Django-Vitae Documentation

Release 0.0.1

Michael Bader

Aug 03, 2020

Contents

1 Overview **1**

2 Installation **3**

3 Organization of the Documentation **5**

4 Contributing to Django-Vitae **7**

5 Indices and tables **9**

6 Documentation Contents **11**

6.1 Getting Started 11
6.2 Complete Vitae Views 13
6.2.1 HTML 13
6.2.2 PDF 14
6.3 CV Sections 14
6.3.1 Achievements 14
6.3.2 Publications 15
6.3.3 Other Works 24
6.4 Shortcuts 26
6.5 Settings 26
6.5.1 `CV_PUBLICATION_STATUS_CHOICES` 26
6.5.2 `CV_FILE_TYPES_CHOICES` 26
6.5.3 `CV_STUDENT_LEVELS_CHOICES` 27
6.5.4 `CV_SERVICE_TYPES_CHOICES` 27
6.5.5 `CV_KEY_CONTRIBUTORS_LIST` 27
6.6 Module Reference 28
6.6.1 `cv.models` 28
6.6.2 `cv.models.managers` 43
6.6.3 `cv.views` 44

Python Module Index **45**

Index **47**

CHAPTER 1

Overview

Django-Vitae allows users to make highly customizable curricula vitae for use on their websites. The application provides models for common entries on curricula vitae such as education, employment, publications, teaching, and service. Django-Vitae eliminates many of the repetitive tasks related to producing curricula vitae. The included templates provide a complete CV "out of the box", but allows researchers who might be interested to customize the format using Django templating language.

CHAPTER 2

Installation

A stable version of Django-Vitae is available in the Python Package Index and can be installed using `pip`:

```
$ pip install django-vitae
```

The latest development version can be obtained from GitHub:

```
$ git clone https://github.com/mikebader/django-vitae
$ cd django-vitae
$ python setup.py install
```

If you do not have experience with Django, you might be interested in the *Getting Started* guide.

CHAPTER 3

Organization of the Documentation

- *Complete Vitae Views*
  - *HTML*
  - *PDF*
- *CV Sections* documents the API to write lines on CV by different sections on a CV
  - *Achievements* (*Degrees* | *Positions* | *Awards*)
  - *Publications* (*Articles* | *Books* | *Chapters* | *Reports*)
  - *Other Works* (*Grants* | *Talks* | *Other Writing* | *Datasets*)
  - Teaching
  - Service
- Templates
  - Template tags & filters
  - Template structure
- *Settings*
- Module Reference

CHAPTER 4

Contributing to Django-Vitae

It's quite possible that Django-Vitae does not include all types of publications necessary. You may open an issue, or–even better–contribute code for other common types of publications not already incorporated into Django-Vitae.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

CHAPTER 6

Documentation Contents

6.1 Getting Started

To get started with Django-Vitae, make sure that you have Python (version 3.5 or later) installed on your machine.

You might want to work in a virtual environment. If you know what those are, go ahead and set one up; if not, then don't worry it (you may want to learn how to if you end up using Python a lot, but if this is your only project, it's not a big deal).

Now you will want to create a directory where you will store all of the files for your CV. Move inside that directory (the `$` represents the command line where you enter text, don't include it in what you type):

```
$ mkdir my_cv
$ cd my_cv
```

Once you are in that directory, you will install Django-Vitae. This will also install Django and a few other Python packages:

```
$ pip install django-vitae
```

Once you have installed Django-Vitae and all of its dependencies, you will start a Django project. This opens up all of Django's magic to help you create your CV. In the example below, your Django project would be called `myvitae`, but you can choose any name you wish as long as the name does not conflict with built-in Python module names. After you make the project, you will move into the directory created for the project, which will have the same name as the project (`myvitae` in this case):

```
$ django-admin.py startproject myvitae
$ cd myvitae
```

Next comes the trickiest part. You will need to edit two different files. Both are in the `myvitae` subdirectory. This can be confusing: you will have two layers of directories, both named `myvitae` (or whatever you chose to call your project). The files we will be editing are in the directory lower in the hierarchy.

The first file is called `settings.py`. Open the file in a text editor of your choice and you will see something that looks like the following:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

At the end of that list, you will want to add two lines so that it looks like this (make sure you include the quotes):

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django_widgets',
    'cv',
]
```

Save the `settings.py` file and close it.

Now, open up the file `urls.py`. Look for the following line:

```
from django.urls import path
```

and change it to:

```
from django.urls import path, include
```

Then, in the same file, you will find the part that looks like this:

```
urlpatterns = [
    path('admin/', admin.site.urls),
]
```

and you will change it to look like this:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('cv.urls', namespace='cv')),
]
```

Save the `urls.py` file and close it. The hard part is done!

Now, in your Terminal you will need to run a series of commands from the *top level* `myvitae` directory (the one directly under `my_cv` if you've used the same names as those used in this guide). These will set up your database (each will produce some text on the screen that you don't need to worry about):

```
$ ./manage.py makemigrations
$ ./manage.py migrate
```

After those commands complete you will run another command that will set up a "superuser" that allows you administrative access to your project. Type:

```
$ ./manage.py createsuperuser
```

You will be prompted to enter a username, an email, and a password.

After you have set all of that up, you will now create a local version of your CV website. To do that, you enter the command:

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

Now open your browser of choice go to http://localhost:8000/admin or http://127.0.0.1:8000/admin. You will see, if everything has gone correctly, a login screen asking for your username and password. These are the same as what you just entered to create your superuser. After you successfully log in, you will see an interface where you can edit all of the entries for you CV. After you so so, you can then point your browser to http://localhost:8000/ to see your CV (if you log out from the admin site, you will not see the add and edit buttons).

And you, my friend, are on your way to making your own vitae!

6.2 Complete Vitae Views

Django Vitae provides two primary views that represent the entire CV document: HTML and PDF.

6.2.1 HTML

The primary view provided by Django Vitae represents a CV as a webpage. This is the view made available at the application's root URL, that is `/`. The URL retrieves the view `cv.views.CVView` that gathers the data from individual models and presents them in appropriate sections.

Template Structure:

```
cv/
  sections/
      <plural model name>.html
  base.html
  cv.html
  skeleton.html
```

The HTML views use a series of templates layered on top of one another. At the lowest level, `cv/skeleton.html` **defines the main structure for the page**. The default template uses CSS styles and Javascript from *Bootstrap* and icons from Font Awesome icons, using their respective CDNs.

At the next layer, the `cv/base.html` template inherits from `cv/skeleton.html` and **defines the order of sections** as a series of Django template blocks. This is done by using blocks from Django templates. The name of each block corresponds to the the plural of the model name, except the blocks for *`OtherWriting`* and `Service` are named `otherwriting` and `service`.

The template `cv/cv.html` inherits from the `cv/base.html` template and **defines the style for each section**. In the default template, each block consists of a `<div>` block and then includes the section template in the `templates/cv/sections` directory. The section template is an html file named for the plural form of the section name (except for `OtherWriting` and `Service`, as above); for example, the section template for articles would be the file `templates/cv/sections/articles.html`. If you would like to customize the look of an individual section, you should save a file with that name in the `cv/sections/` subdirectory of the template directory of your own project.

6.2.2 PDF

Django Vitae will also create a PDF of your CV “on-the-fly”.

The PDF version of your CV can be found at the `/pdf/` URL. The URL retrives the view `cv.views.pdf.cv_pdf`. The view gathers data from different sections of the CV and then creates a PDF using the Report Lab library.

Template Structure:

```
cv/
  pdf/
      pdf_list.json
      <model name>.html
```

Creating PDFs requires that much of the style be controlled internally in the code. The internal coding makes it difficult to customize the *style* of the PDF version of the CV. The *content* can be customized, however, by using templates.

The content of the PDF, including the order, is controlled by the template `pdf_list.json` JSON file. The JSON file is structured as a list of dictionaries. Each dictionary **must** have a `model_name` key that is the model name in lowercase. In addtion, the dictionary **may** have the following keys:

`display_name` A string of the section heading (including any capitalization that you desire)

`date_field` May either be a string representing the name of the field that you would like to use to display as the date in each entry for that section **or** a list of two strings, the field names to be used to render the start and end dates.

`subsections` A list of lists; each of the sub-lists should include two string values: the first contains the heading for the subsection and the second is a string representing the method of the `displayable` manager to use to get the queryset for that subsection.

