# tri*table*Documentation

***Release 8.6.0***

**Anders Hovmöller**

**Dec 01, 2021**

# Contents

**Warning:** tri.table is end of life. It has been merged into iommi.

iommi is backwards incompatible but the porting effort should be fairly mild, the biggest changes are that *show* is now called *include*, tri.query's *Variable* is renamed to *Filter* and plural is used consistently for containers (so *column__foo* is *columns__foo* in iommi).

tri.table is a library to make full featured HTML tables easily:

- generates header, rows and cells
- grouping of headers
- filtering
- sorting
- bulk edit
- pagination
- automatic rowspan
- link creation
- customization on multiple levels, all the way down to templates for cells

All these examples and a bigger example using many more features can be found in the examples django project.

Read the full documentation for more.

# CHAPTER 1

## Simple example

```
def readme_example_1(request):
    # Say I have a class...
    class Foo(object):
        def __init__(self, i):
            self.a = i
            self.b = 'foo %s' % (i % 3)
            self.c = (i, 1, 2, 3, 4)

    # and a list of them
    foos = [Foo(i) for i in range(4)]

    # I can declare a table:
    class FooTable(Table):
        a = Column.number()  # This is a shortcut that results in the css class "rj"␣
↪(for right justified) being added to the header and cell
        b = Column()
        c = Column(cell__format=lambda table, column, row, value, **_: value[-1])  #␣
↪Display the last value of the tuple
        sum_c = Column(cell__value=lambda table, column, row, **_: sum(row.c),␣
↪sortable=False)  # Calculate a value not present in Foo

    # now to get an HTML table:
    return render_table_to_response(request, table=FooTable(data=foos), template=
↪'base.html')
```

And this is what you get:

| A | B | C | Sum c |
|---|---|---|---|
| 0 | foo 0 | 4 | 10 |
| 1 | foo 1 | 4 | 11 |
| 2 | foo 2 | 4 | 12 |
| 3 | foo 0 | 4 | 13 |

CHAPTER 2

# Downloading tri.table and running examples

```
git clone https://github.com/TriOptima/tri.table.git
cd tri.table/examples
virtualenv venv
source venv/bin/activate
pip install "django>=1.8,<1.9"
pip install tri.table
python manage.py migrate
python manage.py runserver localhost:8000
# Now point your browser to localhost:8000 in order to view the examples.
```

# CHAPTER 3

## Fancy django features

Say I have some models:

```
class Foo(models.Model):
    name = models.CharField(max_length=255)
    a = models.IntegerField()

    def __unicode__(self):
        return self.name
```

```
class Bar(models.Model):
    b = models.ForeignKey(Foo, on_delete=models.CASCADE)
    c = models.CharField(max_length=255)
```

Now I can display a list of Bars in a table like this:

```
def readme_example_2(request):
    fill_dummy_data()

    class BarTable(Table):
        select = Column.select()  # Shortcut for creating checkboxes to select rows
        b__a = Column.number(  # Show "a" from "b". This works for plain old objects␣
↪too.
            query__show=True,  # put this field into the query language
            query__gui__show=True)  # put this field into the simple filtering GUI
        c = Column(
            bulk__show=True,  # Enable bulk editing for this field
            query__show=True,
            query__gui__show=True)

    return render_table_to_response(request, table=BarTable(data=Bar.objects.all()),␣
↪template='base.html', paginate_by=20)
```

This gives me a view with filtering, sorting, bulk edit and pagination.

All these examples and a bigger example using many more features can be found in the examples django project.

Read the full documentation for more.

CHAPTER 4

# Running tests

You need tox installed then just *make test*.

CHAPTER 5

# License

BSD

CHAPTER 6

# Documentation

http://tritable.readthedocs.org.

CHAPTER 7

Contents:

## 7.1 Installation

At the command line:

```
$ pip install tri.table
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv tri.table
$ pip install tri.table
```

## 7.2 Usage

You need to add *tri.table* to installed apps or copy the templates to your own template directory.

Other than that it should just be to *from tri_table import Table, Column* and off you go.

