Pyforms Web Documentation

Release 4.0

Ricardo Jorge Vieira Ribeiro

Jan 20, 2023

PYFORMS WEB

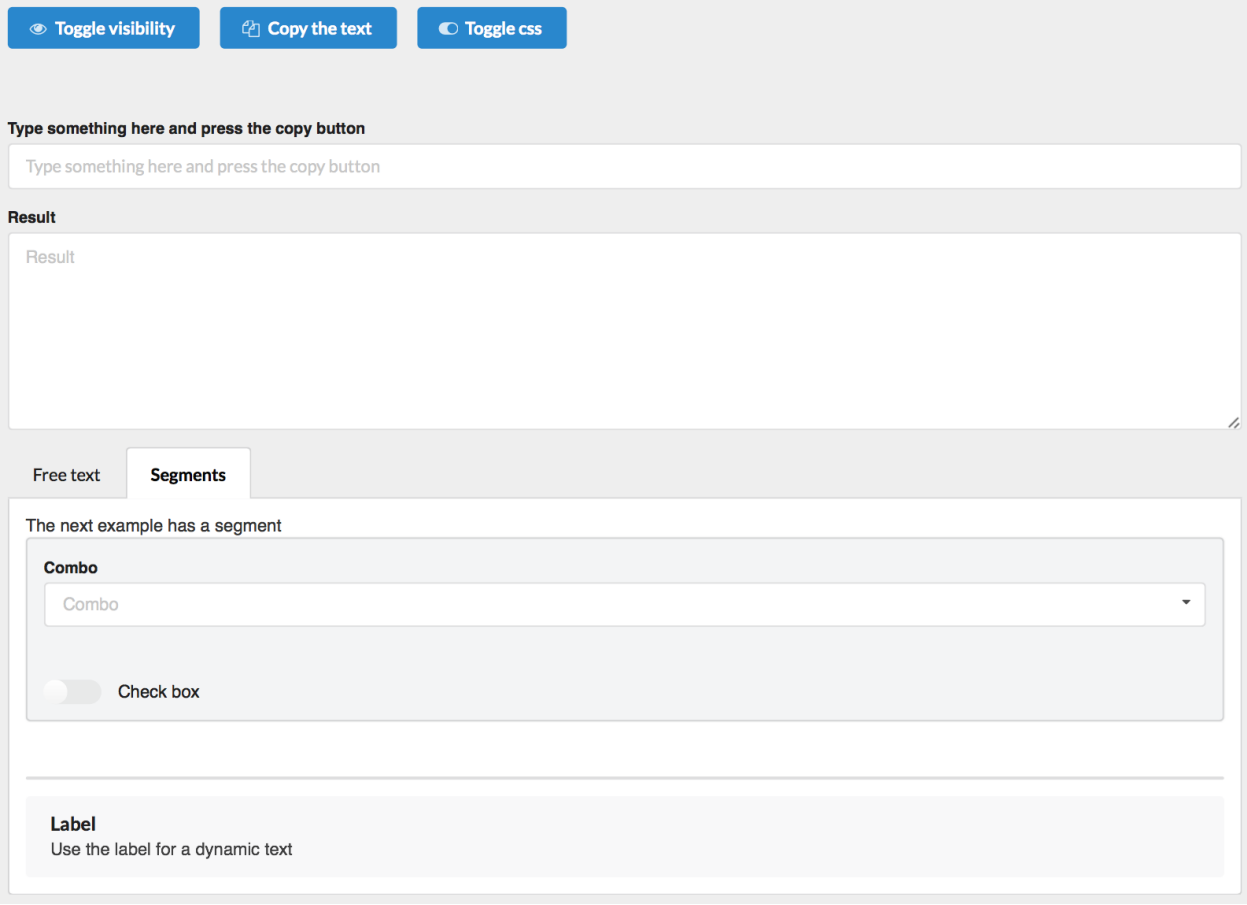
Pyforms Web is Python 3 framework to create single-page web applications.

The framework aims the boost the development productivity by providing an API in Python that generates HTML Forms amd manages all the clientserver forms communications.

Developers do not require to write any HTML or Javascript code, all the code is done in Python 3.

Source code https://github.com/UmSenhorQualquer/pyforms-web

The next code produces the next web application.




```
class DefaultApp(BaseWidget):

  TITLE = 'Demo app'

  def __init__(self, *args, **kwargs):
    super(DefaultApp, self).__init__(*args, **kwargs)

    self._css_btn    = ControlButton('Toggle css')
    self._toggle_btn = ControlButton('Toggle visibility')
    self._copy_btn   = ControlButton('Copy the text')
    self._input      = ControlText('Type something here and press the copy button')
    self._text       = ControlTextArea('Result')
    self._combo      = ControlCombo('Combo')
    self._check      = ControlCheckBox('Check box')
    self._list       = ControlList('List')
```

(continues on next page)

(continued from previous page)

```
    self._label      = ControlLabel('Label', default='Use the label for a dynamic text')

    self.formset = [
      no_columns('_toggle_btn','_copy_btn', '_css_btn'),
      ' ',
      '_input',
      '_text',
      {
        'Free text': [],
        'Segments': [
          'The next example has a segment',
          segment(
            '_combo',
            '_check',
            css='secondary'
          ),
          '_list',
          '_label'
        ]
      }
    ]
```

Note: This framework is a software layer part of the Pyforms framework.

Pyforms https://pyforms.readthedocs.io

CHAPTER

ONE

OVERVIEW

1.1 Pyforms Web

Pyforms Web is part the Pyforms framework. It implements the web layer to manage the communication between the Python applications and the client browser's.

With **Pyforms Web** you can create single-page applications (SPA) in Python quickly and without writing any HTML or Javascript code.

1.2 Pyforms

Pyforms is a Python 3 framework to develop applications capable of executing in 3 diferent environments, Desktop GUI, Terminal and Web.

1.3 Orquestra

Orquestra implements a layout environment to accommodate Pyforms applications. The examples shown on this documentation are based on the orquestra environment, but one can be replaced by any other layout environment.

Note: Please **star** the project at the Github repository to support the project.

CHAPTER

TWO

QUICK START

- Install Pyforms using **pip**.

```
pip install pyforms-web
```

- Clone **pyforms-web** repository.

```
git clone https://github.com/UmSenhorQualquer/pyforms-web.git
```

- Access the **server** directory inside the **pyforms-web** local repository.

```
cd pyforms-web/server
```

- Initiate the django development server.

```
python manage.py migrate
python manage.py runserver
```

CHAPTER

THREE

INSTALL & CONFIGURE

On this page it is explained how to configure your environment and your django app to start using pyforms.

Note: The instructions on this page assumes you know how the Django framework works.

3.1 Configure the environment

- Install Pyforms using **pip**.

```
pip install pyforms-web
```

3.2 Configure a django project

Execute the next command in the terminal to start a Django project.

```
django-admin startproject <project-name>
```

Edit the django project **settings.py** file to include the next configurations.

```
INSTALLED_APPS = [
    'orquestra',
    'sorl.thumbnail',
    'pyforms_web.web',
    'jfu',
    'sorl.thumbnail',
    ...
]

MIDDLEWARE = [
    ...
    'pyforms_web.web.middleware.PyFormsMiddleware'
]
```

