_widget.html` and you can override it should you wish or simple include some styles in your CSS. Add the following line in your template:

```
{% include "stories/pagination_widget.html" %}
```

and it will generate some HTML similar to:

```
<div class="pagination">
        <a href="?page=1" class="previous">&larr; Previous</a>
        <a href="?page=1" class="page">1</a>
        <span class="current">2</span>
        <a href="?page=3" class="page">3</a>
        <a href="?page=3" class="next">Next &rarr;</a>
</div>
```

3.2.5 Pagination widget CSS styles

`div.pagination` The wrapper around the entire widget

`div.pagination a.previous` The anchor for the "previous" link

`div.pagination a.page` The anchor for each page link

`div.pagination span.current` The wrapper for the current page number

`div.pagination a.next` The anchor for the "next" link

CHAPTER

FOUR

INTEGRATION WITH OTHER DJANGO APPLICATIONS

4.1 Integrating with Django Reversion

1. Install django-reversion

   ```
   pip install django-reversion
   ```

2. Add `reversion` to your `INSTALLED_APPS` setting.
3. `./manage.py syncdb`

All the hooks are there to notice when reversion is installed. Versions are tracked from when reversion is first installed.

4.2 Integrating with Django TinyMCE

4.2.1 Install django-tinymce

Django-tinymce makes it very easy to include a GUI text editor for any textarea.

1. Install a special fork of django-tinymce from github

   ```
   pip install git+http://github.com/justquick/django-tinymce.git#egg=tinymce
   ```

2. Download TinyMCE and copy the contents of the `jscript` directory into a `js` directory within your `MEDIA_ROOT` directory.

   ```
   /myproject
       /apps
       /static
           /css
           /img
           /js
   --------->   /tinymce
   ```

3. Add `tinymce` to your `INSTALLED_APPS` setting.

```
INSTALLED_APPS = (
    ...
    'tinymce',
)
```

4. Add `(r'^tinymce/', include('tinymce.urls'))`, to your `urls.py`.

```
urlpatterns = patterns('',
    ...
    (r'^tinymce/', include('tinymce.urls')),
    ...
)
```

5. Add the settings for tinymce. The configuration of plugins and tools shown here is just an example. See the django-tinymce docs and TinyMCE Manual for more information. For example:

```
TINYMCE_DEFAULT_CONFIG = {
    'theme': "advanced",
    'relative_urls': False,
    'plugins': "safari,paste,advimage,preview",
    'theme_advanced_toolbar_location' : "top",
    'theme_advanced_toolbar_align' : "left",
    'theme_advanced_buttons1' : "formatselect,bold,italic,underline,separator,bullist,numlist,se
    'theme_advanced_buttons2' : "",
    'theme_advanced_buttons3' : "",
    #'skin': 'thebigreason',
        'theme_advanced_statusbar_location' : "bottom",
        'width': "600",
        'height': "600",
}
```

6. Add the `TINYMCE_ADMIN_FIELDS` setting:

```
TINYMCE_ADMIN_FIELDS = {
    'stories.story': ('body',),
}
```

You can add other fields for other models in here as well, for example:

```
TINYMCE_ADMIN_FIELDS = {
    'stories.story': ('body',),
    'flatpages.flatpage': ('content',),
}
```

CHAPTER

FIVE

USER MANUAL

5.1 Entering a Story

5.1.1 Field Description

Headline Required. The headline of the story, up to 100 characters.

Subhead A sub-headline, or additional headline, to add more information. Limited to 200 characters.

Teaser Text Required. A brief description that could be used to "tease" the reader into reading the story by clicking on the link.

Body Required. The content of the story. If the GUI is installed, you can use it to format the copy.

Story data

Authors If the author of the story is one or more staff members, select their names here. The byline can be marked up to allow more information.

Non staff author For authors that are not part of the organization, place the information here.

Published Status Required. A workflow status indicator. Only certain stories are allowed on the site.

Enable Comments? Should readers be allowed to comment on the story.

Category Fields

If Categories is integrated, you will see a **Categories** section.

Primary Category Required. The primary category in which to file this story. This may be used in the URL of the story or RSS feeds.

Categories If additional the story would fit into additional categories, select them here.

Advanced Options (hidden)

Advanced options are only required for advanced users.

Origin Required. Defaults to "Admin". Used to track where this story came from, such as a wire source, external system or direct entry.