### 7.2.1 Styling

There is a *table.scss* file included with the project that provides a default styling for the table if you want a fast and easy starting point for styling.

## 7.3 Architecture overview

tri.table is built on top of tri.form (a form library) and tri.query (a query/search/filter library). Both tri.form and tri.query are written in the same API style as tri.table, which enables great integration and cohesion. All three libraries are built on top of tri.declarative. This has some nice consequences that I'll try to explain here.

### 7.3.1 Declarative/programmatic hybrid API

The `@declarative`, `@with_meta` and `@creation_ordered` decorators from tri_declarative enables us to very easily write an API that can look both like a normal simple python API:

```
my_table = Table(
    columns=[
        Column(name='foo'),
        Column('bar'),
    ],
    sortable=False)
```

This code is hopefully pretty self explanatory. But the cool thing is that we can do the exact same thing with a declarative style:

```
class MyTable(Table):
    foo = Column()
    bar = Column()

    class Meta:
        sortable = False

my_table = MyTable()
```

This style can be much more readable. There's a subtle different though between the first and second styles: the second is really a way to declare defaults, not hard coding values. This means we can create instances of the class and set the values in the call to the constructor:

```
my_table = MyTable(
    column__foo__show=False,   # <- hides the column foo
    sortable=True,             # <- turns on sorting again
)
```

... without having to create a new class inheriting from `MyTable`. So the API keeps all the power of the simple style and also getting the nice syntax of a declarative API.

### 7.3.2 Namespace dispatching

I've already hinted at this above in the example where we do `column__foo__show=False`. This is an example of the powerful namespace dispatch mechanism from tri_declarative. It's inspired by the query syntax of Django where you use __ to jump namespace. (If you're not familiar with Django, here's the gist of it: you can do `Table.objects.filter(foreign_key__column='foo')` to filter.) We really like this style and have expanded on it. It enables functions to expose the *full* API of functions it calls while still keeping the code simple. Here's a contrived example:

```
from tri_declarative import dispatch, EMPTY


@dispatch(
    b__x=1,  # these are default values. "b" here is implicitly
             # defining a namespace with a member "x" set to 1
    c__y=2,
)
def a(foo, b, c):
    print('foo:', foo)
```

(continues on next page)

(continued from previous page)

```
    some_function(**b)
    another_function(**c)


@dispatch (
    d=EMPTY,  # explicit namespace
)
def some_function(x, d):
    print('x:', x)
    another_function(**d)


def another_function(y=None, z=None):
    if y:
        print('y:', y)
    if z:
        print('z:', z)

# now to call a()!
a('q')
# output:
# foo: q
# x: 1
# y: 2


a('q', b__x=5)
# foo: q
# x: 5
# y: 2

a('q', b__d__z=5)
# foo: q
# x: 1
# z: 5
# y: 2
```

This is really useful in tri.table as it means we can expose the full feature set of the underling tri.query and tri.form libraries by just dispatching keyword arguments downstream. It also enables us to bundle commonly used features in what we call "shortcuts", which are pre packaged sets of defaults.

## 7.4 API documentation

### 7.4.1 Table

Describe a table. Example:

```
class FooTable(Table):
    a = Column()
    b = Column()

    class Meta:
        sortable = False
        attrs__style = 'background: green'
```

### Refinable members

- *actions*
- *actions_template*
- ***attrs*** dict of strings to string/callable of HTML attributes to apply to the table
- *bulk*
- ***bulk_exclude*** exclude filters to apply to the QuerySet before performing the bulk operation
- ***bulk_filter*** filters to apply to the QuerySet before performing the bulk operation
- *column*
- *default_sort_order*
- *endpoint*
- *endpoint_dispatch_prefix*
- *extra*
- *filter*
- *form_class*
- *header*
- *links*
- *member_class*
- *model*
- *name*
- *page_size*
- *paginator*
- *preprocess_data*
- *preprocess_row*
- *query_class*
- *row*
- ***sortable*** set this to false to turn off sorting for all columns
- *superheader*
- *template*

### Defaults

- ***actions_template***
  - *tri_form/actions.html*
- ***attrs__class__listview***
  - *True*
- ***endpoint__bulk***