The `templates/cv/pdf/` also contains an XML file for each section of the PDF. The XML files use the intra-paragraph markup described in the ReportLab User Guide (subsection 6.3) that include the `<i>` tag for italics, `<b>` for boldface, and `<a>` for links (among others).

6.3 CV Sections

6.3.1 Achievements

The first sections of CVs list one’s achievements. The models below allow CV authors to record these achievements.

Degrees

The *`Degree`* model stores instances of degrees earnned and has three required fields:

- `degree`
- `date_earned`
- `institution` that granted the degree

The `Degree` model inherits from *`DisplayableModel`* and therefore has an `extra` field that can be used to enter information about the degree. For the special case of honors, the `Degree` model has a field, `honors`, that allows information such as whether a degree was attained *cum laude*.

Model instances are sorted in reverse chronological order using the `date_earned` values.

Positions

The *Position* model stores instances of jobs or research experience and has three required fields

- title
- start_date
- institution

The model also contains fields that allow the user to specify the department in which the user worked, as well as a project within the department.

Model instances are sorted in reverse chronological order by the end_date field first and the start_date second.

The model also has a Boolean field primary_position that allows the user to indicate if the position represents the primary title. The primary_position field is used, for example, in the heading of the *HTML* and *PDF* views. The model also comes with a primary_position manager that accesses the *PrimaryPositionManager* that returns only Position instances marked as being primary positions.

Awards

The *Award* model stores instances of honors or awards that the user has received. The model has three required fields:

- name of the award
- organization that grants the award
- date of award

The model also has a description field that can be used to provide more information about the award.

6.3.2 Publications

Publications are the central component of Django-Vitae since publications are the key element of CVs. Django-Vitae includes four types of publications: *books*, *articles*, *chapters*, and *reports*. These models share some *common features*.

Common Features

Publications, regardless of type, all have some common traits such as titles and lists of authors. Django-Vitae defines a number of common features across the four different types of publications. Internally, Django-Vitae does this by defining a series of abstract classes. The different publication models inherit from the *VitaePublicationModel* abstract model.

Common Fields

The following fields are common across the four types of publications:

title The title of the publication (**required**).

short_title A shortened title of the publication with a maximum length of 80 characters (**required**).

This can be the "running head" of a publication. Django-Vitae uses the slugified version of the short title to construct URLs for the item.

slug A slugified version of the short-title to use in URLs (**required**).

The slugs are automatically constructed from the short_title in admin.

`abstract` An abstract or summary of the publication. Expects markdown formatting.

`status` The point in the publication process where the publication currently rests (**required**).

All publication models include an `status` field, which represents the where in publication process the publication currently exists. Django-Vitae implements the `status` field by using an `IntegerField` with the `choices` parameter defined in `CV_PUBLICATION_STATUS_CHOICES`. The default values of the `PUBLICATION_STATUS_CHOICES` setting are:

Integer	Status
0	In preparation
1	Working paper
20	Submitted
30	Revision for resubmission invited
35	Resubmitted
40	Conditionally accepted
50	Forthcoming
55	In press
60	Published
99	"Resting"

A user may *customize* the integer values and labels by defining their own `CV_PUBLICATION_STATUS` option in their `settings.py` file.

`pub_date` The date that the publication was published in final form.

`primary_discipline` The discipline to which the publication contributes most directly.

A `ForeignKey` relationship to a `cv.models.Discipline` object. Can be useful for researchers who work in multiple disciplines to separate their CV into sections for each discipline.

`other_disciplines` Disciplines other than the primary discipline to which the publication contributes.

A `ManyToManyField` relationship to `cv.models.Discipline` objects.

Each publication model contains non-editable fields managed internally that can be accessed for instances of the model:

- **`abstract_html` that converts text entered in Markdown in** `abstract` field to html, and
- `is_published`
  `is_inrevision`
  `is_inprep`: set as booleans based on the status of the publication when saved.

Ordering

The publication models order model instances by `status` in ascending order then by `pub_date` in descending order. This places the publications with the highest probability of changing at the top of sorted lists.

Note: The publication models do not use `pub_date` field to identify published articles and the built-in templates do not print the `pub_date` field. Therefore, users can use the `pub_date` field to order unpublished manuscripts in a convenient order.

Managers

For all types of publications, users may access instances of publication with the `displayable` custom manager. In addition to the `all()` method that returns all objects for which the `display` attribute is `True`, the manager also includes three other methods:

`published` returns all publications that have been accepted for publication or published (forthcoming, in press, and published).

`revise` returns all publications that are in the process of submission or revision (submitted, under revision for resubmission, resubmitted, or conditionally accepted).

`inprep` returns all publications being prepared for submission and publication.

Note: The custom managers the include multiple statuses retain the default ordering of the model (that is, they are ordered by `status`, then `pub_date`, then `submission_date`).

Authorship Sets

Publication types also share the common trait of having authors. More precisely, publications have *authorships* since a list of authors contains information, such as the order of authorship.

For all publication type models, Django-Vitae includes an `authorship` attribute that returns a `QuerySet` of authorships, e.g.:

```
>>> from cv.models import Article
>>> article = Article.objects.all().first()
>>> article.authorship.all()
<QuerySet [<ArticleAuthorship: Kahneman, Daniel>,
   <ArticleAuthorship: Tversky, Amos]>]
```

Internally, the authorship attributes are implemented as a `django.db.models.ManyToManyField` that relate an instance of the publication type (e.g., `Article`, `Book`, etc.) to `Collaborator` through a third model.

Authorship models for all publication types have three common fields:

`display_order` Integer that classifies the position of the author in the list of authors (**required**)

`print_middle` Boolean that indicates whether the author's middle initials should be printed in list of authors (default=True)

`student_colleague` Choice field with possible values defined by *CV_STUDENT_LEVELS_CHOICES* setting; allows display of student collaborations

Custom Methods

Each of the publication models includes the custom functions, `get_previous_published()` and `get_next_published()` that will return next and previous *published* instance of the model using the `pub_date` field.

Note: The `get_previous_published()` and `get_next_published()` functions are designed to emulate the Django built-in methods `get_next_by_FOO` and `get_next_by_FOO`

Articles

Article Model

Model field reference	*cv.models.publications.Article*
Authorship set	*cv.models.publications.ArticleAuthorship*

The *Article* model represents an instance of an article or other publications with similar characteristics as articles (e.g., proceedings).

Article Views

Article List [`cv.views.CVListView`]

Context object	`{{object_list}}`
Template	`'cv/lists/article_list.html'`
URL	`'articles/'`
MIME type	`text/html`

The article list view produces a page with a list of an author's articles. This may be helpful if an author does not wish to display a full CV, but wants to list just their articles. The page renders an instance of the `cv.views.CVListView` view with the named parameter `model_name` set to `'article'`. The view returns the `{{object_list}}` in the context with four objects on its dot path:

`total_articles` Integer of total number of article objects from all three status-based managers:

`article_published_list` queryset of all published articles (uses the `published()` method of the *PublicationManager*)

`article_revise_list` queryset of all articles in the revision process (uses the `revise()` method of the *PublicationManager*)

`article_inprep_list` queryset of all articles in preparation for submission (uses the `inprep()` method of the *PublicationManager*)

The URL can be accessed in templates by using the URL template filter with the named URL `section_list` and `model_name` parameter equal to `article`, i.e.:

```
{% url section_list model_name='article' %}
```

Article Detail: `cv.views.CVDetailView`

Context object	`{{article}}`
Template	`'cv/details/article_detail.html'`
URL	`'articles/<slug:slug>/`
MIME type	`text/html`

The article detail view produces a page that represents a single article. The default template includes the title, the abstract, a link to the published version of the article (if published and a URL is defined), and links to download the citation in both RIS and BibTeX formats (described below). The page is rendered as an instance of the class `cv.views.CVDetailView` with the named parameters `model_name` set to `'article'` and `slug` set to the article's slug attribute. The view returns the context `{{article}}` that represents the `Article` instance.

The URL can be accessed using the named URL `item_detail` with `model_name` set to `'article'` and `slug` set to the article's slug attribute, i.e.:

```
{% url item_detail model_name='article' slug='slug-from-short-title' %}
```

Article Citation: `cv.views.citation_view()`

Context object	`{{article}}`
Templates	`'cv/citations/article.ris'` `'cv/citations/article.bib'`
URL	`'articles/<slug:slug>/cite/<str:format>/'`
MIME type	`application/x-research-info-systems` or `application/x-bibtex`

Returns view to allow citation to be downloaded to citation management software.

