django-ad-code Documentation

Release 1.0.0

Mark Lavin

Apr 21, 2018

Contents

django-ad-code is a reusable application for managing and rendering ad tags from third-party ad server or ad network such Adsense, DoubleClick or OpenX.

django-ad-code is *not* an ad server or full ad management system. It is simply a tool to help you manage the ad tags needed to use an ad network.

CHAPTER 1

Installation

django-ad-code requires Django >= 1.8 and Python 2.7 or 3.3+.

To install from PyPi:

```
pip install django-ad-code
```

CHAPTER 2

Documentation

Documentation on using django-ad-code is available on Read The Docs.

CHAPTER 3

License

django-ad-code is released under the BSD License. See the LICENSE file for more details.

CHAPTER 4

Contributing

If you think you've found a bug or are interested in contributing to this project check out django-ad-code on Github.

CHAPTER 5

Contents

5.1 Getting Started

Below are the basic steps need to get django-ad-code integrated into your Django project.

5.1.1 Installation

It is easiest to install django-ad-code from PyPi using pip:

```
pip install django-ad-code
```

5.1.2 Configure Settings

You need to include `adcode` to your installed apps as well as include a context processor in your project settings.

```
INSTALLED_APPS = (
    # Other installed apps would go here
    'adcode',
)

TEMPLATE_CONTEXT_PROCESSORS = (
    # Other context processors would go here
    'adcode.context_processors.current_placements',
)
```

Note that `TEMPLATE_CONTEXT_PROCESSORS` is not included in the default settings created by `startproject`. You should take care to ensure that the default context processors are included in this list. For a list of default `TEMPLATE_CONTEXT_PROCESSORS` please see the official Django docs.

For the context processor to have any effect you need to make sure that the template is rendered using a RequestContext. This is done for you with the render shortcut.

5.1.3 Create Database Tables

You'll need to create the necessary database tables for storing ad sections and placements. This is done with the `syncdb` management command built into Django:

```
python manage.py syncdb
```

django-ad-code uses South to handle database migrations. If you are also using South then you should run `migrate` instead:

```
python manage.py migrate adcode
```

There is an optional fixture with IAB ad sizes based on the 2012 guidelines. To load this fixture run:

```
python manage.py loaddata iab_sizes.json
```

5.1.4 Using Ad Data in the Template

The django-ad-code includes two template tags to help rendering ad placements. `render_section_header` would be included in your html <head> and would include and step JS needed. In your <body> you would render individual placements with `render_placement` which takes the slug for the placement.

```
{% load adcode_tags %}
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- Other meta, css, js -->
    {% render_section_header %}
</head>
<body>
    <!-- Various body content -->
    {% render_placement 'footer' %}
</body>
</html>
```

Continue on to learn about customizing how these tags render.

5.2 Customizing the Ad Templates

Setting up the templates to render your ad code is straight forward. This section details how the appropriate templates are discovered when using the django-ad-code template tags.

5.2.1 Customizing `render_section_header` Template

`render_section_header` renders the header content for the current section. If no section was matched then it will not render anything. This template tag looks for the template `adcode/{{ section.slug }}/header.html` then `adcode/header.html` where `{{ section.slug }}` is the slug of the current matched section. In most cases it will be sufficient to define `adcode/header.html` to use for all sections.

The following items are passed into the context for this template.

- `section`: The current adcode Section
- `placements`: The full list of Placements for this section

- debug: The value of settings.DEBUG
- MEDIA_URL: The value of settings.MEDIA_URL
- STATIC_URL: The value of settings.STATIC_URL

5.2.2 Customizing render_placement Template

render_placement renders a given placement by the slug. If the placement could not be found then it will not render anything. The template search order is adcode/{{ section.slug }}/{{ placement.slug }}-placement.html, adcode/{{ section.slug }}/placement.html then adcode/placement.html. This allows you to customize each placement individually if needed or on a section or simply a global basis. Many use cases will only require defining adcode/placement.html.

The following items are passed into the context for this template.