Edit the django project **urls.py** file to include the next urls configurations.

```
from django.conf    import settings
from django.contrib import admin
from django.urls    import include, path

urlpatterns = [
    path('pyforms/',  include('pyforms_web.web.urls') ),
    path('',          include('orquestra.urls')       ),
]

if settings.DEBUG:
    from django.conf.urls.static import static
    urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

3.3 Create a pyforms settings file

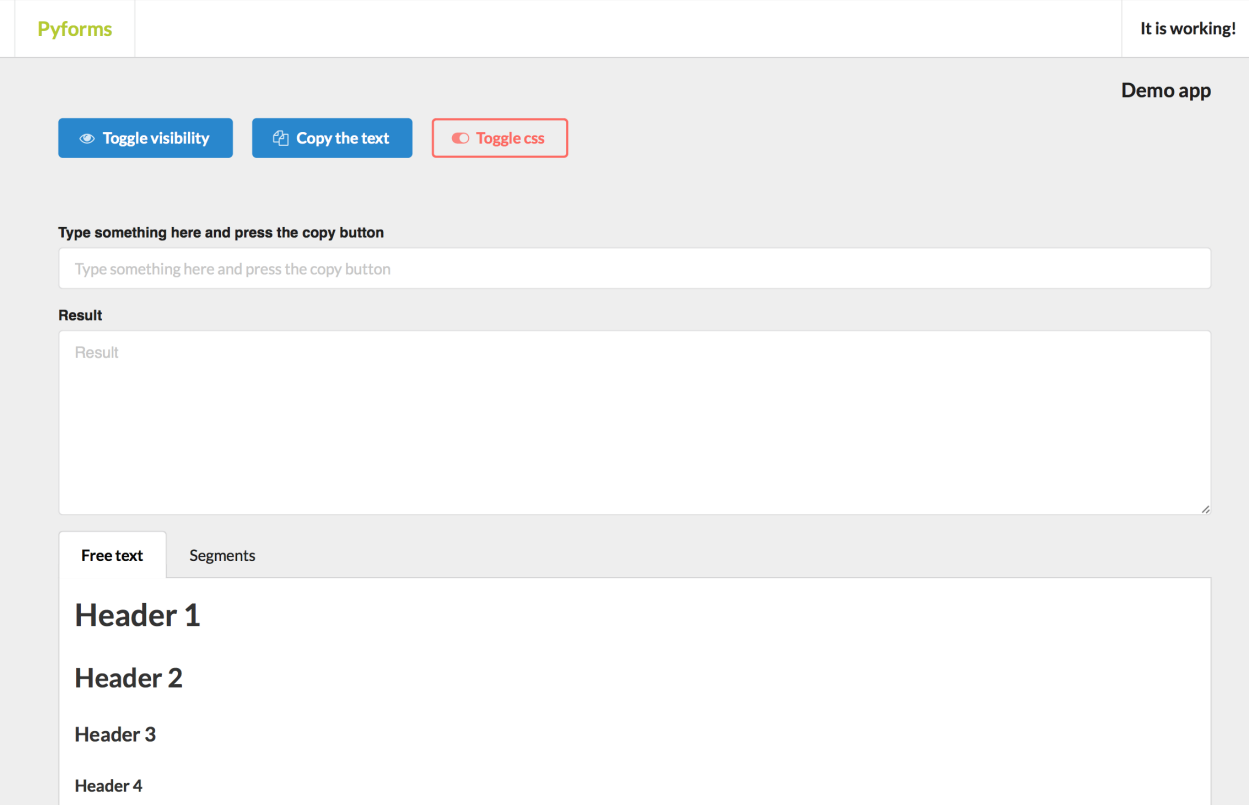
In the django project root folder (same folder of the manage.py file) create the local_settings.py file with the next content.

```
SETTINGS_PRIORITY = 0 # Will define this settings file as priority. Will override all␣
↪the settings with lower priority.
PYFORMS_MODE = 'WEB' # Will configure pyforms to run as WEB mode.
```

3.4 Run the project

```
cd <project-name>
python3 manage.py migrate
python3 manage.py runserver
```

Access to http://localhost:8000



3.5 Configure pyforms-web

Use the **local_settings.py** file to set some pyforms-web settings.

```
# Do not use the compressed JS file, and use the original files
PYFORMS_DEBUG = True

# Activate verbose mode for the
PYFORMS_VERBOSE = True
```

(optional)

3.6 Configure django-allauth

django-allauth is a reusable Django app that allows for both local and social authentication, with flows that just work.

To make it work with Orquestra follow the instructions described @django-allauth documentation.

3.6.1 Typical django-allauth configuration

```
pip install django-allauth
```

Add the next configuration to your Django project setttings.

```
...

INSTALLED_APPS = [
    ...
    'allauth',
    'allauth.account',
    'allauth.socialaccount',
    'allauth.socialaccount.providers.google',
    'django.contrib.sites',
    ...
]

AUTHENTICATION_BACKENDS = [
    'django.contrib.auth.backends.ModelBackend',
    'allauth.account.auth_backends.AuthenticationBackend',
]

LOGIN_URL = '/accounts/login/'
LOGIN_REDIRECT_URL = '/'
```

Edit the urls.py file to add the next code.

```
from django.urls import path

urlpatterns = [
    path('accounts/', include('allauth.urls')),
    ...
]
```

Add the next configuration to the **local_settings.py** file to configure **orquestra** to require always authentication before accessing the applications.

```
ORQUESTRA_REQUIREAUTH = True
```

Note: Do not forget to apply the db migrations to your project.

Initialize allauth configuration by adding a new register to the sites.Site model, using the next command.

```
python manage.py loaddata conf-allauth-site
```

CHAPTER

FOUR

FIRST APPLICATION

Note: More documentation to read about this example at:

- *pyforms_web.basewidget.BaseWidget*
- *pyforms_web.controls.control_base.ControlBase*

Here it is shown how to create the first pyforms app for a django project.

Note: The instructions on this page assumes you know how the Django framework works.

4.1 Prepare the django app

Create a empty django app folder with the next directory structure inside:

```
my_module_name
├── pyforms_apps
│   └── __init__.py
└── __init__.py
```

Add the application to the settings.py

```
INSTALLED_APPS = [
    'my_module_name',
    ...
]
```

4.2 Create the first app

Create the file **my_module_name/pyforms_apps/site_crawl.py** and add the next code to it.

```
from pyforms.basewidget import BaseWidget
from confapp import conf

class SiteCrawlApp(BaseWidget):

```

(continues on next page)

(continued from previous page)

```
    UID                 = 'site-crawl-app'
    TITLE               = 'Site crawl'

    LAYOUT_POSITION     = conf.ORQUESTRA_HOME

    ORQUESTRA_MENU      = 'left'
    ORQUESTRA_MENU_ICON  = 'browser'
    ORQUESTRA_MENU_ORDER = 0
```

In the **my_module_name/pyforms_apps/__init__.py** add the next code:

```
from .site_crawl import SiteCrawlApp
```

You have created the most basic application. Access to http://localhost:8000 using your browser and visualize it.

Now update the **SiteCrawlApp** application with the next code:

```
from pyforms.basewidget import BaseWidget
from confapp import conf

from pyforms.controls import ControlButton
from pyforms.controls import ControlText
from pyforms.controls import ControlList

class SiteCrawlApp(BaseWidget):

    UID                 = 'site-crawl-app'
    TITLE               = 'Site crawl'

    LAYOUT_POSITION     = conf.ORQUESTRA_HOME

    ORQUESTRA_MENU      = 'left'
    ORQUESTRA_MENU_ICON  = 'browser'
    ORQUESTRA_MENU_ORDER = 0


    def __init__(self, *args, **kwargs):

        super(SiteCrawlApp, self).__init__(*args, **kwargs)

        self._url          = ControlText('Page url')
        self._getlinks_btn = ControlButton('Get links', default=self.___getlinks_btn_evt,
↪ label_visible=False)

        self._links_list   = ControlList('Links list', horizontal_headers=['Found links
↪'])

        self.formset = ['_url', '_getlinks_btn', '_links_list']


    def ___getlinks_btn_evt(self):

        self._links_list.value = [
```

(continues on next page)

(continued from previous page)

```
            ['Link1'],
            ['Link2']
        ]
```

Restart your django project to visualize the updates.

Press the button to see what happens.

CHAPTER

FIVE

DATABASE WIDGETS

These examples shows how to build quick forms to manage Django models.

Note: Not all the configurations are explained on this page. Please take a look to the classes bellow to find more information.

- *pyforms_web.widgets.django.modeladmin.ModelAdminWidget*
- *pyforms_web.widgets.django.modelform.ModelFormWidget*
- *pyforms_web.widgets.django.modelviewform.ModelViewFormWidget*

5.1 Prepare the django app

Create a empty django app folder with the next directory structure inside:

```
my_dbmodule_name
├── pyforms_apps
│   └── __init__.py
├── models.py
└── __init__.py
```

Add the next code to the *models.py* file.

```
from django.db import models

class Supplier(models.Model):
    supplier_id = models.AutoField(primary_key=True)  #: Pk ID
    supplier_name = models.CharField('Name', max_length=200) #: Name
    supplier_nif = models.CharField('NIF Number',max_length=50, blank=True, null=True)
↪#: NIF number
    supplier_keywords = models.CharField('Keywords', max_length=200, blank=True,␣
↪null=True) #: Free text Keywords
    supplier_contact = models.CharField('Contact Person', max_length=200, blank=True,␣
↪null=True) #: The mani contact person of a supplier
    supplier_phone = models.CharField('Phone Number', max_length=200, blank=True,␣
↪null=True) #: Phone number of a supplier
    supplier_mail = models.CharField('Email', max_length=200, blank=True, null=True) #:␣
↪Email address of a supplier
    supplier_discounts = models.CharField('Discounts', max_length=200, blank=True,␣
↪null=True) #: Free text Discounts
```

(continues on next page)

(continued from previous page)

```
    country = models.ForeignKey('Country', blank=True, null=True, on_delete=models.
↪CASCADE) #: Fk Supplier's operation country
    category = models.ManyToManyField('Catproduct', blank=True)  #: category of the␣
↪supplied product


def __str__(self):
    return self.supplier_name
```

Add the application to the settings.py

```
INSTALLED_APPS = [
    'my_dbmodule_name',
    ...
]
```

Commit the new model to the database.

