Django_template3d Documentation

Release 0.0.1

Robert Steckroth

August 27, 2016

Contents

Note: Template3d is not recommended for use with production sites. It is still very experimental.

Template minimizing and compiling will aid the Django template rendering mechanism with regards to efficiency. This becoming more significant when the django template caching loader is used in conjunction with template3d. Without the use of template caching, every request to a template will be read from the disk and rendered to the front-end. Template caching however, stores template code into memory for fast access without IO bandwidth. If your templates are smaller, so will your memory usage. Also, smaller templates use less bandwidth and help protect sensitive code from unwanted cut and paste coders.

CHAPTER 1

Getting Started

The repository is Mercurial based and hosted with Bitbucket here.

Template3d is a self-contained package. It is not required to install template3d into your python path. There is an *example_project* directory contained in the repository which contains a known working model. After `django-admin.py startproject` is executed, the contents of the *example_project* directory will serve as a drop in replacement for the new project root directory. If you wish to use data from the python directory, the Quick Install tutorial is simple to follow.

Note: It is perfectly acceptable to install template3d, and run it completely out of a project directory, at the same time. There is no conflicting structure in the package.

1.1 Quick Install

Template3d can run from the python installation directory while allowing the developer to manipulate its functionality in individual projects. There are no required project settings other than the ones listed below.

Note: If you do not define STANDARD_TEMPLATE_DIRS in the projects settings.py file, templates will be used from the global installation directory. You can retrieve this setting with a `python manage.py template3d -l` and put your original templates there. These will be availible to all projects.

Quick Install:

- `hg clone https://surgemcgee@bitbucket.org/surgemcgee/django_template3d`
- `cd django_template3d/`
- `python setup.py install`
- `cd /var/www`
- `django-admin.py startproject new_project`
- `cd new_project/`
- Copy the `serve` directory from template3d package to the new_project base directory.
- **Add the necessary settings to new_project settings.py**
  - import os
  - PROJECT_DIR = os.path.dirname(__file__)

  - TEMPLATE_DIRS = (os.path.join(PROJECT_DIR, 'serve/templates/'))
  - STATICFILES_DIRS = (os.path.join(PROJECT_DIR, 'serve/'),)
  - Add "template3d" to INSTALLED_APPS
- Add: `url(r'^template3d/', include('template3d.urls')),` to new_project urls.py
- `python manage.py template3d`
- `python manage.py runserver`
- Navagate to http://localhost:8000/template3d

CHAPTER 2

Learning Template3d

Here are some reference pages to aid you in learning the entirety of template3d.

2.1 Commands

Template3d is designed to operate in real time for development while de-coupling itself from the project in production. By default, the Template3d system will turn off when *DEBUG* is set to *False* in the projects settings.py file. This can be overridden as well, See Project Settings

Command line funtionality:

`python manage.py template3d` Run diagnostics on current Django project with Template3d. This will also attempt to set proper permissions of the command script to the `web user`. This needs to be done to protect the file system in case of breach.

[`-o`] Optimize templates for use with the front-end Template3d engine. Use this before the push to a public web server.

[`-c`] Run the Google Code compiler on all Javascript in templates. This cannot be used in the views due to incompatibilities with python subprocess and current slow-ness. Use this before sending a repository to a public web server.

[`-d`] Verify and match the standard templates_3d directory structure to the project templates directory. This will not overwrite any files. These two directories must maintain an identical structure.

[`-l`] Lists current template3d settings for a project (including warnings).

[`-f`] or [`--files`] Single out files to compile or optimize. Accepts one or more template names without path information. e.g. [`python manage.py -ocld -f "infor_page.html test2.html"`] will optimize and compile both templates listed only. Also, settings and warnings will display, and create any non-matching directories.