  - *lambda table, key, value: table.bulk_form.endpoint_dispatch(key=key, value=value) if table.bulk is not None else None*

- ***endpoint__query***
  - *lambda table, key, value: table.query.endpoint_dispatch(key=key, value=value) if table.query is not None else None*
- ***filter__template***
  - *tri_query/form.html*
- ***header__template***
  - *tri_table/table_header_rows.html*
- ***page_size***
  - *40*
- ***paginator__template***
  - *tri_table/paginator.html*
- ***row__template***
  - *None*
- ***sortable***
  - *True*
- ***superheader__attrs__class__superheader***
  - *True*
- ***superheader__template***
  - *tri_table/header.html*
- ***template***
  - *tri_table/list.html*

### 7.4.2 Column

Class that describes a column, i.e. the text of the header, how to get and display the data in the cell, etc.

#### Refinable members

- *after*
- ***attr*** What attribute to use, defaults to same as name. Follows django conventions to access properties of properties, so "foo__bar" is equivalent to the python code *foo.bar*. This parameter is based on the variable name of the Column if you use the declarative style of creating tables.
- ***auto_rowspan*** enable automatic rowspan for this column. To join two cells with rowspan, just set this auto_rowspan to True and make those two cells output the same text and we'll handle the rest.
- *bulk*
- *cell*
- *choices*

- *data_retrieval_method*
- ***display_name*** the text of the header for this column. By default this is based on the *name* parameter so normally you won't need to specify it.
- *extra*
- ***group*** string describing the group of the header. If this parameter is used the header of the table now has two rows. Consecutive identical groups on the first level of the header are joined in a nice way.
- *header*
- *model*
- *model_field*
- ***name*** the name of the column
- *query*
- ***show*** set this to False to hide the column
- ***sort_default_desc*** Set to True to make table sort link to sort descending first.
- ***sort_key*** string denoting what value to use as sort key when this column is selected for sorting. (Or callable when rendering a table from list.)
- ***sortable*** set this to False to disable sorting on this column
- *superheader*
- ***url*** URL of the header. This should only be used if "sorting" is off.

### Defaults

- ***auto_rowspan***
  - *False*
- ***bulk__show***
  - *False*
- ***cell__format***
  - *tri_table.default_cell_formatter*
- ***cell__template***
  - *None*
- ***cell__url***
  - *None*
- ***cell__url_title***
  - *None*
- ***cell__value***
  - *lambda table, column, row, **_: getattr_path(row, evaluate(column.attr, table=table, column=column))*
- ***data_retrieval_method***
  - *DataRetrievalMethods.attribute_access*

- ***header__attrs__class__ascending***
  - *lambda bound_column, **_: bound_column.sort_direction == ASCENDING*
- ***header__attrs__class__descending***
  - *lambda bound_column, **_: bound_column.sort_direction == DESCENDING*
- ***header__attrs__class__first_column***
  - *lambda header, **_: header.index_in_group == 0*
- ***header__attrs__class__sorted_column***
  - *lambda bound_column, **_: bound_column.is_sorting*
- ***header__attrs__class__subheader***
  - *True*
- ***header__template***
  - *tri_table/header.html*
- ***query__show***
  - *False*
- ***show***
  - *True*
- ***sort_default_desc***
  - *False*
- ***sortable***
  - *True*

### Shortcuts

#### *boolean*

Shortcut to render booleans as a check mark if true or blank if false.

#### *boolean_tristate*

#### *choice*

#### *choice_queryset*

#### *date*

#### *datetime*

#### *decimal*

#### delete

Shortcut for creating a clickable delete icon. The URL defaults to *your_object.get_absolute_url() + 'delete/'*. Specify the option cell__url to override.

#### download

Shortcut for creating a clickable download icon. The URL defaults to *your_object.get_absolute_url() + 'download/'*. Specify the option cell__url to override.

#### edit

Shortcut for creating a clickable edit icon. The URL defaults to *your_object.get_absolute_url() + 'edit/'*. Specify the option cell__url to override.

#### email

#### float

#### foreign_key

#### icon

Shortcut to create font awesome-style icons.