The `<str:format>` named parameter should be one of:

`'ris'` will create downloadable citation using Reference Manager format specification (see http://endnote.com/sites/rm/files/m/direct_export_ris.pdf).

`'bib'` will create downloadable citation using the BibTeX format specification (see http://www.bibtex.org/Format/)

Books

Book Model

Model field reference	*`cv.models.publications.Book`*
Authorship set	*`cv.models.publications.BookAuthorship`*

The *`Book`* model represents an instance of books, including information about different *editions* of the same book.

Book Views

Book List [`cv.views.CVListView`]

Context object	`{{object_list}}`
Template	`'cv/lists/book_list.html'`
URL	`'books/'`
MIME type	`text/html`

The book list view produces a page with a list of the author's books. This may be useful for profiling an authors' books with, for example, summaries and blurbs. This can be accomplished through the use of custom templates. The default template produces a list of books using the same section formatting as the listing in the book section of the complete CV.

The page renders an instance of the `cv.views.CVListView` view with the named parameter `model_name` set to `'book'`. The view returns `{{object_list}}` in the context with four objects on its dot path:

`total_books` Integer of total number of books from all three managers:

`book_published_list` queryset of all published books (uses the `published()` method of the *`PublicationManager`*)

book_revise_list queryset of all books in the revision process (uses the `revise()` method of the *PublicationManager*)

book_inprep_list queryset of all books in preparation for submission (uses the `inprep()` method of the *PublicationManager*)

The URL can be accessed in templates by using the URL template filter with the named URL `section_list` and `model_name` parameter equal to `book`, i.e.:

```
{% url section_list model_name='book' %}
```

Book Detail: `cv.views.CVDetailView`

Context object	`{{book}}`
Template	`'cv/details/book_detail.html'`
URL	`'books/<slug:slug>/'`
MIME type	`text/html`

The book detail view produces a page that represents a single book. This could be used to, for example, create a feature page for a published book. The default view includes the title, abstract, edition information, and links to download the citation information in both RIS and BibTeX formats (described below). The page is rendered as an instance of the `cv.views.CVDetailView` with the named parameters `model_name` set to `'book'` and `slug` set to the book's slug attribute. The view returns the context `{{book}}` that represents the `Book` instance.

The URL can be accessed using the named URL `item_detail` with `model_name` set to `'book'` and `slug` set to the book's slug attribute, i.e.:

```
{% url item_detail model_name='book' slug='slug-from-short-title' %}
```

Book Citation: `cv.views.citation_view()`

Context object	`{{book}}`
Templates	`'cv/citations/book.ris'` `'cv/citations/book.bib'`
URL	`'books/<slug:slug>/citation/<str:format>/'`
MIME types	`application/x-research-info-systems` or `application/x-bibtex`

Returns view to allow citation to be downloaded to citation management software.

The `<str:format>` named parameter should be one of:

'ris' will create downloadable citation using Reference Manager format specification (see http://endnote.com/sites/rm/files/m/direct_export_ris.pdf).

'bib' will create downloadable citation using the BibTeX format specification (see http://www.bibtex.org/Format/)

Book Editions

Django-Vitae allows users to link multiple editions of a book with the *BookEdition* class. This is done through a ForeignKey relationship to the *book*. The *Book* model includes the *get_editions()* method to return all editions associated with the book in reverse chronological order (i.e., newest first).

If an edition has been related to a book, the default templates will use the publication information (publisher, place of publication, ISBN) of the edition instance, not the publication information defined for the book instance.

Custom Methods

The *Book* class has two custom methods related to editions:

add_edition(*dict*)
Creates a new *BookEdition* instance with the referencing the `Book` instance on which the user calls the method.

- `dict`: a dictionary containing field/value pairs for *BookEdition* fields; `edition` must be one of the `dict` keys

get_editions()
Convenience function that returns a `QuerySet` of all the `BookEdition` objects related to the `Book` instance

Chapters

Chapter Model

Model field reference	*cv.models.publications.Chapter*
Authorship set	*cv.models.publications.ChapterAuthorship*
Editorship set	*cv.models.publications.ChapterEditorship*

The *Chapter* model represents an instance of a chapter. In addition to the `authorship` attribute that saves authorship information, the `Chapter` class also has an `editorship` attribute that contains information about editors of the volume in which the chapter appears. The editorship relationship operates the same way as *authorship sets* and include the same fields, except that the `editorship` model does not contain a `student_colleague` field.

Chapter Views

Chapter List [`cv.views.CVListView`]

Context object	`{{chapter_objects}}`
Template	`'cv/lists/chapter_list.html'`
URL	`'chapters/'`
MIME type	`text/html`

The chapter list view produces a page with a list of the author's chapters. The page renders an instance of the `cv.views.CVListView` with the named parameter `model_name` set to `'chapter'`. This view returns the object `{{object_list}}` in the context with four objects on its dot path:

total_chapters Integer of total number of chapters from all three managers:

chapter_published_list queryset of all published chapters (uses the `published()` method of the *PublicationManager*)

chapter_revise_list queryset of all chapters in the revision process (uses the `revise()` method of the *PublicationManager*)

chapter_inprep_list queryset of all chapters in preparation for submission (uses the `inprep()` method of the *PublicationManager*)

The URL can be accessed in templates by using the URL template filter with the named URL `section_list` and `model_name` parameter equal to `chapter`, i.e.:

```
{% url section_list model_name='chapter' %}
```

Chapter Detail: `cv.views.ChapterDetailView`

Context object	`{{chapter}}`
Template	`'cv/details/chapter_detail.html'`
URL	`'chapters/<slug:slug>/'`
MIME type	`text/html`

The chapter detail view produces a page that represents a single chapter. The default template includes the title, the abstract, and links to download the citation in both RIS and BibTeX formats (described below). The page is rendered as an instance of the `cv.views.CVDetailView` view with the named parameters `model_name` set to `'chapter'` and the `slug` set to the value of the chapter's slug field. The view returns the context `{{chapter}}` that represents a the *`Chapter`* instance.

The URL can be accessed using the named URL `item_detail` with with `model_name` set to `article` and `slug` set to the article's slug attribute, i.e.:

```
{% url item_detail model_name='chapter' slug='slug-from-short-title' %}
```

Chapter Citation: `cv.views.book_citation_view()`

Context object	`{{chapter}}`
Templates	`'cv/citations/chapter.ris'` `'cv/citations/chapter.bib'`
URL	`'chapter/<slug:slug>/citation/<str:format>/'`
MIME types	`application/x-research-info-systems` `application/x-bibtex`

Returns view to allow citation to be downloaded to citation management software.

The `<str:format>` named parameter should be one of:

`'ris'` will create downloadable citation using Reference Manager format specification (see http://endnote.com/sites/rm/files/m/direct_export_ris.pdf).

`'bib'` will create downloadable citation using the BibTeX format specification (see http://www.bibtex.org/Format/)

Reports

Report Model

Model field reference	`cv.models.Report`
Authorship set	`cv.models.ReportAuthorship`

The *`Report`* model represents an instance of a report or a publication with a similar format to a report (e.g., policy brief, working paper, etc.)

Report Views

Report List : `cv.views.CVListView`

Context object	`{{report_objects}}`
Template	`'cv/lists/report_list.html'`
URL	`'reports/'`
MIME type	`text/html`

The report list view produces a page with a list of an author's reports. The page is a rendered instance of the `cv.views.CVListView` view with the named parameter `model_name` set to `'report'`. The view returns the object `{{object_list}}` in the context with with four objects on its dot path:

`total_reports` Integer of total number of books from all three managers:

`report_published_list` `QuerySet` of all published books (uses the *published manager <topics-pubs-published-manager>*)

`report_revise_list` queryset of all books in the revision process (uses the *revise manager <topics-pubs-revise-manager>*)

`report_inprep_list` queryset of all books in preparation for submission (uses the *inprep manager <topics-pubs-published-manager>*)

The URL can be accessed in templates by using the URL template filter with the named URL `section_list` and `model_name` parameter equal to `report`, i.e.:

```
{% url section_list model_name='report' %}
```

Report Detail: `cv.views.CVDetailView`

Context object	`{{report}}`
Template	`'cv/details/report_detail.html'`
URL	`'reports/<slug:slug>/'`
MIME type	`text/html`

The report detail view produces a representation of a single report. The page renders an instance of `cv.views.CVDetailView` with the named parameters `model_name` set to `'report'` and the `slug` set to the value of the report's `slug` field. The view returns the context object `{{report}}` that represents a single `Report` instance.