```
python manage.py migrate
```

5.2 Model admin

To create an app to manage a Django model inherit your App from the ModelAdminWidget class.

Create the file **my_dbmodule_name/pyforms_apps/post_app.py** and add the next code to it.

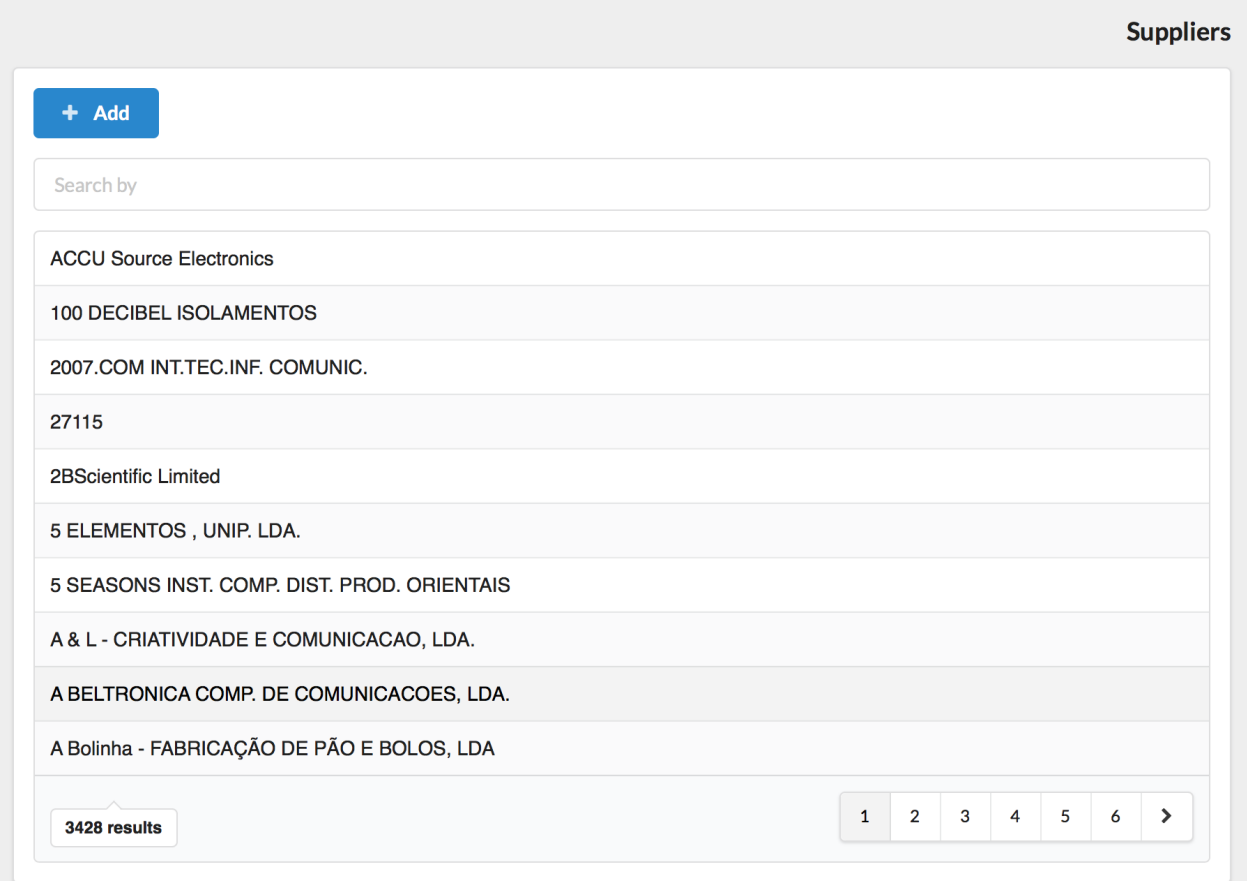
```
from confapp import conf
from pyforms_web.widgets.django import ModelAdminWidget
from pyforms_web.organizers import segment

from my_dbmodule_name.models import Supplier

class SupplierApp(ModelAdminWidget):

    # Default id of the application.
    # Optional when the app is not to be shown in the menu.
    # When used on the menu the UID will used to generate
    # the url for the app: http://[server]/app/[UID]/
    UID = 'supplier-app'

    # Django model to manage.
    MODEL = Supplier

    # Title of the application.
    TITLE = 'Supplier app'

    # Position on HTML where the app should run.
    LAYOUT_POSITION = conf.ORQUESTRA_HOME

    # Optional: used to add the app to the menu.
```

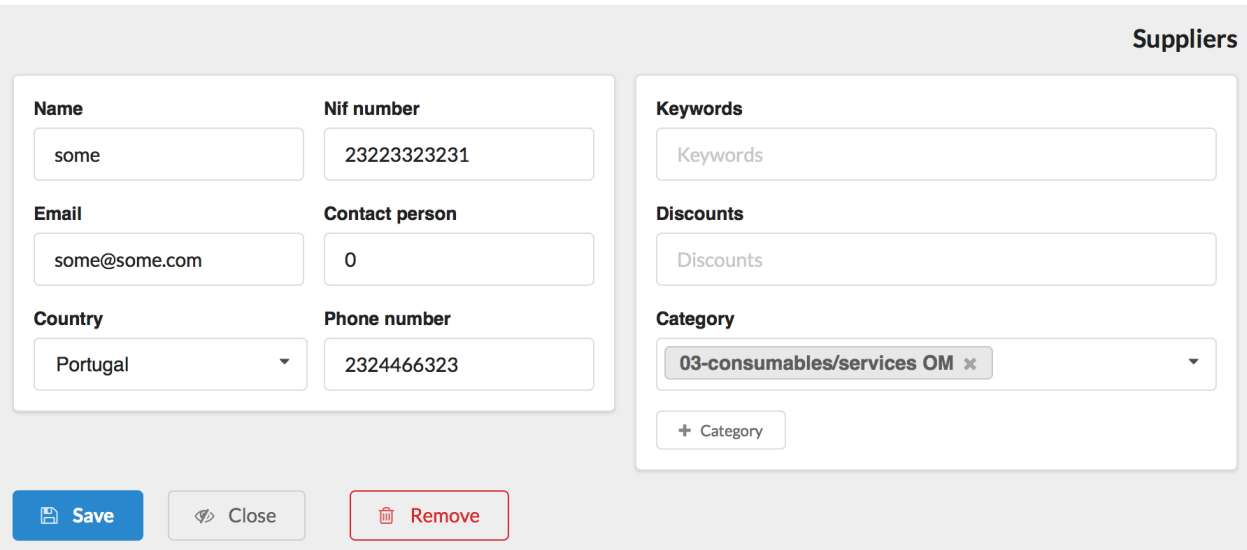
(continues on next page)

(continued from previous page)

```
    # Orquestra environment configuration.
    ORQUESTRA_MENU = 'left'        # Add the application to the left menu.
    ORQUESTRA_MENU_ICON = 'users'  # Icon menu.
    ORQUESTRA_MENU_ORDER = 0       # Order where the application should be shown in the␣
↪menu.
    ######################################################

    # Optional: django model fields to be used in the search field.
    SEARCH_FIELDS = ['supplier_name__icontains']

    # Optional: used to organize the Controls for the model edition form.
    FIELDSETS = [
        (
            segment(
                ('supplier_name','supplier_nif'),
                ('supplier_mail','supplier_contact'),
                ('country','supplier_phone'),
                ('_category','_addcategory_btn')
            ),
            segment(
                'supplier_keywords',
                'supplier_discounts',
                'category',
            )
        )
    ]

    ...
```

Access to http://localhost:8000 using your browser and visualize your application.





Note: You set the apps that will work as edition or creation forms, by using the variables **ModelAdminWidget.EDITFORM_CLASS** and **ModelAdminWidget.ADDFORM_CLASS**.

```
...

class SupplierApp(ModelAdminWidget):
```

(continues on next page)

(continued from previous page)

```
    ...

    EDITFORM_CLASS  = ...
    ADDFORM_CLASS   = ...

...
```

5.3 Model form

To create an app to edit a Django model register, inherit from the ModelFormWidget class.

Add to the beginning of the file **my_dbmodule_name/apps/post_app.py** the next code.

```
from confapp import conf
from pyforms_web.widgets.django import ModelFormWidget
from pyforms_web.organizers import segment

from my_dbmodule_name.models import Supplier

class SupplierFormAdmin(ModelFormWidget):

    MODEL = Supplier  # Model to manage.
    TITLE = 'Suppliers' # Title of the application.