**param icon** the font awesome name of the icon

#### integer

#### link

#### many_to_many

#### multi_choice

#### multi_choice_queryset

#### number

#### run

Shortcut for creating a clickable run icon. The URL defaults to *your_object.get_absolute_url() + 'run/'*. Specify the option cell__url to override.

#### *select*

Shortcut for a column of checkboxes to select rows. This is useful for implementing bulk operations.

**param checkbox_name** the name of the checkbox. Default is "pk", resulting in checkboxes like "pk_1234".

**param checked** callable to specify if the checkbox should be checked initially. Defaults to False.

#### *substring*

#### *text*

#### *time*

### 7.4.3 Form

Describe a Form. Example:

```
class MyForm(Form):
    a = Field()
    b = Field.email()

form = MyForm(data={})
```

You can also create an instance of a form with this syntax if it's more convenient:

```
form = MyForm(data={}, fields=[Field(name='a'), Field.email(name='b')])
```

See tri.declarative docs for more on this dual style of declaration.

**type fields** list of Field :type data: dict[basestring, any] :type model: django.db.models.Model

#### Refinable members

- *actions*
- *actions_template*
- *attrs*
- *base_template*
- *editable*
- *endpoint*
- *endpoint_dispatch_prefix*
- *extra*
- *field*
- *is_full_form*
- *links*
- *links_template*

- *member_class*
- *model*
- *name*
- *post_validation*

#### Defaults

- ***actions__submit__call_target***
  - *tri_form.submit*
- ***actions_template***
  - *tri_form/actions.html*
- ***attrs__action***
  - *""*
- ***attrs__method***
  - *post*
- ***editable***
  - *True*
- ***endpoint__field***
  - *tri_form.default_endpoint__field*
- ***is_full_form***
  - *True*
- ***links_template***
  - *tri_form/links.html*

### 7.4.4 Field

Class that describes a field, i.e. what input controls to render, the label, etc.

Note that, in addition to the parameters with the defined behavior below, you can pass in any keyword argument you need yourself, including callables that conform to the protocol, and they will be added and evaluated as members.

> All these parameters can be callables, and if they are, will be evaluated with the keyword arguments form and field. The only exceptions are is_valid (which gets form, field and parsed_data), render_value (which takes form, field and value) and parse (which gets form, field, string_value). Example of using a lambda to specify a value:

```
Field(id=lambda form, field: 'my_id_%s' % field.name)
```

#### Refinable members

- *after*
- ***attr*** the attribute path to apply or get the data from. For example using "foo__bar__baz" will result in *your_instance.foo.bar.baz* will be set by the apply() function. Defaults to same as name

- ***attrs*** a dict containing any custom html attributes to be sent to the input_template.
- *choice_to_option*
- *choice_tuples*
- *choices*
- *container*
- *display_name*
- ***editable*** default: True
- *empty_choice_tuple*
- *empty_label*
- *endpoint*
- *endpoint_dispatch*
- *endpoint_path*
- ***errors_template*** django template filename for the template for just the errors output. Default: 'tri_form/errors.html'
- *extra*
- ***help_text*** The help text will be grabbed from the django model if specified and available. Default: lambda form, field: '' if form.model is None else form.model._meta.get_field_by_name(field.name)[0].help_text or ''
- ***id*** the HTML id attribute. Default: 'id_%s' % name
- ***initial*** initial value of the field
- *initial_list*
- *input_container*
- ***input_template*** django template filename for the template for just the input control. Default: 'tri_form/input.html'
- ***input_type*** the type attribute on the standard input HTML tag. Default: 'text'
- *is_boolean*
- ***is_list*** interpret request data as a list (can NOT be a callable). Default False
- ***is_valid*** validation function. Should return a tuple of (bool, reason_for_failure_if_bool_is_false) or raise ValidationError. Default: lambda form, field, parsed_data: (True, '')
- *label_container*
- ***label_template*** django template filename for the template for just the label tab. Default: 'tri_form/label.html'
- *model*
- *model_field*
- ***name*** the name of the field. This is the key used to grab the data from the form dictionary (normally request.GET or request.POST)
- ***parse*** parse function. Default just returns the string input unchanged: lambda form, field, string_value: string_value
- *parse_empty_string_as_none*