Report Citation: `cv.views.citation_view()`

Context object	`{{report}}`
Templates	`'cv/citations/report.ris'` `'cv/citations/report.bib'`
URL	`'reports/<slug:slug>/citation/<str:format>/'`
MIME types	`application/x-research-info-systems` `application/x-bibtex`

Creates representation of a report as a file that can be downloaded or exported to citation management software.

The `<str:format>` named parameter should be one of:

`'ris'` will create downloadable citation using Reference Manager format specification (see http://endnote.com/sites/rm/files/m/direct_export_ris.pdf).

`'bib'` will create downloadable citation using the BibTeX format specification (see http://www.bibtex.org/Format/)

6.3.3 Other Works

Grants

Talks

To list public presentations on CVs, Django-Vitae uses two models representing two different ideas. A "talk", represented by *`Talk`*, reflects a single idea conveyed with a title. It can optionally also include other other elements related to that talk such as notes and slides. A "presentation", represented by *`Presentation`*, reflects a specific public performance of a talk at a some location and at some time.

This structure allows multiple presentations of the same talk to be logically connected and can prevent multiple listings with the same title, for example, in the "Presentations" section of a C.V.

Talk Model

The *`Talk`* model has three required fields:

- `title`
- `short_title`
- `slug`

The publication set for a given talk can be accessed with the `presentations` attribute of a *`Talk`* instance.

The *`Talk`* class contains a foreign key field, `article_from_talk` that connects a talk to an article. This may be useful to provide a link to the article on a page about the talk to make it clear where visitors can find the publication that resulted.

The *`Talk`* model also contains a convenience method, `get_latest_presenation()` that returns the *`Presentation`* instance of the talk that was most recently performed (using the `presentation_date` field).

Talk Views

Talk List: `TalkListView`

Display a list of all talks given in order of most recent presentation date.

Context object	`{{talk_list}}`
Template	`'cv/lists/talk_list.html'`
URL	`r'^talks/$'`
URL name	`'talk_object_list'`
MIME type	`text/html`

Talk Detail: `TalkDetailView`

Display detailed information for a particular talk.

Context object	`{{talk}}`
Template	`'cv/details/talk_detail.html'`
URL	`r'^talks/(?P<slug>[-\w]+)/$'`
URL name	`'talk_object_detail'`
MIME type	`text/html`

Presentations

The `Presentation` model instances relate to a `Talk` instance through a foreign key. The `Presentation` model has three required fields in addition to the `Talk` foreign key:

- `presentation_date` that represents when this presentation was "performed;" presentations are ordered by presentation date with the most recent presentation first
- `type` represents the form of the presentation; choices are "Invited", "Conference", "Workshop", and "Keynote".
- `event` contains the name of event or venue at which the presentation was given.

Django-Vitae assumes that presentations will be displayed in conjunction with talks and, therefore, not displayed on their own.

Other Writing

Django-Vitae comes with a model to describe writing other than presenting research findings. These can be book reviews, op eds, blog posts, or other types of non-academic writing. The `OtherWriting` class stores instances of these writings.

The `OtherWriting` model has five required fields:

- `title`
- `short_title`
- `slug`
- `date`
- `venue` (e.g., publication where the writing was published)

The `OtherWriting` includes a field `type` that you may use to group different types of writing together on a CV (Django-Vitae does not, however, currently do this by default).

A full reference of fields included in the `OtherWriting` model can be found in the `cv.models.OtherWriting` model reference.

Datasets

Django-Vitae includes a model to describe datasets produced by the author. The `Dataset` class stores instances of these datasets.

The `Dataset` model has three required fields:

- `title`
- `short_title`
- `slug`

The `Dataset` model also includes an `authorship` field that allows for authorships of the `Dataset`. The authorships are related to the `Dataset` through a foreign-key relationship to the `DatasetAuthorship` model. This model works the same way that the *authorship sets* on publications.

A full description of fields can be found in the `Dataset` field reference.

6.4 Shortcuts

- *Table of default publication status codes*

6.5 Settings

6.5.1 CV_PUBLICATION_STATUS_CHOICES

Default:

```
(
      (0,'INPREP',_('In preparation')),
      (1,'WORKING',_('Working paper')),
      (20,'SUBMITTED',_('Submitted')),
      (30,'REVISE',_('Revise')),
      (35,'RESUBMITTED',_('Resubmitted')),
      (40,'CONDACCEPT', _('Conditionally accepted')),
      (50,'FORTHCOMING',_('Forthcoming')),
      (55,'INPRESS', _('In press')),
      (60,'PUBLISHED',_('Published')),
      (99,'RESTING',_('Resting'))
)
```

A list specifying the constants and display values used to create choices for the `status` field of `VitaePublicationModel` proxy class and which publications *Managers* return

Django-Vitae managers. Each option must be composed of three elements:

- an integer setting the constant used by the database to store values
- a string indicating what the constant will be be called; these values will be used to set a constant with the suffix `_STATUS` in the `cv.settings` module.
- value that will be displayed as the choice

Internally, Django-Vitae organizes the type of publication based on the value of the integer used for the choice. The following table shows the ranges used for different publication statuses.

Values >=	and <	Status	Manager
0	10	In preparation	`InprepManager`
10	20	Reserved for user to use as needed	<none>
20	50	In revision	`ReviseManager`
50	90	Published	`PublishedManager`
90		Reserved for user to use as needed	<none>

6.5.2 CV_FILE_TYPES_CHOICES

Default:

```
CV_FILE_TYPES_CHOICES = (
        (10, 'MANUSCRIPT_FILE', _('Manuscript')),
        (20, 'PREPRINT_FILE', _('Preprint')),
        (30, 'DRAFT_FILE', _('Draft')),
```

(continues on next page)

(continued from previous page)

```
        (40, 'SLIDE_FILE', _('Slides')),
        (50, 'CODE_FILE', _('Code')),
        (60, 'TABLE_FILE', _('Table')),
        (70, 'IMAGE_FILE', _('Image')),
        (80, 'SUPPLEMENT_FILE', _('Supplement')),
        (100, 'OTHER_FILE', _('Other'))
)
```

A tuple that contains the values, names, and labels of choices to classify file types for `CVFile`. The `cv.settings` module stores tuple of values and labels of choices in `FILE_TYPES_CHOICES` and a dictionary of names to access choice values in `FILE_TYPES`.

6.5.3 CV_STUDENT_LEVELS_CHOICES

Default:

```
CV_STUDENT_LEVELS_CHOICES =(
        (0,'UNDERGRAD',_('Undergraduate student')),
        (10,'MASTERS',_('Masters student')),
        (20,'DOCTORAL',_('Doctoral student'))
        )
```

A tuple of three-tuples that each contain the value, name, and label to customize the choices related to the level of student. Used for the `cv.models.Student` model for advising and for student collaborations in publication *authorship sets*.

6.5.4 CV_SERVICE_TYPES_CHOICES

Default:

```
CV_SERVICE_TYPES_CHOICES = (
        (10,'DEPARTMENT',_('Department')),
        (20,'SCHOOL', _('School or College')),
        (30,'UNIVERSITY',_('University-wide')),
        (40,'DISCIPLINE',_('Discipline')),
        (50,'COMMUNITY',_('Community')),
        (90,'OTHER',_('Other'))
        )
```

A tuple of three-tuples that each contain the value, name, and label to customize the choices related to the types of service.

6.5.5 CV_KEY_CONTRIBUTORS_LIST

Default: `[]` (Empty list)

A list of e-mails identifying contributors that should be highlighted in the CV.

6.6 Module Reference

6.6.1 `cv.models`

Reference for `cv.models` generated from docstrings.

Base Models

class `cv.models.base.`**`DisplayableModel`**(**args*, ***kwargs*)

Abstract class that includes fields shared by all models.

The abstract class defines three fields common to all models in Django-Vitae. The model is managed by `cv.models.managers.DisplayManager`, which is the default manager for all models that inherit from DisplayableModel

Parameters

- **`display`** (*`BooleanField`*) – (**required**)
- **`extra`** (*`TextField`*) –
- **`files`** (*`GenericRelation`*) –

class `cv.models.base.`**`Collaborator`**(**args*, ***kwargs*)

Representation of collaborator on publications or projects.