Warning: The google compiler cannot be accessed in real time through Django views. Usage of `template3d [-o]` is the only way to compile Javascript in templates. MAKE SURE that the template3d views are not processing templates after a compilation! This will overwrite your compiled templates with optimized only ones. See Understanding the Views

2.2 Project Settings

Project settings for template3d operations:

- ***STANDARD_TEMPLATE_DIRS = os.path.join(PROJECT_DIR, 'templates_3d/')*** The path to your original templates. This will be the new templates directory to work from. All templates in this directory will remain unchanged. Note: keep these with read permissions only for security purposes in public domains.
- ***WEB_SERVER_USER='www-data'*** A user on your system with lowered permissions. This user will assume ownership of the command script and compiled templates. It is recommended to to use the systems web server user. Defaults to `www-data` if not set.
- ***T3D_OPTIMIZE = DEBUG*** This will toggle template optimization for the front-end template3d loading engine. Change this to *False* if all templates are to remain un-optimized. This is used when all templates are optimized and/or compiled and ready for public use. Setting this also affects command line access. This is a global setting for the current project.
- ***T3D_COMPILE = True*** This will toggle template compilation with the Google closure project. Change this to *False* if all templates are to remain un-compiled. This is used when all templates are optimized and/or compiled and ready for public use. Setting this also affects command line access. This is a global setting for the current project.
- ***TEMPLATE_DIRS = (os.path.join(PROJECT_DIR, 'templates/')*** This is the normal definition of the template directory. All compiled/optimized templates will be stored here. It is not recommended to store un-compiled templates here (although it is feasible to do so). Any templates here will be used by the `django.template.loaders.cached.Loader` and stored into memory. Note: Template3d will automatically set the owner of any files here to the `WEB_SERVER_USER` with write permissions.

Note: When both `T3D_OPTIMIZE` and `'T3D_COMPILE` are set to *False*, no alteration will be allow to the templates. This includes the command line!

Note: Template3d does not support tuples with the TEMPLATE_DIRS setting. If you set it as a tuple, template3d will issue a warning on the command line and ignore all but the first one listed. (See Commands on how to display project warnings.)

Smaller memory usage can be achieved by combining template compilation with the built in `django.template.loaders.cached.Loader`. The django cached loader will store all templates in `TEMPLATE_DIRS` into memory. This is why we define a separate templates directory `STANDARD_TEMPLATE_DIRS` to store un-compiled templates. This can be commented out or enabled like below. The `django.template.loaders.cached.Loader` takes a list of normal template loaders as options.

To enable template caching

Template caching:

```
TEMPLATE_LOADERS = (
     (
      'django.template.loaders.cached.Loader', (
              'django.template.loaders.filesystem.Loader',
              'django.template.loaders.app_directories.Loader',
                                              ),
     )
)
```

Template caching can be disabled as well

Without Template caching:

```
TEMPLATE_LOADERS = (
              'django.template.loaders.filesystem.Loader',
              'django.template.loaders.app_directories.Loader',
                   )
```

2.3 Template Engine and Tags

The template3d front-end is a experimental concept with high quality results. The current implementation requires Javascript to be inserted into base templates via the `{% t3d_engine %}` tag. There are many helper tags included for developer convenience. Included is a template tag and Javascript explanation.

Note: It is perfectly fine to use only the Javascript functions or a combination of both template tags and Javascript calls.

Template Tags:

- **{% t3d_engine %}**
  - *Inserts the Template3d Javascript engine into the template. Put this into the <head> tag of a first dimension template.*
- **{% t3d_link "t3d_main" "Infor Page" template_name="infor_page.html" %}**
  - ***Couples a link to to a template and {% t3d %} tag. Parameters are single entry.***

    t3d_link

    * Define a link to use with the *{% t3d %}* tag.

    "t3d_main"

    * The corresponding id of a *{% t3d %}* tag.

    "Infor Page"

    * The text to display as the link.

    template_name="infor_page.html"

    * The name of the template to render. Does not include paths or slashes. Can omit the *.html* extension if desired.
- **{% t3d "t3d_id" "t3d_main" "css_class" "t3d_style" "html_tag" "div" %}**
  - ***Parameters are used as space separated pairs. Will default to the above parameters if not set.***

    t3d

    * Define a template area to populate with *{% t3d_link %}*.

