

---

# **CCE Toolkit Documentation**

***Release 1.0.0-beta***

**CCE-IT Devs**

**Dec 20, 2018**



---

## Contents

---

<b>1</b>	<b>Toolkit Views, Models and Forms</b>	<b>1</b>
1.1	<code>toolkit.views</code> – Toolkit views	1
1.2	<code>toolkit.models</code> – Toolkit models	9
1.3	<code>toolkit.forms</code> – Toolkit forms	10
1.4	<code>toolkit.templatetags</code> – Toolkit template tags	11
<b>2</b>	<b>Toolkit Helpers</b>	<b>13</b>
2.1	<code>toolkit.helpers.admin</code> – django-admin helpers	13
2.2	<code>toolkit.helpers.reports</code> – custom reports helpers	14
2.3	<code>toolkit.helpers.utils</code> – miscellaneous helpers	18
2.4	<code>toolkit.helpers.bdd</code> – BDD helpers	20
<b>3</b>	<b>Toolkit Apps</b>	<b>25</b>
3.1	<code>toolkit.apps.breadcrumbs</code> – CCE breadcrumbs	25
3.2	<code>toolkit.apps.toolkit_activity_log</code> – CCE Activity Log	25
3.3	<code>toolkit.apps.fabfile</code> – Fabric recipes	25
<b>4</b>	<b>License</b>	<b>27</b>
4.1	License	27
4.2	Contact	27
<b>5</b>	<b>Indices and tables</b>	<b>29</b>
	<b>Python Module Index</b>	<b>31</b>



## 1.1 toolkit.views – Toolkit views

**class** `toolkit.views.views.CCECreateView` (\*\*kwargs)

This view includes all the mixins required in all CreateViews.

Usage:

```
1 class PollCreateView(CCECreateView):
2     model = Poll
3     form_class = PollCreateForm
4     page_title = "Create a poll"
5     sidebar_group = ['polls', ]
6     success_message = "Poll added successfully."
```

Advanced Usage:

```
1 class PollCreateView(CCECreateView):
2     model = Poll
3     form_class = PollCreateForm
4     template_name = "polls/add.html"
5     page_title = "Create a poll"
6     sidebar_group = ['polls', ]
7     success_message = "Poll added successfully."
8
9     def context_menu_items(self):
10         items = super(PollListView, self).context_menu_items()
11         items.append(
12             # label, reversed url, icon class, sidebar_group
13             (
14                 "Link to something else you want",
15                 reverse('link_to_something_else'),
16                 "glyphicon glyphicon-fire",
17                 "something_else",
```

(continues on next page)

(continued from previous page)

```

18         )
19     )
20     return items

```

**class** toolkit.views.views.CCECreateWithInlinesView(\*\*kwargs)

This view includes all the mixins required in all CreateWithInlinesViews.

**class** toolkit.views.views.CCEDeleteView(\*\*kwargs)

This view includes all the mixins required in all DeleteViews.

**class** toolkit.views.views.CCEDetailView(\*\*kwargs)

This view includes all the mixins required in all DetailViews.

#### Usage:

```

1  class PollDetailView(CCEDetailView):
2      model = Poll
3      page_title = "Poll Details"
4      sidebar_group = ['polls']
5      detail_fields = [
6          ('Name', 'name'),
7          ('Active', 'active'),
8          ('Description', 'description'),
9          ('Choices', lambda poll: some_function(poll)),
10     ]
11     show_context_menu = True

```

#### Advanced Usage:

```

1  class PollDetailView(CCEDetailView):
2      model = Poll
3      page_title = "Poll Details"
4      sidebar_group = ['polls']
5      detail_fields = [
6          ('Name', 'name'),
7          ('Active', 'active'),
8          ('Description', 'description'),
9          ('Choices', lambda poll: some_function(poll)),
10     ]
11     show_context_menu = True
12
13     def context_menu_items(self):
14         items = super(PollDetailView, self).context_menu_items()
15         items.append(
16             # label, reversed url, icon class, sidebar_group
17             (
18                 "Link to something else you want",
19                 reverse('link_to_something_else'),
20                 "glyphicon glyphicon-fire",
21                 "something_else",
22             )
23         )
24     return items

```

**class** toolkit.views.views.CCEFormView(\*\*kwargs)

This view includes the mixins required for all FormViews.

**class** toolkit.views.views.CCEListView(\*\*kwargs)

This view includes all the mixins required in all ListViews.

#### Usage:

```

1  class PollListView(CCEListView):
2      model = Poll
3      paginate_by = 10
4      page_title = "Browse Polls"
5      sidebar_group = ['polls', ]
6      columns = [
7          ('Name', 'name'),
8          ('Active', 'active'),
9      ]
10     show_context_menu = True

```

#### Advanced Usage:

```

1  class PollListView(CCEListView):
2      model = Poll
3      paginate_by = 10
4      template_name = "polls/list.html"
5      ordering = ['-created_at']
6      page_title = "Browse Polls"
7      sidebar_group = ['polls', ]
8      # Column widths should add up to 12
9      columns = [
10         ('Name', 'name', 4),
11         ('Active', 'active', 5),
12     ]
13     actions_column_width = 3
14     show_context_menu = True
15     show_add_button = True
16     popover_rows = [
17         ('Description', 'description'),
18         ('Choices', lambda poll: some_function(poll)),
19     ]
20
21     def context_menu_items(self):
22         items = super(PollListView, self).context_menu_items()
23         items.append(
24             # label, reversed url, icon class, sidebar_group
25             (
26                 "Edit All Polls at Once",
27                 reverse('edit_all_polls'),
28                 "glyphicon glyphicon-pencil",
29                 "edit_all_polls",
30             )
31         )
32         return items
33
34     def render_buttons(self, user, obj, **kwargs):
35         button_list = super(PollListView, self).render_buttons(user, obj,
↵**kwargs)
36         button_list.extend([
37             self.render_button(
38                 url_name='edit_poll_permissions',
39                 pk=obj.pk,
40                 icon_classes='fa fa-lock',

```

(continues on next page)

(continued from previous page)

```

41         ),
42         self.render_button(
43             btn_class='warning',
44             label='Button text',
45             icon_classes='glyphicon glyphicon-fire',
46             url=reverse('some_url_name_no_pk_required'),
47             condensed=False,
48         ),
49     ])
50     return button_list

```

**get\_queryset()**

Orders the queryset based on the order\_by query param in the url

**class** toolkit.views.views.CCEModelFormSetView(\*\*kwargs)

This view includes all the mixins required in all ModelFormSetViews.

**class** toolkit.views.views.CCEObjectRedirectView(\*\*kwargs)

This view includes all the mixins required in all RedirectViews.

**class** toolkit.views.views.CCERedirectView(\*\*kwargs)

This view includes all the mixins required in all RedirectViews.

**class** toolkit.views.views.CCESearchView(\*\*kwargs)

ListView variant that filters the queryset on search parameters.

The field 'search\_form\_class' must be defined as a subclass of SearchForm in inheriting classes.

The field 'allow\_empty' (inherited from MultipleObjectMixin) is ignored, since this view must allow an empty object\_list.