Collaborators represent all people listed in entries of a CV that are not the user. Django-Vitae uses the `email` attribute to identify and manage collaborators internally and must, therefore, be unique to each collaborator.

Collaborators are ordered alphabetically by last name by default.

Parameters

- **`id`** (*`AutoField`*) –
- **`first_name`** (*`CharField`*) – (**required**)
- **`last_name`** (*`CharField`*) – (**required**)
- **`email`** (*`EmailField`*) – (**required**)
- **`middle_initial`** (*`CharField`*) –
- **`suffix`** (*`CharField`*) –
- **`institution`** (*`CharField`*) –
- **`website`** (*`URLField`*) –
- **`alternate_email`** (*`EmailField`*) –

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class `cv.models.base.`**`CollaborationModel`**(**args*, ***kwargs*)

Abstract model connecting collaborators to products.

Collaborators are tied to the user through specific collaborations. For example, a paper with two authors–the user and a collaborator–represents one *collaboration* that has unique characteristics such as the order of authorship. A second paper by the same two authors would represent a new collaboration. The abstract collaboration model allows for these connections across a variety of different collaboration types.

Fields:

collaborator : ForeignKey field to the Collaborator model.

print_middle : Should the collaborator's middle initial be inlcuded in the CV entry?

display_order : Integer representing the order in which collaborators are listed.

Parameters

- **collaborator_id** (ForeignKey to *Collaborator*) – (**required**)
- **print_middle** (*BooleanField*) – (**required**), Display author's middle initial?
- **display_order** (*IntegerField*) – (**required**), Order that collaborators should be listed

class cv.models.base.**StudentCollaborationModel**(**args*, ***kwargs*)

Abstract collaboration model to note collaborations with students.

Often advisors wish to highlight collaborations with students on CVs. This abstract class adds a single field that allows the user to indicate whether a collaborator was a student and, if so, the level of the student (e.g., undergrad, masters, doctoral).

Parameters **student_colleague** (*IntegerField*) –

class cv.models.base.**Discipline**(**args*, ***kwargs*)

Model that represents academic discipline.

Some models include a Foreign Key relationship to Discipline to allow instances to be classified by discipline (e.g., to sort CV by discipline in which articles are published).

Parameters

- **id** (*AutoField*) –
- **name** (*CharField*) – (**required**)
- **slug** (*SlugField*) – (**required**)

exception DoesNotExist

exception MultipleObjectsReturned

class cv.models.base.**VitaeModel**(**args*, ***kwargs*)

Reusable model containing basic titling and discipline fields.

Parameters

- **display** (*BooleanField*) – (**required**)
- **extra** (*TextField*) –
- **title** (*CharField*) – (**required**)
- **short_title** (*CharField*) – (**required**)
- **slug** (*SlugField*) – (**required**), Automatically built from short title
- **primary_discipline_id** (ForeignKey to *Discipline*) –
- **other_disciplines** (*ManyToManyField*) –
- **files** (*GenericRelation*) –

class cv.models.base.**VitaePublicationModel**(**args*, ***kwargs*)

Reusable model with fields common to all types of publications.

The model uses `cv.models.managers.PublicationManger` to manage instances of the model. The `PublicationManager` is named `displayable`.

Internally managed fields: `VitaePublicationModel` instances include three fields managed internally related to publication status: `is_published`, `is_inrevision`, and `is_inprep`. The values of each of these boolean fields are set when cleaning the model instance. Django-Vitae also manages an `abstract_html` field internally to save an HTML version of markdown text saved in the `abstract` field.

Custom methods:

`get_next_by_status()` and `get_previous_by_status` mimic Django's built-in methods `get_next_by()` and `get_previous_by` but inlcudes a constraint that the publication status must be the same as that of the current model instance.

`cite()` prints the instance's citation using the CSL format defined in the `CV_CITE_CSL_STYLE` setting.

Parameters

- **`display`** (*`BooleanField`*) – (**required**)
- **`extra`** (*`TextField`*) –
- **`title`** (*`CharField`*) – (**required**)
- **`short_title`** (*`CharField`*) – (**required**)
- **`slug`** (*`SlugField`*) – (**required**), Automatically built from short title
- **`primary_discipline_id`** (ForeignKey to *`Discipline`*) –
- **`abstract`** (*`TextField`*) –
- **`status`** (*`IntegerField`*) – (**required**)
- **`pub_date`** (*`DateField`*) –
- **`submission_date`** (*`DateField`*) –
- **`is_published`** (*`BooleanField`*) – (**required**)
- **`is_inrevision`** (*`BooleanField`*) – (**required**)
- **`is_inprep`** (*`BooleanField`*) – (**required**)
- **`abstract_html`** (*`TextField`*) –
- **`other_disciplines`** (*`ManyToManyField`*) –
- **`files`** (*`GenericRelation`*) –

`save`(**args*, ***kwargs*)
Sets publication status booleans and abstract text in HTML.

`clean`(**args*, ***kwargs*)
Checks ISBN validity.

`get_next_previous_by_status`(*direc*)
Retrieves next or previous instance with same publication status.

`get_primary_files`()
Return queryset of `cv.models.CVFile` objects designated as "primary files" associated with article.

`cite`(*style='html'*, *doi=True*)
Return citation of instance.

The format used for the citation is set using the CV_CITE_CSL_STYLE setting.

class `cv.models.base.`**`Journal`**(**args*, ***kwargs*)
Store object representing journal/periodical in field.

The model contains one internally managed field, `title_no_article`, which stores the name of the title without the leading articles 'A', 'An' or 'The'. The field is used to alphabetize journals by titles without the leading article, per APA style.

The model includes an `issn` field that stores the International Standard Serial Number for the journal. Future versions might require the issn field to prevent duplicate journal entries and to allow automatic updating of journal lists.

Parameters

- **`id`** (*`AutoField`*) –
- **`title`** (*`CharField`*) – (**required**)
- **`abbreviated_title`** (*`CharField`*) – , Abbreviated journal title; use style you wish to display in views
- **`issn`** (*`CharField`*) – , Enter ISSN in format: XXXX-XXXX
- **`website`** (*`URLField`*) –
- **`primary_discipline_id`** (ForeignKey to *`Discipline`*) –
- **`title_no_article`** (*`CharField`*) –
- **`other_disciplines`** (*`ManyToManyField`*) –

`save`(**args*, ***kwargs*)

Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

`exception DoesNotExist`

`exception MultipleObjectsReturned`

Achievement Models

`class` `cv.models.achievements.`**`Award`**(**args*, ***kwargs*)

Award or honor earned.

Parameters

- **`id`** (*`AutoField`*) –
- **`display`** (*`BooleanField`*) – (**required**)
- **`extra`** (*`TextField`*) –
- **`name`** (*`CharField`*) – (**required**)
- **`organization`** (*`CharField`*) – (**required**)
- **`date`** (*`DateField`*) – (**required**)
- **`description`** (*`TextField`*) –
- **`files`** (*`GenericRelation`*) –

`exception DoesNotExist`

`exception MultipleObjectsReturned`

class `cv.models.achievements.`**`Degree`**`(*args, **kwargs)`

Degree earned.

Degrees are sorted in reverse order by `end_date`.

Parameters

- **`id`** (*`AutoField`*) –
- **`display`** (*`BooleanField`*) – (**required**)
- **`extra`** (*`TextField`*) –
- **`degree`** (*`CharField`*) – (**required**)
- **`major`** (*`CharField`*) –
- **`date_earned`** (*`DateField`*) – (**required**)
- **`institution`** (*`CharField`*) – (**required**)
- **`city`** (*`CharField`*) –
- **`state`** (*`CharField`*) –
- **`country`** (*`CharField`*) –
- **`honors`** (*`CharField`*) –
- **`files`** (*`GenericRelation`*) –

exception DoesNotExist

exception MultipleObjectsReturned

class `cv.models.achievements.`**`Position`**`(*args, **kwargs)`

Position of employment or research experience.

Positions are sorted by `end_date`.

In addition to default managers of `DisplayableModel`, `Position` also has a `primarypositions` manager that only returns positions for which `primary_position==True`. This manager can be used, for example, to list positions in the heading of CVs.