    "t3d_id" "t3d_main"

    * The id of the *{% t3d %}* tag. This is used as the first parameter with *{% t3d_link %}*.

    "css_class" "custom_class"

    * The text to display as the link.

    "html_tag" "div"

    * Change the html tag, e.g. span, td, etc..

Javascript functionality: Template3d uses javascript for its optional front-end template loader. The template tags are mere wrappers to insert html and javascript into the templates. Here is what the template3d javascript engine looks like, and how it performs.

The minified template3d engine source code:

```
<script type="text/javascript" id="t3d_engine">function GetHttpRequest(){if(window.XMLHttpRequest)ret
```

The human friendly source:

```
function GetHttpRequest() {
    if ( window.XMLHttpRequest ) // Gecko
         return new XMLHttpRequest() ;
    else if ( window.ActiveXObject ) // IE
          return new ActiveXObject("MsXml2.XmlHttp") ;
                  } // Endo GetHttpRequest()


function load_template( t3d_id, source )  {

       var box_ele = document.getElementById(t3d_id);
       box_ele.innerHTML = "";
       var lines = source.split('\n');
       for ( x = 0; x < lines.length; x += 3 )
           {
             box_ele.innerHTML += lines[x];
             if ( lines[x+2] )
                {
                  var script_params = lines[x+1].split('=');
                  var oHead = document.getElementsByTagName('HEAD')[0];
                  var oScript = document.createElement("script");
                  for ( a = 0; a < script_params.length; a += 2 )
                      {
                        if ( typeof script_params[a+1] == "undefined" )
                             script_params[a+1] = ""
                        oScript[script_params[a]] = script_params[a+1]
                      }
                  oScript.text = lines[x+2];
                  oHead.appendChild(oScript);
                }//Endo if ( lines[x+2] )
           } //Endo for ( x = 0; x < lines.length; x += 3 )
}// Endo load_template( fileUrl, source )


function load_t3d( view, t3d_id ) {
var oXmlHttp = GetHttpRequest();
oXmlHttp.onreadystatechange = function() {
if ( oXmlHttp.readyState == 4 )
     if ( oXmlHttp.status == 200 || oXmlHttp.status == 304 )
        {
          load_template(t3d_id, oXmlHttp.responseText );
          fade_template(t3d_id);
        }
     else
          parent.location=view
          // Redirect to hard url for de-buging
}//Endo oXmlHttp.OnReadyStateChange = function()
oXmlHttp.open('GET', view, true);
oXmlHttp.send(null);
```

```
} // Endo AjaxPage(sId, url)

</script>
```

The concept of this engine is simple.

`onclick="load_t3d( view, t3d_id );"`

- Pass the view name or template name as the first argument to `load_t3d( view, t3d_id )`
- The second argument is the `id` of the html tag where the template data is to be loaded.

2.4 Understanding the Views

The below views are used for development purpose with the front-end template3d loading engine. See Template Engine and Tags
It is not necessary to use this if template compilation/minification is your only use with template3d.

View:

```
def template3d_name(request, page=None):

  if settings.DEBUG:
    t = select_template([template3d().parse_template(page+'.html')])
  else:
    t = select_template([page+'.html'])

  c = RequestContext(request, {})
  return HttpResponse(t.render(c))
```

The `select_template()` function returns rendered template data to the view. It will also parse the templates directory for a given *template name*. This is why it is only necessary to provide the name of the template.

The `template3d().parse_template()` command will parse the standard templates directory, (defined as STANDARD_TEMPLATE_DIRS), for the provided template name. It returns this name string back to any parent functions, e.g. `select_template()`. Also, template optimizations will occur on the template and be placed into the django templates directory (defined as TEMPLATE_DIRS). This is how the real time optimization works for development.