    #formset of the edit form
    FIELDSETS = [
        (
            segment(
                ('supplier_name','supplier_nif'),
                ('supplier_mail','supplier_contact'),
                ('country','supplier_phone'),
            ),
            segment(
                'supplier_keywords',
                'supplier_discounts',
                'category',
                ('_category','_addcategory_btn')
            )
        )
    ]

    ...
```

To call the app to edit a register use the parameter **pk**.

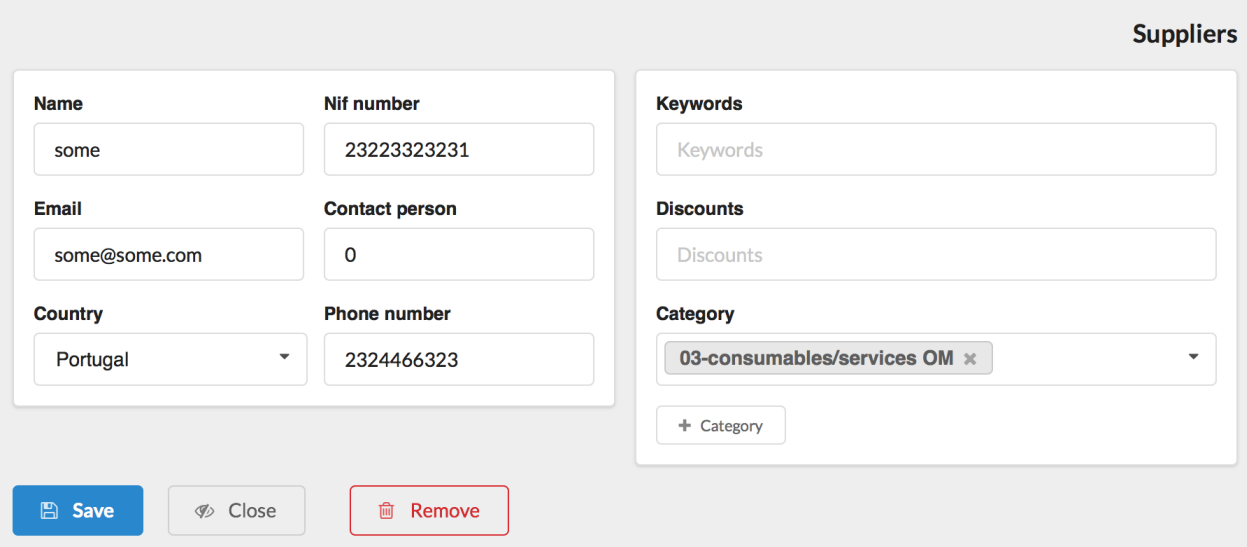
```
obj = SupplierFormAdmin(pk=1)

# or to create an empty register.
```

(continues on next page)

(continued from previous page)

```
obj = SupplierFormAdmin()
```

Access to http://localhost:8000 using your browser and visualize your application.



5.4 Model view form

Use the ModelViewFormWidget widget to create a view only form.

```
from confapp import conf
from pyforms_web.widgets.django import ModelViewFormWidget
from pyforms_web.organizers import segment

from my_dbmodule_name.models import Supplier

class SupplierViewFormAdmin(ModelViewFormWidget):

    MODEL = Supplier  # Model to manage.
    TITLE = 'Suppliers' # Title of the application.

    #formset of the edit form
    FIELDSETS = [
        (
            segment(
                ('supplier_name','supplier_nif'),
                ('supplier_mail','supplier_contact'),
                ('country','supplier_phone'),
            ),
            segment(
                'supplier_keywords',
                'supplier_discounts',
                'category',
```

(continues on next page)

(continued from previous page)

```
                ('_category','_addcategory_btn')
            )
        )
    ]

    ...
```

5.5 Object access permissions

It is possible to restrict the objects a user has access in the widgets above using the Models Queryset manager. The idea here is to define the access rules in the Model side, instead of defining the rules in the Visualization side. These way the Model can be ported from application to application maintaining the access rules.

Example:

```
from django.db import models

class OrderQuerySet(models.QuerySet):
    """
    ORDER QUERYSET MANAGER DEFINITION
    """

    def list_permissions(self, user):
        """
        The function filters the queryset to return only the objects the user has␣
↪permissions to list.
        """
        ...
        return self

    def has_add_permissions(self, user):
        """
        The function returns a Boolean indicating if the user can add or not a new␣
↪object.
        """
        ...
        return True

    def has_view_permissions(self, user):
        """
        The function returns a boolean indicating if the user has view permissions to␣
↪the current queryset.
        """
        ...
        return self

    def has_update_permissions(self, user):
        """
        The function filters the queryset to return only the objects the user has␣
↪permissions to update.
```

(continues on next page)

(continued from previous page)

```
        """
        ...
        return self

    def has_remove_permissions(self, user):
        """
        The function filters the queryset to return only the objects the user has␣
↪permissions to remove.
        """
        ...
        return self



class Order(models.Model):
    """
    MODEL DEFINITION
    """
    ...

    objects = OrderQuerySet.as_manager()
```

CHAPTER

SIX

WIDGET FORMSET

On this page it is shown examples of Widgets formsets configurations.

Example:

```
[
     no_columns('_toggle_btn','_copy_btn', '_css_btn'),
     ' ',
     ('empty:twelve','_input'),
     '_text',
     {
         'a:Free text': [
             'h1:Header 1',
             'h2:Header 2',
             'h3:Header 3',
             'h4:Header 4',
             'h5:Header 5',
             'h1-right:Header 1',
             'h2-right:Header 2',
             'h3-right:Header 3',
             'h4-right:Header 4',
             'h5-right:Header 5',
             '-',
             'Free text here',
             'msg:Message text',
             'info:Info message',
             'warning:Warning message',
             'alert:Alert message'
         ],
         'b:Segments': [
             'The next example has a segment',
             segment(
                 '_combo',
                 '_check',
                 css='secondary'
             ),
             '_list',
             '_label'
         ]
     },
     'notifications-area'
]
```

6.1 Configuration options

6.1.1 tuple

Displays the controls horizontally.

6.1.2 list

Displays the controls vertically.

6.1.3 dict

Displays the controls in Tabs. Use [a:,b:,c:] prefix to sort the tabs.

6.1.4 '-'

Draw a vertical line.

6.1.5 Empty column

Use ' ', or the prefix 'empty:' + size of the column (ex: one, two, ..., sixteen) to add a empty column.

```
...
('empty:twelve', ...),
...
```

6.1.6 segment

Wraps the formset around a segment (Semantic UI segment).

- Use the parameter **css**, to add extra CSS classes to the segment.
- Use the parameter **field_css**, to add CSS classes to the div.field containing the segment.

```
segment(
     ('person_first', 'person_middle', 'person_last'),
     ('person_gender', 'person_birthday'),
     ('degree', 'scientificarea'),
     ('person_cv', 'person_web'),
     'person_bio',
     css='inverted',
     field_css='fourteen wide',
 ),
```

6.1.7 no_columns

Do not apply the fields columns alignments.

6.1.8 Free text

Do not apply the fields columns alignments.

6.1.9 Message

By using the prefixes [msg:,info:,warning:,alert:] you will wrap a free message on message box.

6.1.10 Headers

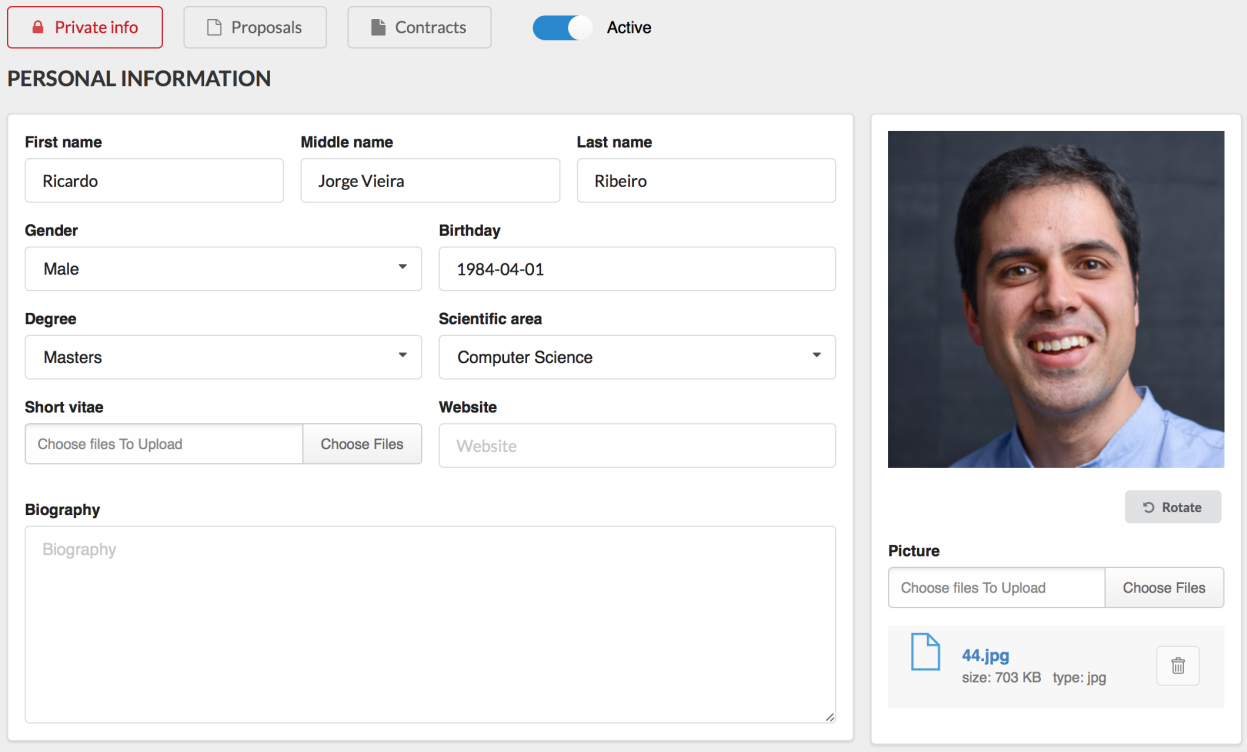
Use the prefixes [h1:,h2:,h3:,h4:,h5:,h1-right:,h2-right:,h3-right:,h4-right:,h5-right:] on free text.