- section: The current adcode Section
- placement: The current Placement to be rendered
- debug: The value of settings.DEBUG
- MEDIA_URL: The value of settings.MEDIA_URL
- STATIC_URL: The value of settings.STATIC_URL

The Placement model contains a placeholder property that can be used for local development to test the layout. An example adcode/placement.html might look something like below.

```
{% if debug %}
    <img alt="{{ placement }}" title="{{ placement }}" src="{{ placement.
↪placeholder }}">
{% else %}
    <!-- Here you would put the actual ad code needed -->
{% endif %}
```

This placeholder image will match the size of the placement. By default this will use Placehold.it but you are free to customize it with the ADCODE_PLACEHOLDER_TEMPLATE setting.

```
# Default setting (not required in settings.py)
ADCODE_PLACEHOLDER_TEMPLATE = 'http://placehold.it/{width}x{height}'

# Use placekitten instead
ADCODE_PLACEHOLDER_TEMPLATE = 'http://placekitten.com/{width}/{height}'
```

5.3 Using django-ad-code with DoubleClick

DoubleClick is an ad serving and ad management tool owned and run by Google. There is also a DoubleClick for Publishers (DFP) Small Business. This section details using django-ad-code to work with the asynchronous Google Publisher Tag code.

This not meant to be a comprehensive guide on using DoubleClick only a guide on integrating your DoubleClick inventory with django-ad-code.

Note: This documentation is primarily for example purposes and should not be taken as an endorsement of DoubleClick.

5.3.1 Header Template

The `adcode/header.html` should contain the asynchronous googletag code as well as define the slots for your page based on the current set of placements. An example of what this might look like is given below.

```
{% if section and placements and not debug %}
<script>
    var googletag = googletag || {};
    googletag.cmd = googletag.cmd || [];
    (function() {
    var gads = document.createElement('script');
    gads.async = true;
    gads.type = 'text/javascript';
    var useSSL = 'https:' == document.location.protocol;
    gads.src = (useSSL ? 'https:' : 'http:') +
    '//www.googletagservices.com/tag/js/gpt.js';
    var node = document.getElementsByTagName('script')[0];
    node.parentNode.insertBefore(gads, node);
    })();
</script>
<script>
    googletag.cmd.push(function() {
        {% for placement in placements %}
        googletag.defineSlot(
            '{{ placement.remote_id }}', [{{ placement.width }}, {{␣
↪placement.height }}], 'div-gpt-ad-{{ placement.id }}'
        ).addService(googletag.pubads());
        {% endfor %}
        googletag.pubads().enableSingleRequest();
        googletag.enableServices();
    });
</script>
{% endif %}
```

Here you can see that the `Placement.remote_id` stores the ad unit name. You can adapt this to fit your needs to include additional targetting. See the prior section on customizing the header template.

5.3.2 Placement Template

The `adcode/placement.html` is responsible for rendering the individual placements in the body content. The element `id` needs to match the `id` given in `defineSlot` call in the header. In this example we used `'div-gpt-ad-{{ placement.id }}'` so we will be consistant in the placement template.

```
<div id="div-gpt-ad-{{ placement.id }}">
    {% if debug %}
        <img alt="{{ placement }}" src="{{ placement.placeholder }}">
    {% else %}
        <script type="text/javascript">
            googletag.cmd.push(function() {
                googletag.display("div-gpt-ad-{{ placement.id }}");
            });
        </script>
    {% endif %}
</div>
```

This will render the placeholder image if `DEBUG=True`. If necessary this can be customized on a per section or per placement basis.

5.4 Using django-ad-code with OpenX

OpenX is an ad server solution with both enterprise/managed and open source editions. It integrates with the OpenX Market for advertisers to bid on your available ad space. Here we will detail how you can configure django-ad-code to work with the OpenX single page call configuration based on http://www.openx.com/docs/2.8/userguide/single%20page%20call.

This will not cover setting a managed OpenX account or a self-managed server. It will only cover integrating an existing OpenX setup with django-ad-code. It is primarily based on using the managed OpenX server.

Note: This documentation is primarily for example purposes and should not be taken as an endorsement of OpenX.