**Usage:**

```

1  class PollListView(CCESearchView):
2      model = Poll
3      paginate_by = 10
4      page_title = "Browse Polls"
5      sidebar_group = ['polls', ]
6      search_form_class = PollSimpleSearchForm
7      advanced_search_form_class = PollAdvancedSearchForm
8      columns = [
9          ('Name', 'name'),
10         ('Active', 'active'),
11     ]
12     show_context_menu = True

```

**Advanced Usage:**

```

1  class PollListView(CCESearchView):
2      model = Poll
3      paginate_by = 10
4      template_name = "polls/list.html"
5      ordering = ['-created_at']
6      page_title = "Browse Polls"
7      sidebar_group = ['polls', ]
8      search_form_class = PollSimpleSearchForm
9      advanced_search_form_class = PollAdvancedSearchForm
10     # Column widths should add up to 12

```

(continues on next page)



(continued from previous page)

```

11     columns = [
12         ('Name', 'name', 4),
13         ('Active', 'active', 5),
14     ]
15     actions_column_width = 3
16     show_context_menu = True
17     show_add_button = True
18     popover_rows = [
19         ('Description', 'description'),
20         ('Choices', lambda poll: some_function(poll)),
21     ]
22
23     def context_menu_items(self):
24         items = super(PollListView, self).context_menu_items()
25         items.append(
26             # label, reversed url, icon class, sidebar_group
27             (
28                 "Edit All Polls at Once",
29                 reverse('edit_all_polls'),
30                 "glyphicon glyphicon-pencil",
31                 "edit_all_polls",
32             )
33         )
34         return items
35
36     def render_buttons(self, user, obj, **kwargs):
37         button_list = super(PollListView, self).render_buttons(user, obj,
38 ↪ **kwargs)
39         button_list.extend([
40             self.render_button(
41                 url_name='edit_poll_permissions',
42                 pk=obj.pk,
43                 icon_classes='fa fa-lock',
44             ),
45             self.render_button(
46                 btn_class='warning',
47                 label='Button text',
48                 icon_classes='glyphicon glyphicon-fire',
49                 url=reverse('some_url_name_no_pk_required'),
50                 condensed=False,
51             ),
52         ])
53         return button_list

```

**get\_context\_data** (\*\*kwargs)

Puts things into the context that the template needs: - the message to display when there are no results in the list - the context menu template name (calculated from the model name) - the table of data from generate\_list\_view\_table

**get\_queryset** ()

Filters the default queryset based on self.search\_form.search.

Requires self.search\_form to be set before this function is called, probably in self.get.

If self.search\_form is invalid, returns an empty list.

**class** toolkit.views.views.CCETemplateView (\*\*kwargs)

This view includes all the mixins required in all TemplateViews.

**class** toolkit.views.views.CCEUpdateView(\*\*kwargs)

This view includes all the mixins required in all UpdateViews.

**Usage:**

```
1 class PollUpdateView(CCECreateView):
2     model = Poll
3     form_class = PollUpdateForm
4     page_title = "Edit Poll"
5     sidebar_group = ['polls', ]
6     success_message = "Poll saved successfully."
```

**Advanced Usage:**

```
1 class PollUpdateView(CCECreateView):
2     model = Poll
3     form_class = PollUpdateForm
4     template_name = "polls/edit.html"
5     page_title = "Edit Poll"
6     sidebar_group = ['polls', ]
7     success_message = "Poll saved successfully."
8
9     def context_menu_items(self):
10         items = super(PollUpdateView, self).context_menu_items()
11         items.append(
12             # label, reversed url, icon class, sidebar_group
13             (
14                 "Link to something else you want",
15                 reverse('link_to_something_else'),
16                 "glyphicon glyphicon-fire",
17                 "something_else",
18             )
19         )
20         return items
```

**class** toolkit.views.views.CCEUpdateWithInlinesView(\*\*kwargs)

This view includes all the mixins required in all UpdateWithInlinesViews.

**class** toolkit.views.views.ReportDownloadDetailView(\*\*kwargs)

Variant of CCEDetailView that downloads the object as an xls or pdf report.

**class** toolkit.views.views.ReportDownloadSearchView(\*\*kwargs)

Variant of CCESearchView that downloads the search results as an xls or pdf report.

Raises Http404 if the GET parameters do not form a valid search query. If no query string is specified, gives a report of all objects returned by get\_queryset.

**The following fields must be defined on inheriting classes:**

- model or queryset (per BaseListView)
- search\_form\_class (per SearchView)
- report\_function (see ReportView)

**class** toolkit.views.mixins.AbstractedDetailMixin

A Django DetailView mixin used to generate a list of label-value pairs and pass it to the template

**Note** Mixin will also set the default template\_name of the view to generic\_detail.html template

**get\_detail\_fields()**

Override this method to make the details shown dynamic.

**get\_details()**

How self.fields should be formatted: - The first item in each tuple should be the label. - The second item should be EITHER a dotted path from this object to what goes on the page, OR a function that takes exactly one argument. - The third item is an optional extra parameter. - - If the thing passed is a related manager, this is an optional dotted path to apply to each object in the manager.

Example:

```

1 fields = [
2     ('Username', 'user.username'),
3     ('Passport file', 'passport'),
4     ('Zip codes', 'address_set', 'zip_code'),
5     ('Active', 'is_active'),
6     ('Joined date', 'joined_date'),
7     ('Type', lambda obj: type(obj)),
8 ]

```

**Returns** OrderedDict of {label: (value, param)} that gets dropped into the template.

**class toolkit.views.mixins.AbstractedListMixin**

Assumes that it's mixed in with a ListView that has self.model.

**generate\_list\_view\_table** (*columns, data*)

Generates a data structure for use in the generic list template. The “columns” parameter is a list of 2-tuples or 3-tuples or 4 tuples. These contain - a column title - EITHER a function that takes one parameter (an object from the list) and returns a cell of data OR a dot-separated string of attributes, and - an optional column width number that will go into a Bootstrap class. - dot-separated string of attributes with underscore replacing the dots to be used in queryset ordering

All of these values will be coerced to strings in the template. So the function can return an object, and its string representation will be used. The column width can be a string or an integer.

**Example:**

```
columns = [ ("Column title", lambda obj: obj.method_name(CONSTANT)), ("Other title", previously_defined_function, '3'), ("Third", 'dotted.attribute', 2, 'dotted.attribute'), ]
```

The “data” parameter should be an iterable. This method returns a data structure of the following form (using the above example as input):

```
[ [("Column title", "", "", ""), ("Other title", '3', ''), ("Third", 2, '', 'dotted.attribute')], [data[0].method_name(CONSTANT), previously_defined_function(data[0]), data[0].dotted.attribute], [data[1].method_name(CONSTANT), previously_defined_function(data[1]), data[1].dotted.attribute], # etc. ]
```

**get\_context\_data** (*\*\*kwargs*)

Puts things into the context that the template needs: - the message to display when there are no results in the list - the context menu template name (calculated from the model name) - the table of data from generate\_list\_view\_table

**render\_button** (*pk=None, btn\_class="", url\_name="", label="", icon\_classes="", button\_text="", url="", condensed=True*)

Takes of boatload of parameters and returns the HTML for a nice Bootstrap button-link. If the URL lookup fails, no button is returned.