```
[
    ... ,
    'h3:PERSONAL INFORMATION',
    ...
]
```

6.1.11 Notifications area

By default the application's messages are shown on the top of each application, but it is possible to define a new position to these messages by including the code 'notifications-area' on the formset.

6.2 2 segments side by side




```
class PeopleFormWidget(BaseWidget):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        ... # fields definition

        # Use the field_style to align the checkbox to the middle.

        self.person_active.field_style = 'text-align: right;margin-top:5px;'
        self.person_active.field_css   = 'two wide'

        self._rotimg_btn = ControlButton(
            '<i class="icon undo" ></i>Rotate',
            default=self.__rotimg_evt,
            label_visible=False,   # Remove all the labels from the fields to make the␣
↪row thinner.
            style='margin-top:5px;',
            field_style='text-align: right;',
            css='mini'
        )

       ...

       self.formset = [
            no_columns('_privateinfo_btn', '_proposals_btn', '_contracts_btn', 'person_
↪active'),
```

(continues on next page)

(continued from previous page)

```
            'h3:PERSONAL INFORMATION',
            ( # Use the tupple to display the segments side by side.
                segment(
                    ('person_first', 'person_middle', 'person_last'),
                    ('person_gender', 'person_birthday'),
                    ('degree', 'scientificarea'),
                    ('person_cv', 'person_web'),
                    'person_bio',
                    field_css='fourteen wide', # Use the argument to resize the segment␣
↪on the left.
                ),
                segment(
                    '_img',
                    '_rotimg_btn',
                    'person_img',
                    field_style='max-width:330px;' # Use the argument to define the␣
↪style of the div.field wrapping the Control.
                )
            ),
        ]
```

6.3 Customize fields css classes and styles

Use the constructor parameters **field_css**, **field_style**, **css**, and **style** to customize the visualization of each field.

Example:

```
self._field_example = ControlText(
    ...,
    style='margin-top:5px;',            # Extra style to add to the control.
    css='mini'                          # Extra css classes to add to the control.
    field_style='text-align: right;',   # Extra style to add to the field div that␣
↪encapsulates the control.
    field_css='mini'                    # Extra css classes to add to the field dive␣
↪that encapsulates the control.
)
```

Note: Check the PeopleFormWidget example above for more examples.

CHAPTER

SEVEN

APPS PERMISSIONS

Note: Checkout the *pyforms_web.basewidget.BaseWidget* class for more information.

For some web applications there is the need of implementing several users profiles, with different access to functionalities and data.

On this page we are going to show how the access to Pyforms applications can be restricted.

7.1 AUTHORIZED_GROUPS variable

This BaseWidget variable can be set with the list of Django groups that has access to the app.

Example:

```
...

class EmployeesAdminApp(ModelAdminWidget):

    ...

    AUTHORIZED_GROUPS = ['superuser', 'django-group']

    ...
```

Note: In the case you want to restrict the access to a **superuser** user then instead of using the name of a Django group you should use the label *'superuser'*.

7.2 Overwrite the BaseWidget functions

The second way of restricting the access to an app is by overwriting the permissions functions from the BaseWidget class.

Note: By overwriting the permissions' methods of the BaseWidget class your are overwriting the functionality of the **AUTHORIZED_GROUPS** variable.

Example:

```
...

class EmployeesAdminApp(ModelAdminWidget):

    ...

    @classmethod
    def has_permissions(cls, user):
        """
        This class method, verifies if a user has permissions to execute the application
        """
        return True or False

    def has_session_permissions(self, user):
        """
        It verifies if a user has permissions to execute the application during the␣
↪runtime.
        """
        return True or False

    ...
```

CHAPTER

EIGHT

PLAY WITH APPS INHERITANCE

Here it will be shown how flexible Pyforms is, and how this flexible can be used to improve productivity and design Applications.

Note: The examples shown here are based on the folder tutorials/tutorial

8.1 Example apps

Consider the next 2 apps

8.1.1 App one

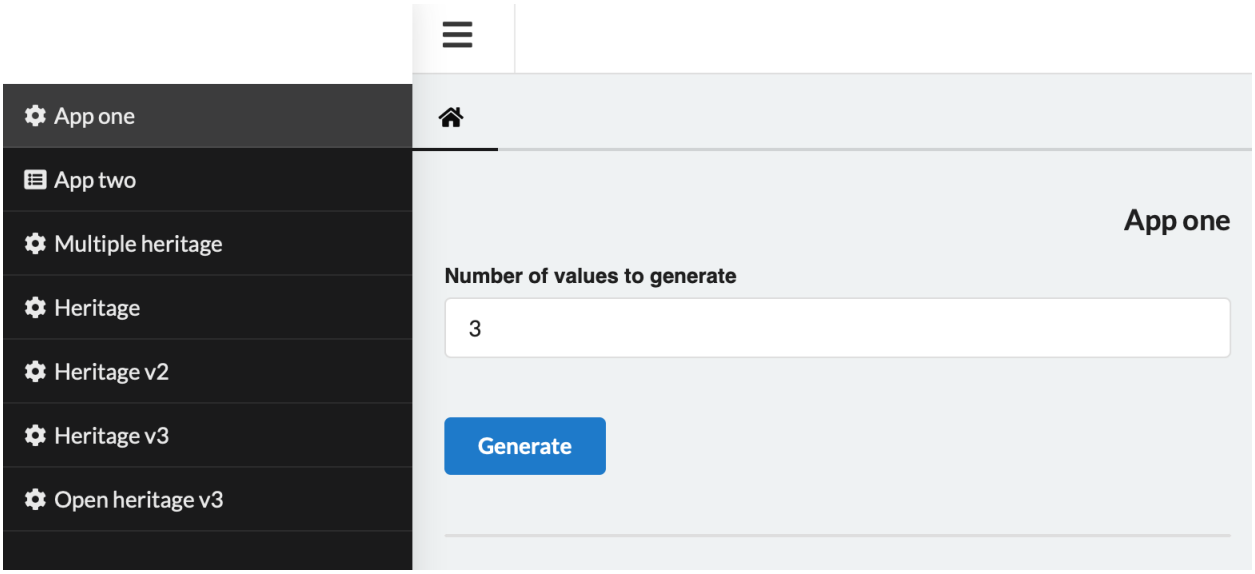
This app has a button that generates a list of random numbers defined in a text control, and display them on a table.