- *post_validation*
- *raw_data*
- *raw_data_list*
- ***read_from_instance*** callback to retrieve value from edited instance. Invoked with parameters field and instance.
- ***render_value*** render the parsed and validated value into a string. Default just converts to unicode: lambda form, field, value: unicode(value)
- *render_value_list*
- ***required*** if the field is a required field. Default: True
- *show*
- ***strip_input*** runs the input data through standard python .strip() before passing it to the parse function (can NOT be callable). Default: True
- ***template*** django template filename for the entire row. Normally you shouldn't need to override on this level, see input_template, label_template and error_template below. Default: 'tri_form/{style}_form_row.html'
- ***template_string*** You can inline a template string here if it's more convenient than creating a file. Default: None
- ***write_to_instance*** callback to write value to instance. Invoked with parameters field, instance and value.

### Defaults

- ***editable***
  - *True*
- ***endpoint__config***
  - *tri_form.default_endpoint__config*
- ***endpoint__validate***
  - *tri_form.default_endpoint__validate*
- ***errors_template***
  - *tri_form/errors.html*
- ***input_template***
  - *tri_form/input.html*
- ***input_type***
  - *text*
- ***is_boolean***
  - *False*
- ***is_list***
  - *False*
- ***label_container__attrs__class__description_container***
  - *True*
- ***label_template***
  - *tri_form/label.html*

- ***parse_empty_string_as_none***
  - *True*
- ***required***
  - *True*
- ***show***
  - *True*
- ***strip_input***
  - *True*
- ***template***
  - *tri_form/{style}_form_row.html*

### Shortcuts

#### *boolean*

#### *boolean_tristate*

#### *choice*

**Shortcut for single choice field. If required is false it will automatically add an option first with the value '' and the title '—'. To**

**param empty_label** default '—'

**param choices** list of objects

**param choice_to_option** callable with three arguments: form, field, choice. Convert from a choice object to a tuple of (choice, value, label, selected), the last three for the <option> element

#### *choice_queryset*

#### *date*

#### *datetime*

#### *decimal*

#### *email*

#### *file*

#### *float*

#### *foreign_key*

#### *heading*

#### *hidden*

#### *info*

Shortcut to create an info entry.

#### *integer*

#### *many_to_many*

#### *multi_choice*

#### *multi_choice_queryset*

#### *password*

#### *phone_number*

#### *radio*

#### *text*

#### *textarea*

#### *time*

#### *url*

### 7.4.5 Query

Declare a query language. Example:

```
class CarQuery(Query):
    make = Variable.choice(choices=['Toyota', 'Volvo', 'Ford])
    model = Variable()

query_set = Car.objects.filter(CarQuery(request=request).to_q())
```

**type variables** list of Variable :type request: django.http.request.HttpRequest

#### Refinable members

- *endpoint*
- *endpoint_dispatch_prefix*
- *form_class*

- *gui*
- *member_class*

#### Defaults

- ***endpoint__errors***
  - *tri_query.default_endpoint__errors*
- ***endpoint__gui***
  - *tri_query.default_endpoint__gui*
- ***endpoint_dispatch_prefix***
  - *query*

### 7.4.6 Variable

Class that describes a variable that you can search for.

Parameters with the prefix "**gui__**" will be passed along downstream to the tri.form.Field instance if applicable. This can be used to tweak the basic style interface.

#### Refinable members

- *after*
- *attr*
- *choices*
- *extra*
- *freetext*
- *gui*
- *gui_op*
- *model*
- *model_field*
- *name*
- *op_to_q_op*
- *show*
- *value_to_q*
- *value_to_q_lookup*