Must have EITHER pk and url\_name OR url.

**render\_buttons** (*user, obj, view\_perm\_func=None, edit\_perm\_func=None, delete\_perm\_func=None*)

This method provides default buttons for an object in a ListView: View, Edit and Delete. The default

permission functions are `can_view`, `can_update` and `can_delete`. We can depend on all three of these permissions methods being there because of `CCEAuditModel`. Other permission functions can be passed, though this is unlikely to happen under the current structure. If a URL is not found, or if the user doesn't have permission to do that thing, the button is not rendered.

**class** `toolkit.views.mixins.AppendURLObjectsMixin`

**this mixin is used to get and append objects, who's pks are passed in view kwargs** making the objects available to all the methods in your view as well your template.

**To use this mixin, you need to set the `append_objects` property with a list** of tuples each containing 3 values

`append_object` format: (Model, context\_variable\_name, field\_lookup) example 1: `append_objects = [(Course, 'course', 'course_pk'), (Student, 'student', 'student_pk')]` in this example, the mixin will try to get the Course and Student objects based on `course_pk` and `student_pk` which are passed through the view url. This mixin assumes that the value being passed by `field_lookup` contains the pk of the object you're trying to fetch. The fetched objects will be appended to context data with variables named after the second value in the triple ('course' or 'student')

example 2: `append_objects = [(Course, 'course', 'course_pk'), (Student, 'student', {'student_id': 'student_pk'})]` In this example, we pass a dict in the `field_lookup` parameter which represents a mapping of model field lookup method and values that will be passed in the url.

the mixin will raise 404 exception if any object is not found

**class** `toolkit.views.mixins.ClassPermissionsMixin`

Use this mixin when table-level `cce_permissions` are required; i.e. `CreateView`, etc.

This class should implement three methods:

`_check_resource()` should verify that the resource we wish to protect (object, model, etc.) exists

`_check_perms_keys()` should verify that the permission functions exist

`_check_permissions()` should actually make the call to the defined `cce_permissions`

**class** `toolkit.views.mixins.FileDownloadMixin`

View mixin to make that view return a file from a model field on GET. Views using this mixin must implement the method `get_file_field_to_download` Note: This is for serving a file as a response and is no longer recommended

**class** `toolkit.views.mixins.ObjectPermissionsMixin`

Use this mixin when `get_object()` is called in your view; i.e. `DetailView`, `UpdateView`, etc.

This class should implement three methods:

`_check_resource()` should verify that the resource we wish to protect (object, model, etc.) exists

`_check_perms_keys()` should verify that the permission functions exist

`_check_permissions()` should actually make the call to the defined `cce_permissions`

**class** `toolkit.views.mixins.PermissionsRequiredMixin`

Inspired by `django-braces`: <https://github.com/brack3t/django-braces>

Requires `django-rulez` <https://github.com/chrisglass/django-rulez> TODO: Does this actually require `django-rulez` anymore?

Parent class of view mixins which allow you to specify a list of rules for access to a resource (model or instance of a model, currently) via `django-rulez` by http verb. Each permission that is listed will be required (logical AND for multiple `cce_permissions`).

A 403 will be raised if any of the checks fails.

Simply create a dictionary with http verbs as keys, and each of their values should be a list of functions (as strings) in ‘function\_name’ format. Each permission should be a function in the same class as the model or model instance defined on the view that inherits one of the children mixins.

Example Usage on an UpdateView subclass:

```
cce_permissions = { "get": ["add_object"] "post": ["change_object", "delete_object"] }
```

```
class toolkit.views.mixins.SuccessMessageMixin
```

Use when you’d like a success message added to your Create or Update response.

**Assumptions:**

- Class has method form\_valid()

**Limitations:**

- Class cannot override form\_valid()
- Success message may only be one line.

Use when you’d like a success message added to your Delete response.

**Assumptions:**

- Class has method delete()

**Limitations:**

- Class cannot override delete()
- Success message may only be one line.

```
class toolkit.views.mixins.ViewMetaMixin
```

Mixin will be used capture optional and required meta data about each view that are then passed to the template

## 1.2 toolkit.models – Toolkit models

```
class toolkit.models.models.CCEAuditModel(*args, **kwargs)
```

Abstract model with fields for the user and timestamp of a row’s creation and last update.

---

**Note:**

- Inherits from **CCEModel**
  - Requires **django-cuser** package to determine current user
- 

**Tags** django-cuser

```
class toolkit.models.models.CCEModel(*args, **kwargs)
```

Abstract base model with permissions mixin.

```
class toolkit.models.mixins.ModelPermissionsMixin
```

Defines the permissions methods that most models need,

**Raises NotImplementedError** – if they have not been overridden.

**classmethod can\_create** (*user\_obj*)

CreateView needs permissions at class (table) level. We'll try it at instance level for a while and see how it goes.

**can\_delete** (*user\_obj*)

DeleteView needs permissions at instance (row) level.

**can\_update** (*user\_obj*)

UpdateView needs permissions at instance (row) level.

**can\_view** (*user\_obj*)

DetailView needs permissions at instance (row) level.

**classmethod can\_view\_list** (*user\_obj*)

ListView needs permissions at class (table) level. We'll try it at instance level for a while and see how it goes.

## 1.3 toolkit.forms – Toolkit forms

**class** toolkit.forms.forms.CCEModelForm (*\*args, \*\*kwargs*)

Django model form using CCE ModelFormMetaClass

**class** toolkit.forms.forms.CCEModelFormMetaClass

Defines custom Model Form Meta Class

**class** toolkit.forms.forms.CCEModelSearchForm (*\*args, \*\*kwargs*)

Django Model form with SearchFormMixin to create an Advanced Search form using multiple fields and filters

Allows filtering a queryset using a list of fields, Q objects and custom filter methods

**class** toolkit.forms.forms.CCESimpleSearchForm (*\*args, \*\*kwargs*)

Model Search Form with a single search field, provides similar functionality to django-admin search box

Allows filtering a queryset using a list of fields, Q objects and custom filter methods

### Fields

- search

**class** toolkit.forms.forms.DynamicNullBooleanSelect (*attrs=None, null\_label=None, true\_label=None, false\_label=None*)

An overridden Select widget to be used with NullBooleanField. Takes a kwarg “null\_label” that indicates the text on the null option.