```
import random
from confapp import conf
from decimal import Decimal
from pyforms.basewidget import BaseWidget
from pyforms.controls import ControlDecimal
from pyforms.controls import ControlButton
from pyforms.controls import ControlList

class AppOne(BaseWidget):

    UID                 = 'app-one'
    TITLE               = 'App one'

    LAYOUT_POSITION     = conf.ORQUESTRA_HOME
    ORQUESTRA_MENU      = 'left'
    ORQUESTRA_MENU_ICON = 'cog'
    ORQUESTRA_MENU_ORDER = 0

    def __init__(self, *args, **kwargs):
        self._table = ControlList('Values')

        super().__init__(*args, **kwargs)
```

(continues on next page)

(continued from previous page)

```
        self._ndata = ControlDecimal('Number of values to generate', default=Decimal(3))
        self._btn   = ControlButton('Generate', default=self.btn_evt)

        self.formset = ['_ndata', '_btn', '_table']

    def generate(self):
        data = []
        n = int(self._ndata.value)
        for i in range(n):
            data.append( (str(i), random.randint(0,1000)) )
        return data

    def btn_evt(self):
        self._table.value = self.generate()
```



8.1.2 App two

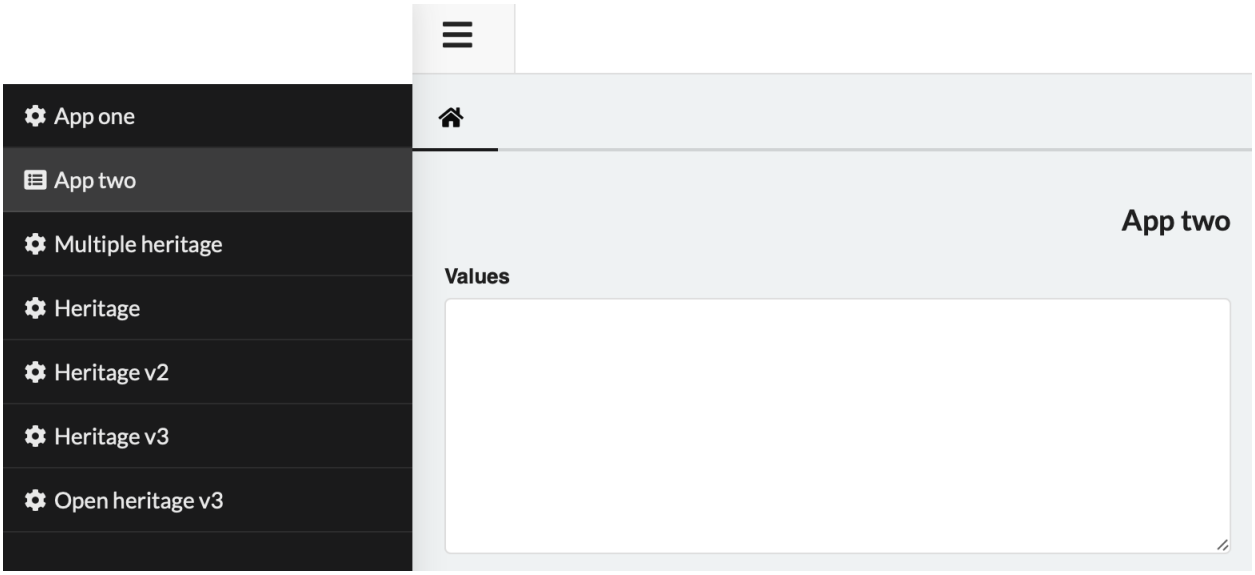
This app does nothing, only includes a text field.

```
from pyforms.basewidget import BaseWidget
from confapp import conf
from pyforms.controls import ControlTextArea

class AppTwo(BaseWidget):

    UID                 = 'app-two'
    TITLE               = 'App two'

    LAYOUT_POSITION     = conf.ORQUESTRA_HOME

    ORQUESTRA_MENU      = 'left'
    ORQUESTRA_MENU_ICON = 'browser'
```

(continues on next page)

(continued from previous page)

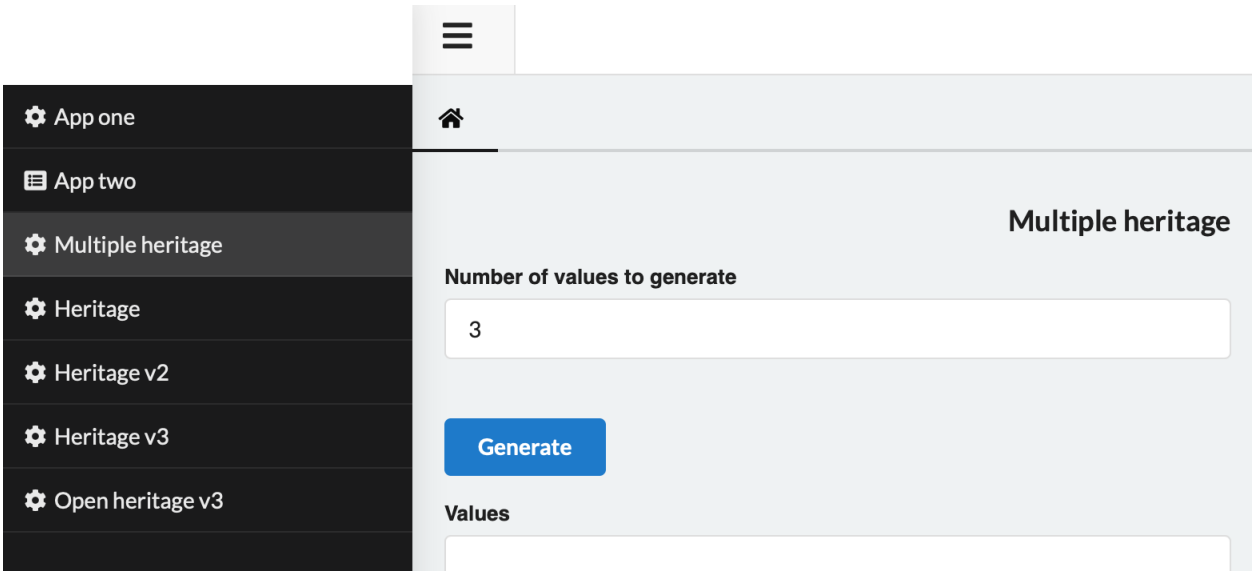
```
    ORQUESTRA_MENU_ORDER = 1

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self._table = ControlTextArea('Values')

        self.formset = ['_table']
```



8.2 Multiple inheritance

Based on the 2 apps above we can combine them on a 3nd app using multiple inheritance. This new app will display the random numbers on a text area, instead of a table.

```
from .app_one import AppOne
from .app_two import AppTwo

class MultipleInheritance(AppOne, AppTwo):

    UID   = 'multiple-inheritance'
    TITLE = 'Multiple inheritance'

    ORQUESTRA_MENU_ORDER = 1

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
```

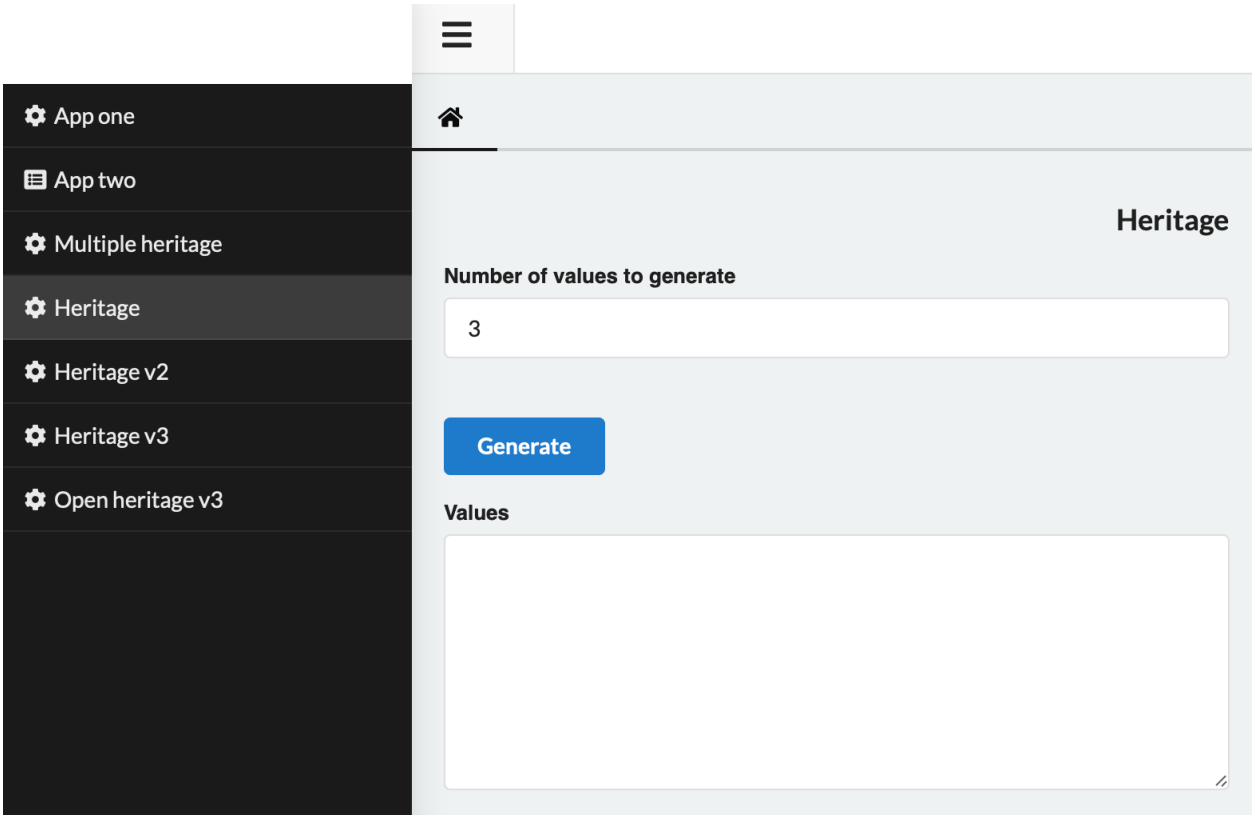


8.3 Inheritance

8.3.1 Example 1

The same of the multiple inheritance example can be done using a simple inheritance.