We return an `HttpResponse` with the rendered data and context data (if any). It is not necessary to use Django shortcuts here such as `render`. You may use the render command however by changing the view structure like this.

Shortcut View:

```
def template3d_name(request, page=None):

  if settings.DEBUG:
    return render(request, template3d().parse_template(page+'.html'), {} )
  else:
    return render(request, page+'.html' , {} )
```

Note: Do not use the `render` shortcut with `select_template()`. This will cause the template to be rendered twice, thus lowering efficiency.

2.5 Developers

2.5.1 Understanding the Template3d class

class template3d():

- **t3d = template3d()**
  - The __init__ definition contains primary error checking and is ran any time the template3d() class is referenced. Therefore, the above statement will execute all error checking. Subsequent commands using the t3d object will not trigger the __init__ definition. A simple call to template3d() can also be used to return exceptions and messages.
- **template_3d.parse_templates()**
  - The nuts and bolts of template3d. This definition will: parse a single template, optimize for the font-end engine, compile the javascript, and put the results into the *TEMPLATE_DIRS* directory.
- **template_3d.create_directories()**
  - This function will parse the STANDARD_TEMPLATE_DIRS directory and match the TEMPLATE_DIRS to it. It is important to keep both of these directories in sync. Is is safe to include the [`-d`] argument into your template3d commands. It will not overwrite or replace any existing directories or contents.
- **template_3d.check_directories()**
  - Only checks for non-matching directory structure in STANDARD_TEMPLATE_DIRS to the TEMPLATE_DIRS structure. This will not create any directories and outputs an error if inconsistencies are found. Returns true if directories are the same.
- **template_3d.toggle_engine([`bool`][`bool`])**
  - Toggles the `parse_template()` definition to optimize templates in the front-end real time template loader and/or compilation with the Google closure tools. e.g. `t3d.toggle_engine(False, True)` will turn off the optimization and turn on closure compilation in the view or script. This is a direct override of the *T3D_OPTIMIZE* and *T3D_COMPILE* settings. Should be used in views.py or other scripts as a global template3d() class instance. e.g. `t3d = template3d()` at the top of the script.
- **template_3d.check_duplicates()**
  - Due to the nature of template3d, duplicate template names are not supported anywhere in your *STANDARD_TEMPLATE_DIRS* directory. Template3d uses functions like `select_template()` which parse the *TEMPLATE_DIRS* directory and return the first template it finds. Therefore, the `check_duplicates()` definition will find any templates with the same name and return it as a warning.
- **template_3d.by_filename(filename_list)**
  - Accepts a list of *template names* and sends them one by one to the `parse_templates()` definition for optimizing/compiling.
- **template_3d.create_all()**
  - Will parse the *STANDARD_TEMPLATE_DIRS* directory for all files ending with *.html* and send them one by one to the `parse_templates()` definition for optimizing/compiling.
- **template_3d.list_settings()**
  - **Ad simple call to return all project settings and warnings together. Useful for debugging information.**
    e.g. `print '\n%s' % template_3d.list_settings()`

- **template_3d.add_warning(["message'], ["lable"])**
  - **Adds a warning message to the warning_queue list. This is a list variable with class scope for persistence func**

    e.g. template3d.add_warning('Hmmmm, this is kinda bad. File '+filename+' is mi
    ```
    template3d.add_warning('A simple warning to display')
    ```
- **template_3d.list_warnings()**
  - **Returns a new line formatted string from the class list template3d.warning_queue. The template_3d.w**

    e.g. print template_3d.list_warnings() print template_3d.warning_queue[0]

CHAPTER 3

Concepts

For those of you who like to play with ideas

3.1 Tips and Ideas

Features:

```
- Compile/minimize native templates without adding tags, denoting, or reformatting.
- Smaller memory footprint with template caching
- Works in conjunction with a real time template loading front-end.
- Seamless development environment to a static production model.
- Self contained package can be easily used with or without installing the package.
```