Slug Required. Defaults to a modified headline. The slug is a unique identifier for the story that people can read. It must be unique for each date. It is commonly used in URLS, and can only contain letters, numbers, dashes and underscores.

Publish Date, Publish Time The date that the story originally was published, automatically filled in to today's date and time for a new story.

Update Date When the story was last modified.

Site Required. By default this is the current site. It is possible to run several sites with shared content.

Story Relations

In order to make publishing to different formats easier (such as RSS/Atom feeds, ePublications, web, and print for example), we don't recommend placing media and other related content inside the story. Story relations are pieces of content that relate to a story in some way. These could be photos, galleries, movies, documents, and other stories for example. they make it easy to mark and relate objects to use differently, depending on the output destination.

Content type Required. One of the configured types of content, such as photo or story.

Object id Required. The Id of the object. Click the magnifying glass to search and select the correct object.

Relation Type This is a free form field to use however you want. It allows you to tag related items for a specific use, such as a lead photo, or a pull quote.

5.2 Revisions

5.2.1 Accessing Revisions

Revisions are maintained every time a change is made. To access the versions, open up the story and click on the History button in the upper-left corner of the window, just left of the View on Site button.

The history list shows each version of this story. The blue timestamps are clickable and lead to a page detailing the content. The list also shows who made the change and a brief description of what they changed. To continue, select a revision from the list by clicking on the timestamp.

Change history: Pernell Roberts, last of 'Bonanza' TV stars, dies : 2010-01-26

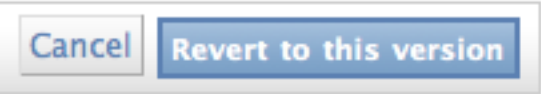
Choose a date from the list below to revert to a previous version of this object.

Date/time	User	Comment
Jan. 25, 2010, 10 a.m.	coordt	Changed body.
Jan. 26, 2010, 7:13 a.m.	coordt	Reverted to previous version, saved on Jan. 25, 2010, 10 a.m.
Jan. 26, 2010, 11 a.m.	coordt	Changed body.
Jan. 26, 2010, 11:01 a.m.	coordt	Changed authors, non_staff_author and status.

Each field of the story is shown with the values it contained in that version. It is not editable. You can leave the screen by clicking the browser's back arrow, or the "Cancel" button at the bottom of the screen.

5.2.2 Reverting to a Previous Version

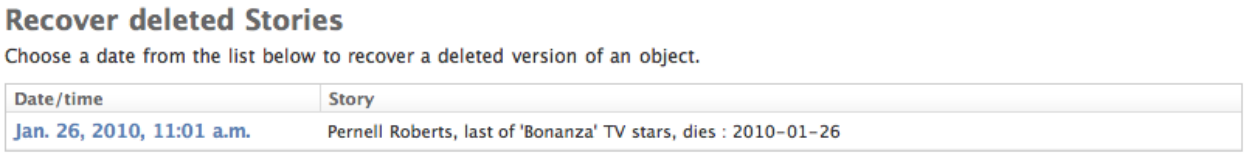
Once you have opened up a previous version and want to revert back to it, click on the "Revert to this version" button.



5.2.3 Recovering a Deleted Story

In the Stories listing, click on the "Recover Deleted Story" button in the upper-right corner of the page.

Then select a deleted story from the list by clicking on the timestamp.



The deleted story will open up as if you were editing it. Click the "Save" button to finish the recovery.

CHAPTER

SIX

REFERENCE MANUAL

6.1 Models

class `Story` (**args, **kwargs*)

A newspaper or magazine type story or document that was possibly also printed in a periodical.

exception `DoesNotExist`

exception `MultipleObjectsReturned`

`author`

Easy way to get a combination of authors without having to worry which fields are set (author/one-off author)

`authors`

`categories`

`get_absolute_url` (**moreargs, **morekwargs*)

`get_next_by_modified_date` (**moreargs, **morekwargs*)

`get_origin_display` (**moreargs, **morekwargs*)

`get_previous_by_modified_date` (**moreargs, **morekwargs*)

`get_related_content_type` (*content_type*)

`get_relation_type` (*relation_type*)

`get_status_display` (**moreargs, **morekwargs*)

`paragraphs`

Return the paragraphs as a list

`primary_category`

`save` (**a, **kw*)

`site`

`storyrelation_set`

CHAPTER

SEVEN

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*

MODULE INDEX

S

INDEX