#### Defaults

- ***gui__required***
  - *False*

- ***gui__show***
  - *False*
- ***gui_op***
  - =
- ***show***
  - *True*

### Shortcuts

#### *boolean*

#### *boolean_tristate*

#### *case_sensitive*

#### *choice*

**Field that has one value out of a set.**

**type choices** list

#### *choice_queryset*

**Field that has one value out of a set.**

**type choices** django.db.models.QuerySet

#### *date*

#### *datetime*

#### *decimal*

#### *email*

#### *float*

#### *foreign_key*

#### *integer*

#### *many_to_many*

#### *multi_choice*

**Field that has one value out of a set.**

**type choices** list

#### *multi_choice_queryset*

#### *text*

#### *time*

#### *url*

### 7.4.7 Link

#### Refinable members

- *attrs*
- *extra*
- *group*
- *show*
- *tag*
- *template*
- *title*

#### Defaults

- ***show***
  - *True*
- ***tag***
  - *a*

#### Shortcuts

#### *button*

#### *icon*

#### *submit*

## 7.5 History

- Remove deprecated use of force_text in django

### 7.5.1 8.5.1 (2020-12-04)

- Removed broken validation of sort columns. This validation prevented sorting on annotations which was very confusing as it worked in dev.
- NOTE: tri.table is a legacy library and is fully replaced by iommi

### 7.5.2 8.5.0 (2020-08-21)

- Include tri.struct 4.x as possible requirement

### 7.5.3 8.4.0 (2020-04-24)

- Fix bulk form missing requests attribute. (Failing on ajax selects)
- Upped dependency tri.declarative to 5.x

### 7.5.4 8.3.0 (2020-01-09)

- Change python version to 3.7

### 7.5.5 8.2.0 (2019-11-21)

- Introduced *data_retrivial_method*, and turned it on by default for *foreign_key* and *many_to_many*. This means that by default tables are now efficient instead of requiring you to use *prefetch_related* or *select_related* manually.
- Added missing *UUIDField* factory
- Added missing *Column.multi_choice*
- *page_size* wasn't refinable

### 7.5.6 8.1.1 (2019-10-23)

- Upped dependency on tri.form due to a bug fix there, and the use of that bug fix in tri.table
- Handle late binding of *request* member of *Table*
- Removed deprecated use of *@creation_ordered*

### 7.5.7 8.1.0 (2019-10-15)

- Implemented *Table.actions* as a replacement for *render_table's argument 'links*.
- *Column.multi_choice_queryset* was broken.
- Fixed *many_to_many* shortcut.
- **Deprecated the following parameters to *render_table*:**
  - *template*: replaced by *Table.template*
  - *paginate_by*: replaced by *Table.page_size*

  - *show_hits*: no replacement
  - *hit_label*: no replacement
  - *page*: no replacement
  - *blank_on_empty*: no replacement
  - *links*: replaced by *Table.actions*
- Bumped dependency tri.declarative to 4.x

### 7.5.8 8.0.0 (2019-06-14)

- Renamed module from *tri.table* to *tri_table*
- Dropped support for python2 and Django < 2.0

### 7.5.9 7.0.2 (2019-05-06)

- Fixed cases where from_model lost the type when inheriting

### 7.5.10 7.0.1 (2019-05-03)

- Fixed a bug where columns that had query or bulk but attr=None would crash

### 7.5.11 7.0.0 (2019-04-12)

- Make *Column* shortcuts compatible with subclassing. The previous fix didn't work all the way.
- Use the new major tri.declarative, and update to follow the new style of class member shortcuts
- Removed support for django 1.8
- *bulk_queryset* is now usable to create your own bulk actions without using *Table.bulk_form*
- Bulk form now auto creates via *Form.from_model* correctly
- Query is now auto created via *Query.from_model* correctly

### 7.5.12 6.3.0 (2019-03-15)

- Make Column shortcuts compatible with subclassing

### 7.5.13 6.2.1 (2019-03-05)

- Fixed a crash when you used a custom paginator in django 2.0+

### 7.5.14 6.2.0 (2019-03-04)

- Fixes for jinja2 compatibility (still not fully working)
- *preprocess_data* now takes a new keyword argument *table*
- You can now get the paginator context itself via *Table.paginator_context*
- Paginator template is configurable
- Fixed a bug where we triggered our own deprecation warning for *Column*
- Use the new paginator API for django 2.0+

### 7.5.15 6.1.0 (2019-01-29)

- Deprecated *Column* argument *attrs* in favor of *header__attrs*
- Added CSS classes *ascending*/*descending* on headers
- Added ability to customize superheaders via *Column.superheader*
- Added ability to customize *Column* header template via *header__template*
- Deprecated *title* parameter to *Column*
- Deprecated *css_class* parameter to *Column*
- Removed class='row{1,2}' from <tr> tags. This is better accomplished with CSS.