Parameters

- **`id`** (*`AutoField`*) –
- **`display`** (*`BooleanField`*) – (**required**)
- **`extra`** (*`TextField`*) –
- **`title`** (*`CharField`*) – (**required**)
- **`start_date`** (*`DateField`*) – (**required**)
- **`end_date`** (*`DateField`*) – , If current, set date to future (by default positions will be ordered by end date
- **`project`** (*`CharField`*) –
- **`department`** (*`CharField`*) –
- **`institution`** (*`CharField`*) – (**required**)
- **`current_position`** (*`BooleanField`*) – (**required**), Are you currently in this position?

- **primary_position** (*BooleanField*) – (**required**), Should this position be displayed as the main position (e.g., on heading of CV)?
- **files** (*GenericRelation*) –

clean()
Ensure start date is before end date.

exception DoesNotExist

exception MultipleObjectsReturned

Publication Models

Defines Django-CV publication models.

class cv.models.publications.**Article**(**args*, ***kwargs*)
Store instance representing an article.

Parameters

- **id** (*AutoField*) –
- **display** (*BooleanField*) – (**required**)
- **extra** (*TextField*) –
- **title** (*CharField*) – (**required**)
- **short_title** (*CharField*) – (**required**)
- **slug** (*SlugField*) – (**required**), Automatically built from short title
- **primary_discipline_id** (ForeignKey to *Discipline*) –
- **abstract** (*TextField*) –
- **status** (*IntegerField*) – (**required**)
- **pub_date** (*DateField*) –
- **submission_date** (*DateField*) –
- **is_published** (*BooleanField*) – (**required**)
- **is_inrevision** (*BooleanField*) – (**required**)
- **is_inprep** (*BooleanField*) – (**required**)
- **abstract_html** (*TextField*) –
- **journal_id** (ForeignKey to *Journal*) –
- **volume** (*CharField*) –
- **issue** (*CharField*) –
- **start_page** (*CharField*) –
- **end_page** (*CharField*) –
- **series** (*CharField*) –
- **number** (*CharField*) –
- **url** (*URLField*) –
- **doi** (*CharField*) –

- **pmcid** (*CharField*) – , PubMed Central reference number (for more info see: https://publicaccess.nih.gov/include-pmcid-citations.htm#Difference)
- **pmid** (*CharField*) – , PubMed Central reference number (for more info see: https://publicaccess.nih.gov/include-pmcid-citations.htm#Difference)
- **other_disciplines** (*ManyToManyField*) –
- **authors** (*ManyToManyField*) – (**required**)
- **grants** (*ManyToManyField*) –
- **files** (*GenericRelation*) –

exception DoesNotExist

exception MultipleObjectsReturned

class cv.models.publications.**ArticleAuthorship**(**args*, ***kwargs*)
Store object relating collaborators to article.

Parameters

- **id** (*AutoField*) –
- **collaborator_id** (ForeignKey to *Collaborator*) – (**required**)
- **print_middle** (*BooleanField*) – (**required**), Display author's middle initial?
- **display_order** (*IntegerField*) – (**required**), Order that collaborators should be listed
- **student_colleague** (*IntegerField*) –
- **article_id** (ForeignKey to *Article*) – (**required**)

exception DoesNotExist

exception MultipleObjectsReturned

class cv.models.publications.**Book**(**args*, ***kwargs*)
Store instance representing a book.

Parameters

- **id** (*AutoField*) –
- **display** (*BooleanField*) – (**required**)
- **extra** (*TextField*) –
- **title** (*CharField*) – (**required**)
- **short_title** (*CharField*) – (**required**)
- **slug** (*SlugField*) – (**required**), Automatically built from short title
- **primary_discipline_id** (ForeignKey to *Discipline*) –
- **abstract** (*TextField*) –
- **status** (*IntegerField*) – (**required**)
- **pub_date** (*DateField*) –
- **submission_date** (*DateField*) –
- **is_published** (*BooleanField*) – (**required**)
- **is_inrevision** (*BooleanField*) – (**required**)

- **is_inprep** (*BooleanField*) – (**required**)
- **abstract_html** (*TextField*) –
- **publisher** (*CharField*) –
- **place** (*CharField*) –
- **volume** (*IntegerField*) –
- **series** (*CharField*) –
- **series_number** (*CharField*) –
- **num_pages** (*IntegerField*) –
- **isbn** (*CharField*) –
- **url** (*URLField*) –
- **other_disciplines** (*ManyToManyField*) –
- **authors** (*ManyToManyField*) – (**required**)
- **grants** (*ManyToManyField*) –
- **files** (*GenericRelation*) –

add_edition(***kwargs*)
Add edition to book.

get_editions()
Return queryset of all editions associated with book.

exception DoesNotExist

exception MultipleObjectsReturned

class cv.models.publications.**BookAuthorship**(**args*, ***kwargs*)
Store authorship object relating collaborators to book.

Parameters

- **id** (*AutoField*) –
- **collaborator_id** (ForeignKey to *Collaborator*) – (**required**)
- **print_middle** (*BooleanField*) – (**required**), Display author's middle initial?
- **display_order** (*IntegerField*) – (**required**), Order that collaborators should be listed
- **student_colleague** (*IntegerField*) –
- **book_id** (ForeignKey to *Book*) – (**required**)

exception DoesNotExist

exception MultipleObjectsReturned

class cv.models.publications.**BookEdition**(**args*, ***kwargs*)
Store edition information of a book.

Parameters

- **id** (*AutoField*) –
- **display** (*BooleanField*) – (**required**)
- **extra** (*TextField*) –

- **book_id** (ForeignKey to *Book*) – (**required**)
- **edition** (*CharField*) – (**required**)
- **pub_date** (*DateField*) –
- **submission_date** (*DateField*) –
- **publisher** (*CharField*) –
- **place** (*CharField*) –
- **num_pages** (*IntegerField*) –
- **isbn** (*CharField*) –
- **files** (*GenericRelation*) –

clean()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

exception DoesNotExist

exception MultipleObjectsReturned

class cv.models.publications.**Chapter**(**args*, ***kwargs*)

Store instance representing book chapter.

Parameters

- **id** (*AutoField*) –
- **display** (*BooleanField*) – (**required**)
- **extra** (*TextField*) –
- **title** (*CharField*) – (**required**)
- **short_title** (*CharField*) – (**required**)
- **slug** (*SlugField*) – (**required**), Automatically built from short title
- **primary_discipline_id** (ForeignKey to *Discipline*) –
- **abstract** (*TextField*) –
- **status** (*IntegerField*) – (**required**)
- **pub_date** (*DateField*) –
- **submission_date** (*DateField*) –
- **is_published** (*BooleanField*) – (**required**)
- **is_inrevision** (*BooleanField*) – (**required**)
- **is_inprep** (*BooleanField*) – (**required**)
- **book_title** (*CharField*) – (**required**)
- **volume** (*CharField*) –
- **volumes** (*CharField*) –
- **edition** (*CharField*) –
- **publisher** (*CharField*) –

- **place** (*CharField*) –
- **series** (*CharField*) –
- **series_number** (*CharField*) –
- **start_page** (*CharField*) –
- **end_page** (*CharField*) –
- **isbn** (*CharField*) –
- **url** (*URLField*) –
- **abstract_html** (*TextField*) –
- **other_disciplines** (*ManyToManyField*) –
- **authors** (*ManyToManyField*) – (**required**)
- **editors** (*ManyToManyField*) – (**required**)
- **grants** (*ManyToManyField*) –
- **files** (*GenericRelation*) –

exception DoesNotExist

exception MultipleObjectsReturned

class cv.models.publications.**ChapterAuthorship**(**args*, ***kwargs*)

Store object relating collaborators to article.

Parameters

- **id** (*AutoField*) –
- **collaborator_id** (ForeignKey to *Collaborator*) – (**required**)
- **print_middle** (*BooleanField*) – (**required**), Display author's middle initial?
- **display_order** (*IntegerField*) – (**required**), Order that collaborators should be listed
- **student_colleague** (*IntegerField*) –
- **chapter_id** (ForeignKey to *Chapter*) – (**required**)

exception DoesNotExist

exception MultipleObjectsReturned

class cv.models.publications.**ChapterEditorship**(**args*, ***kwargs*)

Store object relating editors to chapter.