**class** toolkit.forms.forms.ReportSelector (*user, reports\_list, \*args, \*\*kwargs*)

Form used for CCE Report Views to provide a dropdown of available reports

**class** toolkit.forms.forms.SearchForm (*data=None, files=None, auto\_id=u'id\_%s', pre-fix=None, initial=None, error\_class=<class 'django.forms.utils.ErrorList'>, label\_suffix=None, empty\_permitted=False, field\_order=None, use\_required\_attribute=None, renderer=None*)

Regular Django form with SearchFormMixin

toolkit.forms.forms.cce\_formfield\_callback (*f, \*\*kwargs*)

overrides django formfield widgets default values

**class** toolkit.forms.mixins.SearchFormMixin

Mixin used to create a search form. Allows you to define a list of field-filters pairs used to filter the queryset passed to the form

filters can be a combination of list of fields, Q objects and custom filter methods

**filters** (*queryset*)

Prepares the dictionary of queryset filters based on the form cleaned data

**Parameters** **queryset** (*Queryset*) – the queryset to filter through.

**Returns** tuple of q\_objects and kwargs used in filtering the queryset

**Warning:** method uses cleaned\_data, self.full\_clean() must be called first.

**search** (*queryset*)

Filter the given queryset according to the form's cleaned data.

**Warning:** self.full\_clean() must be called first.

**Parameters** **queryset** (*Queryset*) – the queryset to filter through.

**Returns** filtered version of the initial queryset.

**Raises** AttributeError, if the form is not valid.

## 1.4 toolkit.templatetags – Toolkit template tags

### 1.4.1 Pagination tags

toolkit.templatetags.pagination.**append\_querystrings\_as\_hidden\_fields** (*request*)

Append querystrings to a form as hidden fields

toolkit.templatetags.pagination.**generate\_ordering\_links** (*request, title, value*)

**Generates a Table heading with ordering paramaters based values passed** by AbstractedListMixin that allows users to order an object\_list table in a ListView by a particular column(ASC and DESC)

toolkit.templatetags.pagination.**generate\_page\_links** (*page\_obj, request*)

Generates the pagination footer in a ListView template

toolkit.templatetags.pagination.**url\_replace** (*request, value*)

Method used to the page querystring and return an update url

### 1.4.2 Bootstrap tags

toolkit.templatetags.bootstrap\_tags.**boolean\_icon** (*value*)

**Parameters** **value** (*Bool*) – Nullable boolean value

**Returns** **html** i tag with fontawesome icon representation

toolkit.templatetags.bootstrap\_tags.**render\_detail** (*value, param*)

**Returns** appropriate html rendering of value

### 1.4.3 Templates helpers

**class** `toolkit.templatetags.template_helpers.MakeListNode` (*items, varname*)  
Helper method for `make_list`

<http://stackoverflow.com/questions/3715550/creating-a-list-on-the-fly-in-a-django-template>

`toolkit.templatetags.template_helpers.is_bootstrap_styled_field` (*formfield*)

**Template tag used in a django template to check if a form field can be** styled using bootstrap

**Parameters** `formfield` (*FormField*) – Django form field

**Returns bool** True if the widget of the form field is one of the following - `widgets.TextInput` - `widgets.Textarea` - `widgets.NumberInput` - `widgets.EmailInput` - `widgets.Select` - `widgets.DateInput` - `widgets.PasswordInput` - `DynamicNullBooleanSelect` - `widgets.URLInput` - `widgets.NullBooleanSelect` - `widgets.SelectMultiple`

`toolkit.templatetags.template_helpers.is_date_field` (*formfield*)

**Template tag used in a django template to check if a form field is a** date field

**Parameters** `formfield` (*FormField*) – Django form field

**Returns bool** True if `FormField` is a date field

`toolkit.templatetags.template_helpers.is_datetime_field` (*formfield*)

**Template tag used in a django template to check if a form field is a** datetime field

**Parameters** `formfield` (*FormField*) – Django form field

**Returns bool** True if `FormField` is a datetime field

`toolkit.templatetags.template_helpers.is_time_field` (*formfield*)

**Template tag used in a django template to check if a form field is a** time field

**Parameters** `formfield` (*FormField*) – Django form field

**Returns bool** True if `FormField` is a time field

`toolkit.templatetags.template_helpers.make_list` (*parser, token*)

Template tag to create a python list within a Django template

**Usage:**

```
{% make_list var1 var2 var3 as some_list %}
```

<http://stackoverflow.com/questions/3715550/creating-a-list-on-the-fly-in-a-django-template>



### 2.1 `toolkit.helpers.admin` – django-admin helpers

`toolkit.helpers.admin.auto_admin_register(app_name, exclude=())`

Helper allows you to register models to django-admin in bulk.

Register all unregistered models in the given app with the admin site. Any previously-registered models will retain their original registration.

#### Parameters

- **app\_name** (*string*) – the name of the app to `auto_admin_register`.
- **exclude** (*iterable*) – an iterable of model names to exclude from registration

#### Usage:

```
1 from polls.models import Poll
2 from toolkit.admin import auto_admin_register
3
4 auto_admin_register(__package__, exclude=(Poll.__name__, ))
```

**Recommended usage:** make `auto_admin_register(__package__)` the last line in desired `admin.py` files.

**Caution:** Registers all models found in **app\_name** that are not listed in **exclude** with the django-admin site.

**Author:** Fredrick Wagner

## 2.2 toolkit.helpers.reports – custom reports helpers

`toolkit.helpers.reports.csv_response(filename, table)`

Return a CSV file of the given table as an HttpResponse.

Args:

**filename:** the name of the downloaded CSV file. The extension will be `‘.csv’`. This parameter is inserted directly to the response’s Content-Disposition, and must be escaped accordingly.

**table:** a 2-dimensional iterable, in row-major order.

Returns:

A CSV HttpResponse with appropriate content\_type and Content-Disposition.

`toolkit.helpers.reports.generate_basic_table(columns, data)`

Generate a table of functions applied to data objects.

Argument `‘columns’` is an iterable of 2-tuples of the form (title, function) where `‘title’` is the column title and `‘function’` is a single-parameter function that will be applied to each data element to create the column.

Returns a 2-dimensional row-major-ordered generator. The first row is the column titles; subsequent rows are evaluated functions of the data points.

Args:

**columns: an iterable of pairs (title, function):** title: the string to appear at the top of the column. function: a callable to be applied to each datum in the column.

**data: a QuerySet, list, or other iterable of arbitrary objects that can** be passed to the provided functions.

Yields:

**rows of the table:** first row: the given column titles. subsequent rows: the given column functions applied to each datum.

Example usage:

```
>>> columns = [('Numbers', lambda x: x), ('Squares', lambda x: x ** 2)]
>>> data = [1, 2, 3, 4, 5]
>>> list(list(row) for row in generate_basic_table(columns, data))
[['Numbers', 'Squares'], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25]]
```

**Author:** Fredrick Wagner

`toolkit.helpers.reports.generate_table(columns, data, model=None, capitalize_title=True, remove_nones=False, **kwargs)`

Wrapper around `generate_basic_table` with fancier column specifications.

Argument `‘columns’` is an iterable of specifications for table columns. Each specification is in one of three forms:

- **a tuple (title, func), where ‘title’ is the column title and ‘func’** is a function that will be applied to each datum in the column. Specifications in this format will be passed through unchanged.
- **a tuple (title, attr). The column function will be a lookup of the** provided attribute, which may be a model field, an ordinary object attribute, or a property.
- **a string, which is an attribute name as specified above. If the** attribute is a model field and the `‘model’` arg is given, this column’s title will be the model’s `verbose_name`; otherwise, the title will be the string, with single and double underscores converted to single spaces.