```
from pyforms.controls import ControlTextArea
from .app_one import AppOne

class Inheritance(AppOne):

    UID   = 'inheritance'
    TITLE = 'inheritance'

    ORQUESTRA_MENU_ORDER = 4

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self._table = ControlTextArea('Values')
```



8.3.2 Example 2

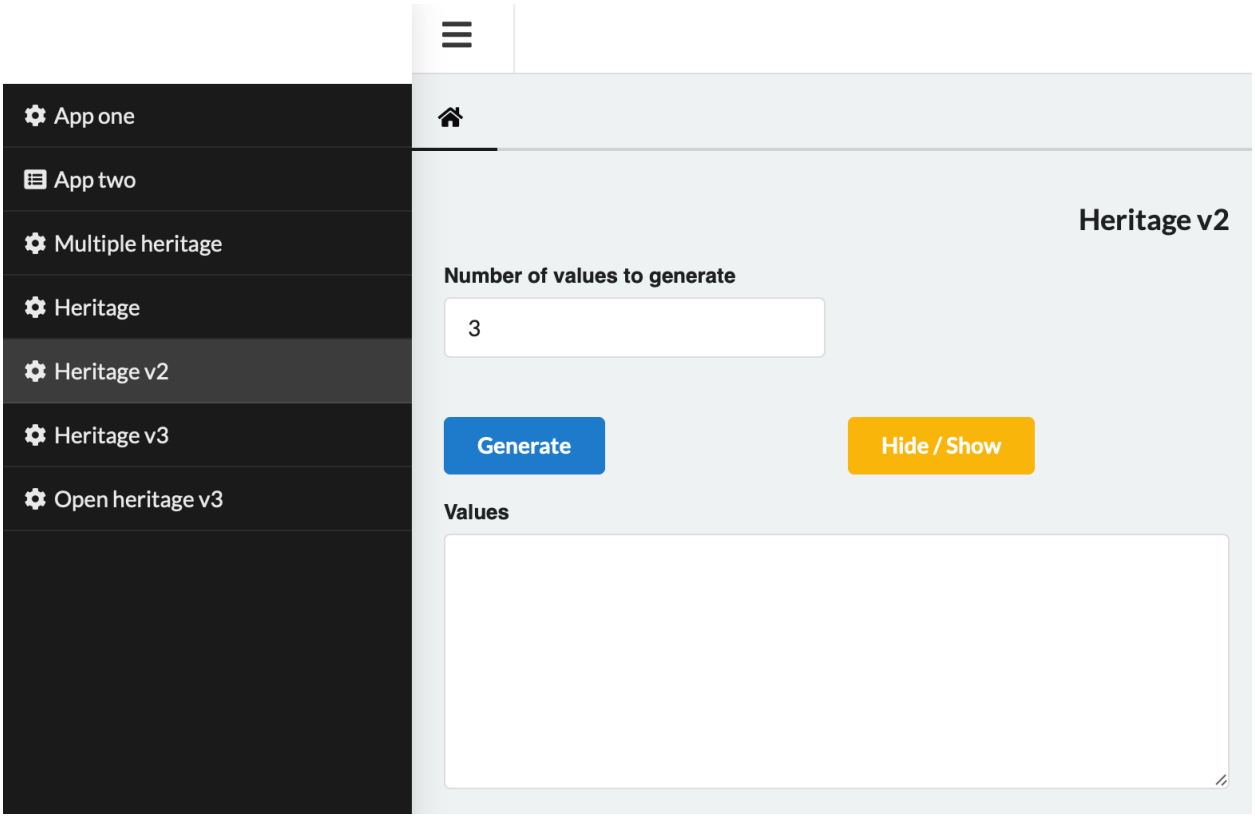
Below the app inherits from the example above and an extra button to hide or show the text field.

```
from pyforms.controls import ControlButton
from .inheritance import Inheritance

class InheritanceV2(Heritage):

    UID                 = 'inheritance-v2'
    TITLE               = 'Inheritance v2'

    ORQUESTRA_MENU_ORDER = 5

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self._togglebtn = ControlButton('Hide / Show', default=self.__togglebnt_evt, css=
↪'yellow')

        self.formset = [
            ('_ndata', ' '),
            ('_btn', '_togglebtn'),
            '_table'
        ]
```

(continues on next page)

(continued from previous page)

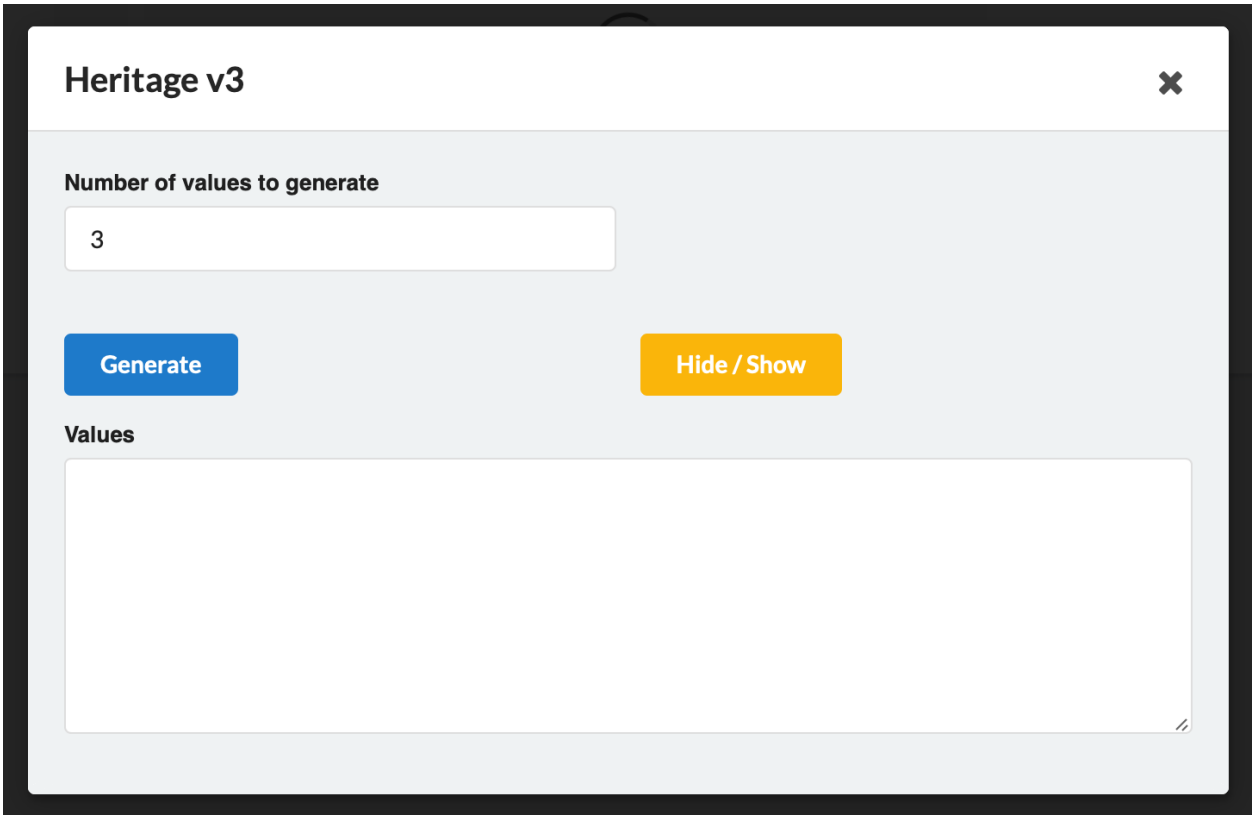
```
    def __togglebnt_evt(self):
        if self._table.visible:
            self._table.hide()
        else:
            self._table.show()
```



8.3.3 Example 3

This app inherits all the functionalities of the app above, but it is configured to open on a popup window.

```
from .inheritance_v2 import HeritageV2
from confapp import conf

class InheritanceV3(InheritanceV2):

    UID = 'inheritance-v3'
    TITLE = 'Inheritance v3'

    ORQUESTRA_MENU_ORDER = 6

    LAYOUT_POSITION = conf.ORQUESTRA_NEW_WINDOW
```



8.3.4 Example 4

This app opens, is exemplifies how the app above can be executed after the press of a button.