### 7.5.16 6.0.3 (2018-12-06)

- Bug fix: "Select all" header button should fire click event, not just toggle the state.

### 7.5.17 6.0.2 (2018-12-06)

- Bug fix: "Select all items" question hidden when select all clicked again.
- Bug fix: only show "Select all item" question if a paginator is present.

### 7.5.18 6.0.1 (2018-12-04)

- Bug fix: "Select all items" question should only be presented once.

### 7.5.19 6.0.0 (2018-12-03)

- Removed argument *pks* to *post_bulk_edit*. This argument is incompatible with non-paginated bulk edit, and it's redundant with the *queryset* argument.
- Added support for bulk editing of an entire queryset, not just the selected items on the current page.
- Fixed bug where the template context was not carried over to the row rendering when using a custom row template.
- Removed *paginator* template tag, moved the functionality into *Table.render_paginator*. This means it can be used from jinja2 and is generally easier to work with.

- Avoid filtering with tri.query if not needed. This means you can now take a slice of a queryset before you pass it to tri.table, if and only if you don't then have filters to apply.
- New feature: refinable attribute *preprocess_data* on *Table*. This is useful if you want to for example display more than one row per result of a queryset or convert the paginated data into a list and do some batch mutation on the items.
- *preprocess_row* returning None is now deprecated. You should now return the row. Just returning the object you were sent is probably what you want.

### 7.5.20 5.3.1 (2018-10-10)

- Added *Column.boolean_tristate* for optionally filter boolean fields.
- Add support for setting namespace on tables to be able to reuse column names between two tables in the same view.
- Removed buggy use of *setdefaults*. This could cause overriding of nested arguments to not take.

### 7.5.21 5.3.0 (2018-08-19)

- Added *preprocess_row* feature. You can use it to mutate a row in place before access.
- Made *Table* a *RefinableObject*

### 7.5.22 5.2.2 (2018-06-29)

- Fix bad mark_safe invocation on custom cell format output.

### 7.5.23 5.2.1 (2018-06-18)

- Fixed bug with backwards compatibility for *Link*.

### 7.5.24 5.2.0 (2018-06-15)

- New feature: default sort ordering. Just pass *default_sort_order* to *Table*.
- *Link* class is now just inherited from tri_form *Link*. Introduced a deprecation warning for the constructor argument *url*.
- Simplified *prepare* handling for *Table*. You should no longer need to care about this for most operations. You will still need to call *prepare* to trigger the parsing of URL parameters for sorting etc.
- Fixed many_to_many_factory

### 7.5.25 5.1.1 (2018-04-09)

- Lazy and memoized BoundCell.value

### 7.5.26 5.1.0 (2018-01-08)

- Fix sorting of columns that contains None, this was not working in Python 3

### 7.5.27 5.0.0 (2017-08-22)

- Moved to tri.declarative 0.35, tri.form 5.0 and tri.query 4.0. Check release notes for tri.form and tri.query for backwards incompatible changes
- Removed deprecated *template_name* parameter to *render_table*
- Note that *foo__class* to specify a constructor/callable is no longer a valid parameter, because of updated tri.form, use *foo__call_target* or just *foo*

### 7.5.28 4.3.1 (2017-05-31)

- Bugfix: sorting on reverse relations didn't work

### 7.5.29 4.3.0 (2017-04-25)

- Bugfix for Django 1.10 template handling
- Updated to tri.form 4.7.1
- Moved bulk button inside the table tag
- Dropped support for Django 1.7

### 7.5.30 4.2.0 (2017-04-21)

- New feature: post bulk edit callback

### 7.5.31 4.1.2 (2017-04-19)

- Fixed silly non-ascii characters in README.rst and also changed to survive silly non-ascii characters in that same file.