See also `getattr_chain`, which this function uses to look up attributes given by the latter two types of column specifications. Any keyword args are passed through to it.

Args:

`columns`: an iterable of column specifications.

**model**: the model to look up field names from. Only used to get verbose field names for column titles, and not used at all if all column specifications include titles.

**capitalize\_title**: if True (the default), derived column names will have their first letter capitalized.

**remove\_nones**: if True, 'None' results from column functions will be replaced with the empty string.

`kwargs`: passed through to function `'getattr_chain'`.

**Yields**: Column specifications in the form (title, func).

Example:

```
def scholarship_report_view(request):
    table = generate_table([
        'id',
        'parent',
        ('Submission Date', 'submission_date'),
        ('Email Address', Scholarship.get_user_email),
        ('Random Numbers', lambda _: random.random()),
    ], data=Scholarship.objects.all(), model=Scholarship)
    return csv_response('Scholarship Information', table)
```

Author:

Fredrick Wagner

`toolkit.helpers.reports.get_width(string, bold=False)`

Assuming a standard 10-point Arial font, returns the width of the string, in BIFF column width units.

`toolkit.helpers.reports.getattr_chain(obj, name_chain, suppress_attr_errors=False, sep='__')`

Apply `getattr` successively to a chain of attribute names.

Argument `'name_chain'` is a string containing sequence of attribute names to look up, starting with the initial object `'obj'` and progressing through the chain. By default, a double underscore (`'__'`) is expected to separate attribute names (as in Django's admin config and queryset keyword args), but any string may be specified in argument `'sep'`. If `'sep'` is None, argument `'name_chain'` is instead expected to be an already-separated iterable of attribute names.

When evaluating a chain of attributes such as `'foo__bar__baz'`, in some cases `'bar'` may sometimes be None, such as in database models with nullable foreign keys. In order to simplify the process of attempting to look up such values, argument `'suppress_attr_errors'` may be given: if it is True, any `AttributeErrors` raised by lookups on None (e.g., `'None.baz'`) will be caught, and the value None will be returned instead. (Attempted lookups of invalid names will still raise errors as usual.) Be aware, though, that specifying this option will result in the same behavior whether `'bar'` or `'baz'` is None.

Note that while Django's uses of such string-specified attribute lookups are limited to database relations, this function performs just as well with regular object attributes, and even with properties.

If a more complex lookup involving function calls or other logic is desired consider a lambda function, such as `lambda obj: obj.foo.bar.baz.qux()`.

Args:

**obj:** the object start the attribute lookup from.

**name\_chain:** a string containing a sequence of attribute names, separated by the value of argument 'sep'. May instead be an iterable of attribute names if 'sep' is None.

**suppress\_attr\_errors:** if True, catches `AttributeErrors` raised from an attempted lookup on a None value anywhere in the attribute chain, and returns None instead of raising the exception.

**sep:** the delimiting characters between the consecutive attribute names in argument 'name\_chain'. Default is '\_', but may be any string. If None, 'name\_chain' is expected to be an iterable sequence of names, rather than a single string.

**Returns:** The evaluation of the consecutive lookup of attributes in 'name\_chain'.

Example usage:

```
>>> class Obj(object): pass
>>> obj, obj.foo = Obj(), Obj()

>>> obj.foo.bar = None
>>> getattr_chain(obj, 'foo__bar')
>>> # None returned.
>>> getattr_chain(obj, 'foo__bar__baz')
Traceback (most recent call last):
...
AttributeError: 'NoneType' object has no attribute 'baz'
>>> getattr_chain(obj, 'foo__bar__baz', suppress_attr_errors=True)
>>> # None returned; no exception raised.

>>> obj.foo.bar = 'spam'
>>> getattr_chain(obj, 'foo__bar')
'spam'
>>> getattr_chain(obj, 'foo__bar__baz')
Traceback (most recent call last):
...
AttributeError: 'str' object has no attribute 'baz'
>>> getattr_chain(obj, 'foo__bar__baz', suppress_attr_errors=True)
Traceback (most recent call last):
...
AttributeError: 'str' object has no attribute 'baz'
>>> # Only AttributeError from NoneType are suppressed.
```

`toolkit.helpers.reports.write_to_worksheet(ws, row, column, cell)`

Write a single cell to a worksheet with xlwt. Used with `xls_multiple_worksheets_response`.

If "cell" is a dict and the key "merge" is present, the value of "merge" is also a dict with the potential to have keys called "row\_span" and "col\_span". These parameters indicate what cells (starting at row, column) should be merged together.

**Parameters** `cell` (*str or dict with keys label, style and merge*) – Simple or complex data to be written to the cell

`toolkit.helpers.reports.xls_multiple_worksheets_response(filename, data, padding=0)`

Take a filename and a dictionary (data) and return a .xls response that can have multiple sheets.

The user may indicate a style for a cell by passing in a dictionary with keys 'label' and 'style' instead of just a string.

The user may provide a header for each sheet. A header is a set of cells under the key “header” that appears first in the sheet.

data dict format:

```

1 mystyle = 'font: bold 1'
2 data = {
3     sheet_name: {
4         'header': [
5             ['cell 1:1', 'cell 1:2'],
6             ['cell 2:1', {'label': 'cell 2:2', 'style': mystyle}]
7         ],
8         'table': [
9             [{'label': 'cell 3:1', 'style': mystyle}, 'cell 3:2'],
10            ['cell 4:1', 'cell 4:2']
11        ]
12    },
13 }
```

Styles are XFStyle strings. Comprehensive documentation for the format of these strings is difficult to find.

**Brief example:** <http://xlwt.readthedocs.org/en/latest/api.html#xlwt.Style.easyxf>

**Our example:**

```

1 style = 'font: bold 1,
2         'name Tahoma, ' \
3         'height 160;' \
4         'borders: left thick, right thick, top thick, ' \
5         'bottom thick;' \
6         'pattern: pattern solid, pattern_fore_colour ' \
7         'yellow,pattern_back_colour yellow'
```

```

toolkit.helpers.reports.xls_response(filename, sheetname, table, header=None,
                                     footer=None, include_totals=False, total_label='Total', grouper_col=None,
                                     value_col=None)
```

Return a Microsoft Excel file of the given table as an HttpResponse.

Args:

**filename:** the name of the downloaded file. The extension will be ‘.xls’ This parameter is inserted directly to the response’s Content-Disposition, and must be escaped accordingly.

**sheetname:** the name of the spreadsheet.

**table:** a 2-dimensional iterable, in row-major order.

**header:** an optional 2-dimensional iterable, in row-major order.

**include\_totals:** an optional boolean to include total values.

**total\_label:** Name of the total column, defaults to ‘Total’

**grouper\_col:** Name of the group to subtotal values for (e.g. ‘Site’).

**value\_col:** Name of the column which holds the values to be summed (e.g. ‘Amount’).

Returns:

A Microsoft Excel HttpResponse with appropriate content\_type and Content-Disposition.