Parameters

- **id** (*AutoField*) –
- **collaborator_id** (ForeignKey to *Collaborator*) – (**required**)
- **print_middle** (*BooleanField*) – (**required**), Display author's middle initial?
- **display_order** (*IntegerField*) – (**required**), Order that collaborators should be listed
- **chapter_id** (ForeignKey to *Chapter*) – (**required**)

exception DoesNotExist

exception MultipleObjectsReturned

class cv.models.publications.**Report**(**args*, ***kwargs*)

Store instance representing reports.

Parameters

- **id** (*AutoField*) –
- **display** (*BooleanField*) – (**required**)
- **extra** (*TextField*) –
- **title** (*CharField*) – (**required**)
- **short_title** (*CharField*) – (**required**)
- **slug** (*SlugField*) – (**required**), Automatically built from short title
- **primary_discipline_id** (ForeignKey to *Discipline*) –
- **abstract** (*TextField*) –
- **status** (*IntegerField*) – (**required**)
- **pub_date** (*DateField*) –
- **submission_date** (*DateField*) –
- **is_published** (*BooleanField*) – (**required**)
- **is_inrevision** (*BooleanField*) – (**required**)
- **is_inprep** (*BooleanField*) – (**required**)
- **report_number** (*CharField*) –
- **report_type** (*CharField*) –
- **series_title** (*CharField*) –
- **place** (*CharField*) –
- **institution** (*CharField*) –
- **pages** (*CharField*) –
- **url** (*URLField*) –
- **doi** (*CharField*) –
- **abstract_html** (*TextField*) –
- **other_disciplines** (*ManyToManyField*) –
- **authors** (*ManyToManyField*) – (**required**)
- **grants** (*ManyToManyField*) –
- **files** (*GenericRelation*) –

exception DoesNotExist

exception MultipleObjectsReturned

class cv.models.publications.**ReportAuthorship**(**args*, ***kwargs*)

Store object relating collaborators to report.

Parameters

- **id** (*AutoField*) –

- **collaborator_id** (ForeignKey to `Collaborator`) – (**required**)
- **print_middle** (*BooleanField*) – (**required**), Display author's middle initial?
- **display_order** (*IntegerField*) – (**required**), Order that collaborators should be listed
- **student_colleague** (*IntegerField*) –
- **report_id** (ForeignKey to `Report`) – (**required**)

exception DoesNotExist

exception MultipleObjectsReturned

Works Models

class cv.models.works.**Grant** (**args*, ***kwargs*)

Create instance of funded grant.

Parameters

- **id** (*AutoField*) –
- **display** (*BooleanField*) – (**required**)
- **extra** (*TextField*) –
- **title** (*CharField*) – (**required**)
- **short_title** (*CharField*) – (**required**)
- **slug** (*SlugField*) – (**required**), Automatically built from short title
- **primary_discipline_id** (ForeignKey to `Discipline`) –
- **source** (*IntegerField*) – (**required**), Internal/external source of funding
- **agency** (*CharField*) –
- **agency_acronym** (*CharField*) –
- **division** (*CharField*) –
- **division_acronym** (*CharField*) –
- **grant_number** (*CharField*) –
- **amount** (*IntegerField*) – (**required**)
- **start_date** (*DateField*) – (**required**)
- **end_date** (*DateField*) –
- **role** (*CharField*) –
- **is_current** (*BooleanField*) – (**required**)
- **abstract** (*TextField*) –
- **abstract_html** (*TextField*) –
- **other_disciplines** (*ManyToManyField*) –
- **collaborators** (*ManyToManyField*) – (**required**)
- **files** (*GenericRelation*) –

save (*force_insert=False*, *force_update=False*, **args*, ***kwargs*)

Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

exception DoesNotExist

exception MultipleObjectsReturned

class `cv.models.works.`**GrantCollaboration** (**args*, ***kwargs*)

Store object relating collaborators to grant.

Parameters

- **id** (*AutoField*) –
- **collaborator_id** (ForeignKey to *Collaborator*) – (**required**)
- **print_middle** (*BooleanField*) – (**required**), Display author's middle initial?
- **display_order** (*IntegerField*) – (**required**), Order that collaborators should be listed
- **grant_id** (ForeignKey to *Grant*) – (**required**)
- **is_pi** (*BooleanField*) – (**required**)
- **role** (*CharField*) –

exception DoesNotExist

exception MultipleObjectsReturned

class `cv.models.works.`**Talk** (**args*, ***kwargs*)

Store object representing a talk.

Parameters

- **id** (*AutoField*) –
- **display** (*BooleanField*) – (**required**)
- **extra** (*TextField*) –
- **title** (*CharField*) – (**required**)
- **short_title** (*CharField*) – (**required**)
- **slug** (*SlugField*) – (**required**), Automatically built from short title
- **primary_discipline_id** (ForeignKey to *Discipline*) –
- **abstract** (*TextField*) –
- **abstract_html** (*TextField*) –
- **latest_presentation_date** (*DateField*) –
- **created** (*DateField*) –
- **modified** (*DateField*) –
- **other_disciplines** (*ManyToManyField*) –
- **collaborator** (*ManyToManyField*) –
- **grants** (*ManyToManyField*) –
- **files** (*GenericRelation*) –

save (*force_insert=False*, *force_update=False*, **args*, ***kwargs*)
Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

exception DoesNotExist

exception MultipleObjectsReturned

class `cv.models.works.`**Presentation** (**args*, ***kwargs*)
Create an instance in which a talk was given.

This model creates separate objects for each time the same talk was given.

Parameters

- **id** (*AutoField*) –
- **talk_id** (ForeignKey to *Talk*) – (**required**)
- **presentation_date** (*DateField*) – (**required**)
- **type** (*IntegerField*) – (**required**)
- **event** (*CharField*) – (**required**)
- **event_acronym** (*CharField*) –
- **city** (*CharField*) –
- **state** (*CharField*) –
- **country** (*CharField*) –

save (**args*, ***kwargs*)
Save latest presentation date in related talk if instance is later than current latest presentation date.

exception DoesNotExist

exception MultipleObjectsReturned

class `cv.models.works.`**OtherWriting** (**args*, ***kwargs*)
Create an instance of writing in venues other than traditional scholarly venues.

Default ordering by `type` and then `date` in descending order.

Parameters

- **id** (*AutoField*) –
- **display** (*BooleanField*) – (**required**)
- **extra** (*TextField*) –
- **title** (*CharField*) – (**required**)
- **short_title** (*CharField*) – (**required**)
- **slug** (*SlugField*) – (**required**), Automatically built from short title
- **primary_discipline_id** (ForeignKey to *Discipline*) –
- **type** (*CharField*) – , Genre of writing (e.g., 'book review','op ed', 'blog post') that can be used for grouping contributions by type.
- **abstract** (*TextField*) –
- **venue** (*CharField*) – (**required**)

- **date** (*DateField*) – (**required**)
- **pages** (*CharField*) –
- **url** (*URLField*) –
- **place** (*CharField*) –
- **volume** (*CharField*) –
- **issue** (*CharField*) –
- **abstract_html** (*TextField*) –
- **other_disciplines** (*ManyToManyField*) –
- **files** (*GenericRelation*) –

save (*force_insert=False*, *force_update=False*, **args*, ***kwargs*)
Saves abstract in html format.

exception DoesNotExist

exception MultipleObjectsReturned

class cv.models.works.**Dataset** (**args*, ***kwargs*)
Stores instance representing a dataset.

Parameters

- **id** (*AutoField*) –
- **display** (*BooleanField*) – (**required**)
- **extra** (*TextField*) –
- **title** (*CharField*) – (**required**)
- **short_title** (*CharField*) – (**required**)
- **slug** (*SlugField*) – (**required**), Automatically built from short title
- **primary_discipline_id** (ForeignKey to *Discipline*) –
- **pub_date** (*DateField*) –
- **version_number** (*CharField*) –
- **format** (*CharField*) – , Form of data (e.g., 'Datafile and Codebook' or 'Datafile')
- **producer** (*CharField*) –
- **producer_place** (*CharField*) –
- **distributor** (*CharField*) –
- **distributor_place** (*CharField*) –
- **retrieval_url** (*URLField*) – , Used for URL linked to dataset
- **available_from_url** (*URLField*) – , Used to link to a download page
- **doi** (*CharField*) –
- **other_disciplines** (*ManyToManyField*) –
- **authors** (*ManyToManyField*) – (**required**)
- **files** (*GenericRelation*) –

get_absolute_url()
“Returns reverse URL for an instance of a dataset.

exception DoesNotExist

exception MultipleObjectsReturned

class cv.models.works.**DatasetAuthorship**(**args*, ***kwargs*)
Store object relating creators of dataset to a dataset instance.