### 7.5.32 4.1.1 (2017-04-10)

- Fix missing copy of *attrs__class*

### 7.5.33 4.1.0 (2017-03-22)

- *Column* class now inherits from *object*, making the implementation more pythonic. (Attributes still possible to override in constructor call, see *NamespaceAwareObject*)
- **.template* overrides can now be specified as *django.template.Template* instances.
- The *template_name* parameter to *render_table* is now deprecated and superceeded by a *template* parameter.

### 7.5.34 4.0.0 (2016-09-15)

- Updated to newest tri.form, tri.query, tri.declarative. This gives us simpler factories for *from_model* methods.
- Added shortcuts to *Column*: *time* and *decimal*
- The following shortcuts have been updated to use the corresponding *Variable* shortcuts: date, datetime and email
- Fix failure in endpoint result return on empty payload. *[]* is a valid endpoint dispatch result.
- *render_table*/*render_table_to_response* no longer allow table to be passed as a positional argument

### 7.5.35 3.0.1 (2016-09-06)

- Fix crash on unidentified sort parameter.

### 7.5.36 3.0.0 (2016-09-02)

- *bound_row* is passed to row level callables. This is a potential breaking change if you didn't do **_ at the end of your function signatures (which you should!)
- *bound_row* and *bound_column* is passed to cell level callables. This is a potential breaking change like above.
- *BoundRow* now supports *extra*.
- compatibible with Django 1.9 & 1.10
- Added strict check on the kwargs config namespace of *Table*
- Added *extra* namespace to *Table*
- Added *bound_cell* parameter to rendering of cell templates.

### 7.5.37 2.5.0 (2016-07-14)

- Added optional *endpoint_dispatch_prefix* table configuration to enable multiple tables on the same endpoint.

### 7.5.38 2.4.0 (2016-07-13)

- Made more parts of *BoundCell* available for reuse.

### 7.5.39 2.3.0 (2016-07-12)

- Added pass-through of extra arguments to *Link* objects for custom attributes.

### 7.5.40 2.2.0 (2016-06-23)

- Fix missing namespace collection for column custimization of Table.from_model

### 7.5.41 2.1.0 (2016-06-16)

- Renamed *db_compat.register_field_factory* to the clearer *register_column_factory*
- Improved error reporting on missing django field type column factory declaration.
- Added iteration interface to table to loop over bound rows
- Added *endpoint* meta class parameter to table to enable custom json endpoints

### 7.5.42 2.0.0 (2016-06-02)

- Support for ajax backend
- Dependent tri.form and tri.query libraries have new major versions

### 7.5.43 1.16.0 (2016-04-25)

- Minor bugfix for fields-from-model handling of auto fields

### 7.5.44 1.15.0 (2016-04-21)

- Table.from_model implemented

### 7.5.45 1.14.0 (2016-04-19)

- Added *after* attribute on *Column* to enable custom column ordering (See *tri.declarative.sort_after()*)
- Enable mixing column definitions in both declared fields and class meta.
- Don't show any results if the form is invalid

### 7.5.46 1.13.0 (2016-04-08)

- Add python 3 support

### 7.5.47 1.12.0 (2016-02-29)

- Changed syntax for specifying html attributes and classes. They are now use the same way of addressing as other things, e.g.: Column(attrs__foo="bar", attrs__class__baz=True) will yield something like *<th class="baz" foo=bar>...</th>*

### 7.5.48 1.11.0 (2016-02-04)

- Fix missing evaluation of row__attr et al.

### 7.5.49 1.10.0 (2016-01-28)

- Changed cell__template and row__template semantics slightly to enable customized cell ordering in templates. row__template implementations can now access a BoundCell object to use the default cell rendering. cell__template implementation are now assumed to render the <td> tags themself.

### 7.5.50 1.9.0 (2016-01-19)

- Fixed to work with latest version of tri.form

## 7.6 Credits

- Anders Hovmöller <anders.hovmoller@trioptima.com>
- Johan Lübcke <johan.lubcke@trioptima.com>

## 7.7 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. Issues, feature requests, etc are handled on github.

CHAPTER 8

# Indices and tables

- genindex
- modindex
- search