`toolkit.helpers.reports.xlsx_multiple_worksheets_response` (*filename*, *data*,  
*max\_width=118*,  
*max\_height=90*)

Takes a filename and an ordered dictionary (data) and returns an .xlsx response that can have multiple worksheets.

data dict format:

```
1 data = {  
2     sheet_name1: {  
3         'table': [  
4             ['cell 1:1', 'cell 1:2', 'cell 1:3'], # This is often the header row  
5             ['cell 2:1', 'cell 2:2', 'cell 2:3'],  
6             ['cell 3:1', 'cell 3:2', 'cell 3:3'],  
7         ]  
8     },  
9     sheet_name2: {  
10        'table': [  
11            ['cell 1:1', 'cell 1:2'],  
12            ['cell 2:1', 'cell 2:2'],  
13            ['cell 3:1', 'cell 3:2'],  
14            ['cell 4:1', 'cell 4:2'],  
15        ]  
16    },  
17 }
```

`toolkit.helpers.reports.xlsx_response` (*filename*, *table*, *max\_width=118*, *max\_height=90*)

**Return a Microsoft Excel 2007+ file of the given table as an `HttpResponse`.**

Args:

*filename*: the name of the downloaded file. The extension will be `.xlsx`. This parameter is inserted directly to the response's Content-Disposition, and must be escaped accordingly.

*table*: a 2-dimensional iterable, in row-major order.

Returns:

A Microsoft Excel 2007+ `HttpResponse` with appropriate `content_type` and `Content-Disposition`.

## 2.3 `toolkit.helpers.utils` – miscellaneous helpers

`toolkit.helpers.utils.generate_username_from_name` (*first\_name*, *last\_name*)

Method generates a valid username based off the given first and last names. It ensures that the username is unique by querying the user model. Usernames are generated by combining the first letter of the first name and the full last name (fbar). If this combination already exists, a number is appended to the username (fbar1, fbar2) and retested until a unique username is found.

### Parameters

- **first\_name** (*string*) – user first name
- **last\_name** (*string*) – user last name

**Returns** unique, valid username based off the given first and last names

**Raises** `IndexError` – if *first\_name* is empty or contains no characters valid for use in a username.

---

**Note:** The method will not create the user object, it will only return a valid username that can be used in creating a user object outside this method

---

**Usage:**

```

1  >>> generate_username_from_name('Foo', 'Bar')
2  fbar
3  >>> generate_username_from_name('Foo', 'Bar')
4  fbar1
5  >>> generate_username_from_name('Foo', 'Bar')
6  fbar2

```

`toolkit.helpers.utils.get_object_or_none(klass, *args, **kwargs)`

Method returns the queried object or None if the object does not exist

**Usage:**

```

1  >>> get_object_or_none(Poll, pk=1)
2  None
3  >>> get_object_or_none(Poll, {'pk': 1, 'user': 1})
4  None
5  >>> get_object_or_none(Poll, {'pk': 2})
6  Poll object

```

`toolkit.helpers.utils.get_subclass_instance(obj)`

Returns the utilized child class instance of a superclass instance.

**Parameters** `obj` (*Model*) – Django Model instance

**Returns** Subclass instance or None

`toolkit.helpers.utils.hasfield(model, field_name)`

Returns whether the specified `field_name` string is a valid field on `model` or its related models

**Parameters**

- **model** (*Model*) – Django Model object
- **field\_name** (*string*) – attribute string, dotted or dundered. example: 'user.first\_name' or 'user\_\_first\_name'

**Returns** Field object or False

**Usage:**

```

1  >>> hasfield(Poll, 'question')
2  Django Model
3  >>> hasfield(Poll, 'user__name')
4  Django Model
5  >>> hasfield(Poll, 'user.username')
6  Django Model
7  >>> hasfield(Poll, 'user.full_name')
8  False # full_name is a property method not a field

```

`toolkit.helpers.utils.snakify(*args, **kwargs)`

Converts to ASCII. Converts spaces to underscores. Removes characters that aren't alphanumerics, underscores, or hyphens. Converts to lowercase. Also strips leading and trailing whitespace.

**Parameters** `value` (*string*) – unsanitized value

**Returns** snakified value

**Usage:**

```
1 >>> snakify('polls-report May 1, 2016')
2 u'polls_report_may_1_2016'
```

`toolkit.helpers.utils.usenamify(username, special_chars='@.+_-')`

Remove characters invalid for use in a username and convert to lowercase.

**Parameters**

- **username** (*string*) – unsanitized string
- **special\_chars** (*string*) – special characters that need to be removed from the provided username. *Default value:* '@.+\_-'

**Returns** sanitized string to be used as username

---

**Note:** If username contains no valid characters, the returned value will be the empty string, which is not a valid username on its own.

---

**Author:** Fredrick Wagner

## 2.4 toolkit.helpers.bdd – BDD helpers

`toolkit.helpers.bdd.bdd.assert_text_in_table(browser, values, date_format='%m/%d/%Y')`

Asserts that a list of values exists in table rows on the page

**Parameters**

- **browser** – browser object
- **values** (*list*) – list of values
- **date\_format** (*string*) – optional field to specify a specific string format for date values sent in the values list

`toolkit.helpers.bdd.bdd.assert_text_not_in_table(browser, values, date_format='%m/%d/%Y')`

Asserts that a list of values does not exist in table rows on the page

**Parameters**

- **browser** – browser object
- **values** (*list*) – list of values
- **date\_format** (*string*) – optional field to specify a specific string format for date values sent in the values list



`toolkit.helpers.bdd.bdd.assert_text_on_page` (*browser*, *values*,  
*date\_format*='%m/%d/%Y')

Asserts that a list of values does exist on the page

#### Parameters

- **browser** – browser object
- **values** (*list*) – list of values
- **date\_format** (*string*) – optional field to specify a specific string format for date values sent in the values list

`toolkit.helpers.bdd.bdd.click_element_by_name` (*browser*, *name*, *index*=0)

Clicks an element in the DOM by the element name

#### Parameters

- **browser** – browser object
- **name** (*string*) – name of element
- **index** (*integer*) – index of the element in the DOM

`toolkit.helpers.bdd.bdd.compare_content_types` (*browser*, *context*, *file\_type*)

Attempts to find the download link, request the file and asserts that the file extension matches the expected file type

`toolkit.helpers.bdd.bdd.fill_and_submit_form` (*browser*, *fields*, *sub-*  
*mit\_button\_name*='submit')

Fills a dictionary of form fields on a page and clicks the submit button

#### Parameters

- **browser** – browser object
- **fields** – iterable of fields
- **submit\_button\_name** (*string*) – optional button name field in case there's multiple buttons on the page

`toolkit.helpers.bdd.bdd.fill_form` (*browser*, *fields*)

Fills a dictionary of form fields on a page

#### Parameters

- **browser** – browser object
- **fields** – iterable of fields

`toolkit.helpers.bdd.bdd.get_file_content_type` (*extension*)