5.4.1 Header Template

The `adcode/header.html` should contain the `spcjs.php` script tag including the account id. The below example is using a managed OpenX account server.

```
{% if section and placements and not debug %}
<script type='text/javascript' src='http://d1.openx.org/spcjs.php?id=XXXX'></
↪script>
{% endif %}
```

Here `XXXX` is account id for the managed account id. There are additional options that can be configured with this script tag. See *Websites & Zones -> Website properties -> Invocation Code* tab for more options.

5.4.2 Placement Template

The `adcode/placement.html` is responsible for rendering the individual placements in the body content. These placements are called zones in the OpenX documentation.

```
<div id="div-openx-ad-{{ placement.id }}">
    {% if debug %}
        <img alt="{{ placement }}" src="{{ placement.placeholder }}">
    {% else %}
        <script type="text/javascript"><!--// <![CDATA[
            OA_show({{ placement.remote_id }});
        // ]]> --></script>
    {% endif %}
</div>
```

Here you can see the `remote_id` in the Placement model corresponds to the OpenX zone id. More options exist for generating this tag which could be included in this placement template such as a `noscript` option. See the *Zones > Invocation Code* tab for a full list of these options.

5.5 Available Settings

Below are the settings available for configuring django-ad-code.

5.5.1 ADCODE_PLACEHOLDER_TEMPLATE

The placement model has a `placeholder` property. This is used to render a placeholder image for debugging. This setting can be used to configure the placeholder image service used.

Default: `'http://placehold.it/{width}x{height}'`

5.5.2 ADCODE_CACHE_TIMEOUT

This configures the cache timeout for the section and placement cache.

Default: 12 hours

5.6 Release History

Release and change history for django-ad-code

5.6.1 v1.0.0 (Released 2018-04-21)

This release adds support for Django 1.8-2.0. Previous versions of Django are no longer supported. This puts this project on a stable footing for the next few years of Django support and that is now reflected in the version.

5.6.2 v0.5.0 (Released 2014-09-07)

This release adds support for 1.7 and the new style migrations. If you are using Django < 1.7 and South >= 1.0 this should continue to work without issue.

For those using Django < 1.7 and South < 1.0 you'll need to add the `SOUTH_MIGRATION_MODULES` setting to point to the old South migrations.

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

No new migrations were added for this release but this will be the new location for future migrations. If your DB tables are up to date from v0.4.X then upgrading to 1.7 and running:

```
python manage.py migrate adcode
```

should automatically fake the initial migration using the new-style migrations.

- Added support for Django 1.7 migrations
- Added customizable AppConfig for Django 1.7
- Refactored signal handling for caching to use the AppConfig if available
- Dropped Django 1.3 support. Minimum version is now Django 1.4.2
- Dropped support for Python 2.6

5.6.3 v0.4.1 (Released 2013-06-08)

- Reorganized test suite to ensure compatibility with test runner in Django 1.6
- Refactored Travis CI integration

5.6.4 v0.4.0 (Released 2012-12-19)

A fairly minor release and users should feel safe to upgrade. Beyond some helpful new documentation there is experimental support for Python 3.2+. This requires using Django 1.5 or higher.

Features

- Documentation for integrating DoubleClick tags
- Documentation for integrating OpenX tags
- Travis CI integration
- Experimental Python 3 (3.2+) support

5.6.5 v0.3.1 (Released 2012-05-19)

Bug Fixes

- Added missing IAB sizes from the Universal Ad Package. (Issue #10)
- Fixed minor formatting issue with template syntax error message.

5.6.6 v0.3.0 (Released 2012-05-16)

This release made a few changes to the model fields. To upgrade you should run:

```
python manage.py migrate adcode
```

This assumes you are using South. Otherwise you should manually add the `priority` column to your `adcode_section` table.

Features

- Added width/height properties to Placement model.
- Added priority field for Sections to resolve overlapping regex patterns.
- Additional defaults for model admin clases.
- Caching and performance improvements.
- Fixture for standard IAB ad sizes.

5.6.7 v0.2.0 (Released 2012-05-12)

Features

- Template tag for rendering section headers and individual placements.

5.6.8 v0.1.0 (No public release)

- Initial version.

Features

- Basic ad section/placement models.
- Context processor for accessing model data in the template.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search