Parameters

- **id** (*AutoField*) –
- **collaborator_id** (ForeignKey to *Collaborator*) – (**required**)
- **print_middle** (*BooleanField*) – (**required**), Display author’s middle initial?
- **display_order** (*IntegerField*) – (**required**), Order that collaborators should be listed
- **student_colleague** (*IntegerField*) –
- **dataset_id** (ForeignKey to *Dataset*) – (**required**)

exception DoesNotExist

exception MultipleObjectsReturned

6.6.2 cv.models.managers

class cv.models.managers.**DisplayManager**
Returns displayable objects from models.

get_queryset()
Return objects for which field `display` has been set to `True`.

class cv.models.managers.**PublicationManager**
Class to manage publications.

This class subclasses `DisplayManager` and includes the default queryset of all displayable objects. In addition, it provides three methods: `published`, `revise`, and `inprep` to return querysets of publications at stages in the publication process.

published()
Return queryset of articles accepted for publication or published.

revise()
Return queryset of articles in revision process.

inprep()
Return queryset of articles being prepared for submission.

class cv.models.managers.**GrantManager**
Class to manage grants.

This class subclasses `DisplayManager` and includes the default queryset of all displayable objects. In addition, it provides two methods: `internal_grants` and `external_grants` for different grant sources.

class cv.models.managers.**ServiceManager**
Class to manage service work.

This class subclasses `DisplayManager` and includes the default queryset of all displayable objects. In addition, it provides three methods: `department_services`, `university services`, and `discipline_services` for service work to different institutions.

class `cv.models.managers.`**`PrimaryPositionManager`**

Manages positions used in heading of CV.

Returns a queryset of positions in which `primary_position` has been set to `True`.

`get_queryset``()`

Return positions user indicated as 'primary' positions.

6.6.3 `cv.views`

Reference for `cv.views` generated from docstrings.

Python Module Index

m

`cv.models.achievements`, 31
`cv.models.base`, 28
`cv.models.managers`, 43
`cv.models.publications`, 33
`cv.models.works`, 39

Index

A

add_edition(), 21
add_edition() (*cv.models.publications.Book method*), 35
Article (*class in cv.models.publications*), 33
Article.DoesNotExist, 34
Article.MultipleObjectsReturned, 34
ArticleAuthorship (*class in cv.models.publications*), 34
ArticleAuthorship.DoesNotExist, 34
ArticleAuthorship.MultipleObjectsReturned, 34
Award (*class in cv.models.achievements*), 31
Award.DoesNotExist, 31
Award.MultipleObjectsReturned, 31

B

Book (*class in cv.models.publications*), 34
Book.DoesNotExist, 35
Book.MultipleObjectsReturned, 35
BookAuthorship (*class in cv.models.publications*), 35
BookAuthorship.DoesNotExist, 35
BookAuthorship.MultipleObjectsReturned, 35
BookEdition (*class in cv.models.publications*), 35
BookEdition.DoesNotExist, 36
BookEdition.MultipleObjectsReturned, 36

C

Chapter (*class in cv.models.publications*), 36
Chapter.DoesNotExist, 37
Chapter.MultipleObjectsReturned, 37
ChapterAuthorship (*class in cv.models.publications*), 37
ChapterAuthorship.DoesNotExist, 37
ChapterAuthorship.MultipleObjectsReturned, 37
ChapterEditorship (*class in cv.models.publications*), 37
ChapterEditorship.DoesNotExist, 37
ChapterEditorship.MultipleObjectsReturned, 37
cite() (*cv.models.base.VitaePublicationModel method*), 30
clean() (*cv.models.achievements.Position method*), 33
clean() (*cv.models.base.VitaePublicationModel method*), 30
clean() (*cv.models.publications.BookEdition method*), 36
CollaborationModel (*class in cv.models.base*), 28
Collaborator (*class in cv.models.base*), 28
Collaborator.DoesNotExist, 28
Collaborator.MultipleObjectsReturned, 28
cv.models.achievements (*module*), 31
cv.models.base (*module*), 28
cv.models.managers (*module*), 43
cv.models.publications (*module*), 33
cv.models.works (*module*), 39
CV_FILE_TYPES_CHOICES
  setting, 26
CV_KEY_CONTRIBUTORS_LIST
  setting, 27
CV_PUBLICATION_STATUS_CHOICES
  setting, 26
CV_SERVICE_TYPES_CHOICES
  setting, 27
CV_STUDENT_LEVELS_CHOICES
  setting, 27

D

Dataset (*class in cv.models.works*), 42
Dataset.DoesNotExist, 43
Dataset.MultipleObjectsReturned, 43
DatasetAuthorship (*class in cv.models.works*), 43
DatasetAuthorship.DoesNotExist, 43
DatasetAuthorship.MultipleObjectsReturned, 43
Degree (*class in cv.models.achievements*), 31

Degree.DoesNotExist, 32
Degree.MultipleObjectsReturned, 32
Discipline (*class in cv.models.base*), 29
Discipline.DoesNotExist, 29
Discipline.MultipleObjectsReturned, 29
DisplayableModel (*class in cv.models.base*), 28
DisplayManager (*class in cv.models.managers*), 43

G

get_absolute_url() (*cv.models.works.Dataset method*), 42
get_editions(), 21
get_editions() (*cv.models.publications.Book method*), 35
get_next_previous_by_status() (*cv.models.base.VitaePublicationModel method*), 30
get_primary_files() (*cv.models.base.VitaePublicationModel method*), 30
get_queryset() (*cv.models.managers.DisplayManager method*), 43
get_queryset() (*cv.models.managers.PrimaryPositionManager method*), 44
Grant (*class in cv.models.works*), 39
Grant.DoesNotExist, 40
Grant.MultipleObjectsReturned, 40
GrantCollaboration (*class in cv.models.works*), 40
GrantCollaboration.DoesNotExist, 40
GrantCollaboration.MultipleObjectsReturned, 40
GrantManager (*class in cv.models.managers*), 43

I

inprep() (*cv.models.managers.PublicationManager method*), 43

J

Journal (*class in cv.models.base*), 30
Journal.DoesNotExist, 31
Journal.MultipleObjectsReturned, 31

O

OtherWriting (*class in cv.models.works*), 41
OtherWriting.DoesNotExist, 42
OtherWriting.MultipleObjectsReturned, 42

P

Position (*class in cv.models.achievements*), 32
Position.DoesNotExist, 33
Position.MultipleObjectsReturned, 33
Presentation (*class in cv.models.works*), 41
Presentation.DoesNotExist, 41
Presentation.MultipleObjectsReturned, 41
PrimaryPositionManager (*class in cv.models.managers*), 44
PublicationManager (*class in cv.models.managers*), 43
published() (*cv.models.managers.PublicationManager method*), 43

R

Report (*class in cv.models.publications*), 38
Report.DoesNotExist, 38
Report.MultipleObjectsReturned, 38
ReportAuthorship (*class in cv.models.publications*), 38
ReportAuthorship.DoesNotExist, 39
ReportAuthorship.MultipleObjectsReturned, 39
revise() (*cv.models.managers.PublicationManager method*), 43

S

save() (*cv.models.base.Journal method*), 31
save() (*cv.models.base.VitaePublicationModel method*), 30
save() (*cv.models.works.Grant method*), 39
save() (*cv.models.works.OtherWriting method*), 42
save() (*cv.models.works.Presentation method*), 41
save() (*cv.models.works.Talk method*), 40
ServiceManager (*class in cv.models.managers*), 43
setting
    CV_FILE_TYPES_CHOICES, 26
    CV_KEY_CONTRIBUTORS_LIST, 27
    CV_PUBLICATION_STATUS_CHOICES, 26
    CV_SERVICE_TYPES_CHOICES, 27
    CV_STUDENT_LEVELS_CHOICES, 27
StudentCollaborationModel (*class in cv.models.base*), 29

T

Talk (*class in cv.models.works*), 40
Talk.DoesNotExist, 41
Talk.MultipleObjectsReturned, 41

V

VitaeModel (*class in cv.models.base*), 29
VitaePublicationModel (*class in cv.models.base*), 29