**Parameters** *extension* (*string*) – file extension

**Returns** *string* mime type based on file extension

`toolkit.helpers.bdd.bdd.log` (*context*, *text*)

logs text in the js console

`toolkit.helpers.bdd.bdd.scroll_to_top` (*browser*)

executes the following js code on the page: `window.scrollTo(0, 0)`

`toolkit.helpers.bdd.bdd.set_test_obj` (*context*, *model*)

Sets the test object

`toolkit.helpers.bdd.bdd.set_test_obj_pk` (*context*)

Sets the test object pk

```
toolkit.helpers.bdd.utils.setup_test_environment (context, scenario, visible=0,  
                                                  use_xvfb=True)
```

**Method used to setup the BDD test environment**

- Sets up virtual display
- Sets up webdriver instance
- Sets window size
- Flushes cookies
- Enables debug (Allows for more verbose error screens)
- Sets scenario
- Truncates database tables

**Options:**

- `visible` (0 or 1) - Toggle Xephyr to view the Xvfb instance for limited debugging. 0: Off, 1: On.
- `use_xvfb` (True/False) - Toggle Xvfb to run the tests on your desktop for in-depth debugging.

```
toolkit.helpers.bdd.shared_steps.log_in_as (context, user)
```

Allows you to log in as a user in the system. I am logged in as (.\*)

**Parameters**

- **context** (*object*) – behave’s global object
- **user** (*string*) – user in the system

**Usage:**

```
1 Scenario Outline: Manage applications
2   Given I am logged in as manager
```

```
toolkit.helpers.bdd.shared_steps.visit_page (context, url_name)
```

Allows you to visit a page in the system. I visit (.\*)

**Parameters**

- **context** (*object*) – behave’s global object
- **url\_name** (*string*) – url name to visit

**Usage:**

```
1 Scenario Outline: Manage applications
2   Given I am logged in as manager
3   When I visit manage_application
```

```
toolkit.helpers.bdd.shared_steps.verify_navigation (context, url_names)
```

Allows you to find a link on the current page

**Parameters**

- **context** (*object*) – behave’s global object
- **url\_names** (*string*) – url name(s) to visit, add comma between url name if there’s more than one

**Usage:**

```

1 Scenario Outline: Manage applications
2   Given I am logged in as manager
3   When I visit manage_application
4   Then I should find a link to add_application

```

`toolkit.helpers.bdd.shared_steps.verify_post_form(context, should_not, url_name)`

Allows you know if there's a form that can be posted to a certain place on the page. I should( not)? find a form that posts to (.\*)

**Parameters**

- **context** (*object*) – behave's global object
- **should\_not** (*string*) – include “not” if the form should not be found
- **url\_name** (*string*) – url name to visit

**Usage 1:**

```

1 Scenario Outline: Manage applications
2   Given I am logged in as manager
3   And I have a valid submitted application
4   When I visit view_application
5   Then I should find a form that posts to approve_application

```

**Usage 2:**

```

1 Scenario Outline: Manage applications
2   Given I am logged in as manager
3   And I have a valid application
4   When I visit view_application
5   Then I should not find a form that posts to approve_application

```

`toolkit.helpers.bdd.shared_steps.verify_file_download(context, should_or_shouldnt, file_type)`

Allows you to compare the file download type. I (should|shouldn't) receive a (.\* ) file download

**Parameters**

- **context** (*object*) – behave's global object
- **should\_or\_shouldnt** (*string*) – should or shouldn't
- **file\_type** (*string*) – file type

**Usage:**

```

1 Scenario: Download as .docx
2   Given I am logged in as manager
3   And I have valid applications
4   When I go to application_report
5   Then I should receive a docx file download

```



### 3.1 `toolkit.apps.breadcrumbs` – CCE breadcrumbs

`toolkit.apps.breadcrumbs.templatetags.breadcrumbs.breadcrumbs` (*context*)  
Tag renders breadcrumb in template

### 3.2 `toolkit.apps.toolkit_activity_log` – CCE Activity Log

### 3.3 `toolkit.apps.fabfile` – Fabric recipes

#### 3.3.1 Fabfile

#### 3.3.2 Django Tools

#### 3.3.3 Environment

#### 3.3.4 Git Tools

#### 3.3.5 Server Tools



### 4.1 License

Copyright (c) 2016, CCE IT and individual contributors. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of CCE IT nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

### 4.2 Contact

Questions? Please contact [devs@cce.ou.edu](mailto:devs@cce.ou.edu)





## CHAPTER 5

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### t

- `toolkit.apps.breadcrumbs.templatetags.breadcrumbs,`  
25
- `toolkit.forms.forms,` 10
- `toolkit.forms.mixins,` 10
- `toolkit.helpers.admin,` 13
- `toolkit.helpers.bdd.bdd,` 20
- `toolkit.helpers.bdd.shared_steps,` 22
- `toolkit.helpers.bdd.utils,` 21
- `toolkit.helpers.reports,` 14
- `toolkit.helpers.utils,` 18
- `toolkit.models.mixins,` 9
- `toolkit.models.models,` 9
- `toolkit.templatetags.bootstrap_tags,` 11
- `toolkit.templatetags.pagination,` 11
- `toolkit.templatetags.template_helpers,`  
12
- `toolkit.views.mixins,` 6
- `toolkit.views.views,` 1



## A

AbstractedDetailMixin (class in toolkit.views.mixins), 6  
 AbstractedListMixin (class in toolkit.views.mixins), 7  
 append\_querystrings\_as\_hidden\_fields() (in module toolkit.templatetags.pagination), 11  
 AppendURLObjectsMixin (class in toolkit.views.mixins), 8  
 assert\_text\_in\_table() (in module toolkit.helpers.bdd.bdd), 20  
 assert\_text\_not\_in\_table() (in module toolkit.helpers.bdd.bdd), 20  
 assert\_text\_on\_page() (in module toolkit.helpers.bdd.bdd), 20  
 auto\_admin\_register() (in module toolkit.helpers.admin), 13

## B

boolean\_icon() (in module toolkit.templatetags.bootstrap\_tags), 11  
 breadcrumbs() (in module toolkit.apps.breadcrumbs.templatetags.breadcrumbs), 25

## C

can\_create() (toolkit.models.mixins.ModelPermissionsMixin class method), 9  
 can\_delete() (toolkit.models.mixins.ModelPermissionsMixin method), 10  
 can\_update() (toolkit.models.mixins.ModelPermissionsMixin method), 10  
 can\_view() (toolkit.models.mixins.ModelPermissionsMixin method), 10  
 can\_view\_list() (toolkit.models.mixins.ModelPermissionsMixin class method), 10  
 cce\_formfield\_callback() (in module toolkit.forms.forms), 10  
 CCEAuditModel (class in toolkit.models.models), 9  
 CCECreateView (class in toolkit.views.views), 1  
 CCECreateWithInlinesView (class in toolkit.views.views), 2

CCEDeleteView (class in toolkit.views.views), 2  
 CCEDetailView (class in toolkit.views.views), 2  
 CCEFormView (class in toolkit.views.views), 2  
 CCEListView (class in toolkit.views.views), 2  
 CCEModel (class in toolkit.models.models), 9  
 CCEModelForm (class in toolkit.forms.forms), 10  
 CCEModelFormMetaclass (class in toolkit.forms.forms), 10  
 CCEModelFormSetView (class in toolkit.views.views), 4  
 CCEModelSearchForm (class in toolkit.forms.forms), 10  
 CCEObjectRedirectView (class in toolkit.views.views), 4  
 CCERedirectView (class in toolkit.views.views), 4  
 CCESearchView (class in toolkit.views.views), 4  
 CCESimpleSearchForm (class in toolkit.forms.forms), 10  
 CCETemplateView (class in toolkit.views.views), 5  
 CCEUpdateView (class in toolkit.views.views), 5  
 CCEUpdateWithInlinesView (class in toolkit.views.views), 6  
 ClassPermissionsMixin (class in toolkit.views.mixins), 8  
 click\_element\_by\_name() (in module toolkit.helpers.bdd.bdd), 21  
 compare\_content\_types() (in module toolkit.helpers.bdd.bdd), 21  
 csv\_response() (in module toolkit.helpers.reports), 14

## D

DynamicNullBooleanSelect (class in toolkit.forms.forms), 10

## F

FileDownloadMixin (class in toolkit.views.mixins), 8  
 fill\_and\_submit\_form() (in module toolkit.helpers.bdd.bdd), 21  
 fill\_form() (in module toolkit.helpers.bdd.bdd), 21  
 filters() (toolkit.forms.mixins.SearchFormMixin method), 11

## G

generate\_basic\_table() (in module toolkit.helpers.reports), 14

`generate_list_view_table()`  
(`toolkit.views.mixins.AbstractedListMixin`  
method), 7

`generate_ordering_links()` (in module  
`toolkit.templatetags.pagination`), 11

`generate_page_links()` (in module  
`toolkit.templatetags.pagination`), 11

`generate_table()` (in module `toolkit.helpers.reports`), 14

`generate_username_from_name()` (in module  
`toolkit.helpers.utils`), 18

`get_context_data()` (`toolkit.views.mixins.AbstractedListMixin`  
method), 7

`get_context_data()` (`toolkit.views.views.CCESearchView`  
method), 5

`get_detail_fields()` (`toolkit.views.mixins.AbstractedDetailMixin`  
method), 6

`get_details()` (`toolkit.views.mixins.AbstractedDetailMixin`  
method), 6

`get_file_content_type()` (in module  
`toolkit.helpers.bdd.bdd`), 21

`get_object_or_none()` (in module `toolkit.helpers.utils`), 19

`get_queryset()` (`toolkit.views.views.CCEListView`  
method), 4

`get_queryset()` (`toolkit.views.views.CCESearchView`  
method), 5

`get_subclass_instance()` (in module `toolkit.helpers.utils`),  
19

`get_width()` (in module `toolkit.helpers.reports`), 15

`getattr_chain()` (in module `toolkit.helpers.reports`), 15

## H

`hasfield()` (in module `toolkit.helpers.utils`), 19

## I

`is_bootstrap_styled_field()` (in module  
`toolkit.templatetags.template_helpers`), 12

`is_date_field()` (in module  
`toolkit.templatetags.template_helpers`), 12

`is_datetime_field()` (in module  
`toolkit.templatetags.template_helpers`), 12

`is_time_field()` (in module  
`toolkit.templatetags.template_helpers`), 12

## L

`log()` (in module `toolkit.helpers.bdd.bdd`), 21

`log_in_as()` (in module `toolkit.helpers.bdd.shared_steps`),  
22

## M

`make_list()` (in module  
`toolkit.templatetags.template_helpers`), 12

`MakeListNode` (class in  
`toolkit.templatetags.template_helpers`), 12

`ModelPermissionsMixin` (class in `toolkit.models.mixins`),  
9

## O

`ObjectPermissionsMixin` (class in `toolkit.views.mixins`),  
8

## P

`PermissionsRequiredMixin` (class in  
`toolkit.views.mixins`), 8

## R

`render_button()` (`toolkit.views.mixins.AbstractedListMixin`  
method), 7

`render_buttons()` (`toolkit.views.mixins.AbstractedListMixin`  
method), 7

`render_detail()` (in module  
`toolkit.templatetags.bootstrap_tags`), 11

`ReportDownloadDetailView` (class in  
`toolkit.views.views`), 6

`ReportDownloadSearchView` (class in  
`toolkit.views.views`), 6

`ReportSelector` (class in `toolkit.forms.forms`), 10

## S

`scroll_to_top()` (in module `toolkit.helpers.bdd.bdd`), 21

`search()` (`toolkit.forms.mixins.SearchFormMixin`  
method), 11

`SearchForm` (class in `toolkit.forms.forms`), 10

`SearchFormMixin` (class in `toolkit.forms.mixins`), 10

`set_test_obj()` (in module `toolkit.helpers.bdd.bdd`), 21

`set_test_obj_pk()` (in module `toolkit.helpers.bdd.bdd`), 21

`setup_test_environment()` (in module  
`toolkit.helpers.bdd.utils`), 21

`snakify()` (in module `toolkit.helpers.utils`), 19

`SuccessMessageMixin` (class in `toolkit.views.mixins`), 9

## T

`toolkit.apps.breadcrumbs.templatetags.breadcrumbs`  
(module), 25

`toolkit.forms.forms` (module), 10

`toolkit.forms.mixins` (module), 10

`toolkit.helpers.admin` (module), 13

`toolkit.helpers.bdd.bdd` (module), 20

`toolkit.helpers.bdd.shared_steps` (module), 22

`toolkit.helpers.bdd.utils` (module), 21

`toolkit.helpers.reports` (module), 14

`toolkit.helpers.utils` (module), 18

`toolkit.models.mixins` (module), 9

`toolkit.models.models` (module), 9

`toolkit.templatetags.bootstrap_tags` (module), 11

`toolkit.templatetags.pagination` (module), 11

`toolkit.templatetags.template_helpers` (module), 12

toolkit.views.mixins (module), [6](#)

toolkit.views.views (module), [1](#)

## U

url\_replace() (in module toolkit.templatetags.pagination),  
[11](#)

usernamify() (in module toolkit.helpers.utils), [20](#)

## V

verify\_file\_download() (in module  
toolkit.helpers.bdd.shared\_steps), [23](#)

verify\_navigation() (in module  
toolkit.helpers.bdd.shared\_steps), [22](#)

verify\_post\_form() (in module  
toolkit.helpers.bdd.shared\_steps), [23](#)

ViewMetaMixin (class in toolkit.views.mixins), [9](#)

visit\_page() (in module toolkit.helpers.bdd.shared\_steps),  
[22](#)

## W

write\_to\_worksheet() (in module toolkit.helpers.reports),  
[16](#)

## X

xls\_multiple\_worksheets\_response() (in module  
toolkit.helpers.reports), [16](#)

xls\_response() (in module toolkit.helpers.reports), [17](#)

xlsx\_multiple\_worksheets\_response() (in module  
toolkit.helpers.reports), [17](#)

xlsx\_response() (in module toolkit.helpers.reports), [18](#)