```
from pyforms.basewidget import BaseWidget
from confapp import conf
from pyforms.controls import ControlButton
from .inheritance_v3 import InheritanceV3

class OpenInheritanceV3(BaseWidget):

    UID                 = 'app-inheritancev3'
    TITLE               = 'Open inheritance v3'

    LAYOUT_POSITION     = conf.ORQUESTRA_HOME
    ORQUESTRA_MENU      = 'left'
    ORQUESTRA_MENU_ICON = 'cog'
    ORQUESTRA_MENU_ORDER = 7

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self._btn = ControlButton('Open inheritance v3', default=self.btn_evt)

        self.formset = ['_btn']
```

(continues on next page)

(continued from previous page)

```
    def btn_evt(self):
        InheritanceV3()
```

CHAPTER

NINE

FROM SQL TO DJANGO MODELS' APPS

This example demonstrates how to start from a raw database and end up with the forms to manage it in few steps.

Note: This examples is based on the code available at the folder tutorials/tutorial_advance_db

First of all start your django project:

```
> django-admin startproject tutorial_advance_db
```

Configure your django app settings.py file to point your database and include the pyforms configurations.

```
...

INSTALLED_APPS = [
    'tutorial',

    'orquestra',
    'pyforms_web.web',
    'jfu',

    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'pyforms_web.web.middleware.PyFormsMiddleware'
]
...
```

Enter in the project folder and create the tutorial app:

```
> cd tutorial_advance_db
> mkdir tutorial
```

Generate the Django models from your SQL database.

```
> python manage.py inspectdb > tutorial/models.py
```

Generate the Pyforms apps to manage the models by executing the next command:

```
> python manage.py export_pyforms_apps tutorial
```

Run migrations and execute the django project:

```
> python manage.py migrate
> python manage.py runserver
```

CHAPTER

TEN

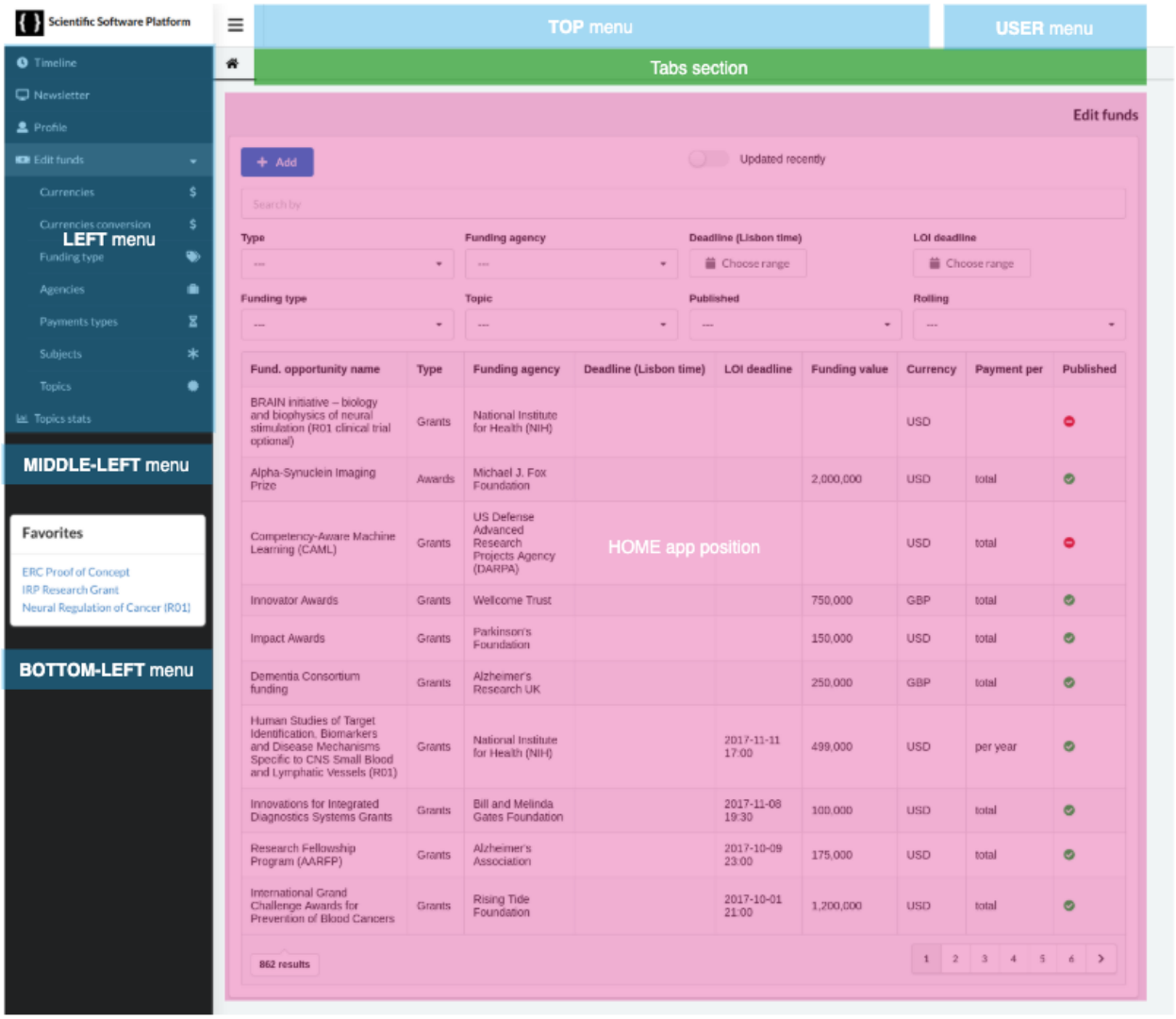
ORQUESTRA

Orquestra is an application layout responsible for rendering applications' layout.

Note: The HTML generated by Orquestra is already prepared to be responsive, thanks to the SemanticUI framework.

10.1 Configure an app layout

The Orquestra layout has predefined the next menus and apps rendering sections:



The apps may contain Orquestra flags to configure their behavior on the layout.

The flags are configured in the app class as it is exemplified bellow:

```
class SiteCrawlApp(BaseWidget):

    ...

    LAYOUT_POSITION      = conf.ORQUESTRA_HOME

    ORQUESTRA_MENU       = 'left'
    ORQUESTRA_MENU_ICON  = 'browser'
    ORQUESTRA_MENU_ORDER = 0
```

10.1.1 Available flags

10.1.1.1 UID

The flag besides being used to define the unique identifier of the application it is also used by Orquestra to define the **url** of the app, therefore if an app has the UID='something' the url of the app will be /app/something

10.1.1.2 LAYOUT_POSITION

This flags is used to configure where the application should be rendered within the layout. It can has the next values:

VALUE	Description
ORQUESTRA_HOME	The app should be rendered in the home position.
ORQUESTRA_HOME_FULL	The app should be rendered in the home position but should occupy all the available space.
ORQUESTRA_NEW_TAB	The app should be rendered in the home position, but in a new tab.
ORQUESTRA_NEW_TAB_FULL	The app should be rendered in the home position, but in a new tab and should occupy all the available space.
ORQUESTRA_APPEND_HOME	The app should be appended to the end of the home position. This configuration can be used to stack several apps on the home.
ORQUESTRA_NEW_WINDOW	The app should be rendered in a popup window.
ORQUESTRA_NEW_BIGWINDOW	The app should be rendered in a wider popup window.

10.1.1.3 Menus related

ORQUESTRA_MENU

The flag has configures the menu where the application should be appear. The available options are:

- left
- middle-left
- bottom-left
- top-menu
- user-menu

ORQUESTRA_MENU_ORDER

This flag sets the order to witch the app should appear in the menu.

ORQUESTRA_MENU_ICON

Name of the icon to be used for the app. The name should be available in the SemanticUI framework.

ORQUESTRA_URL

You may want a menu option which is a simple link. For this you can create an empty app and set this flag. The menu will be rendered with the url configured here instead of the app url.

ORQUESTRA_TARGET

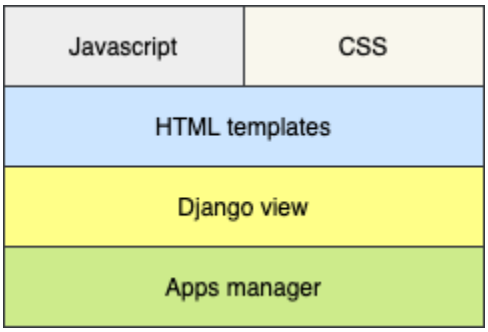
The flag will be used just in the case the ORQUESTRA_URL flag is set. The value of this flag will be setting the target attribute of the html tab <a>.

STATIC_FILES

The flag should be a list with the url of extra files we would like to import in the template. So if some app needs a specific javascript or css file to runs, you can use the flag to import them.

10.2 How Orquestra works

This module composed by the next layers.



10.2.1 Apps manager layer

The apps manager is the module responsible for finding and selecting the apps that are going to used in the construction of the layout menus. Orquestra use the apps manager to find

10.2.2 Django view layer

Orquestra has a django's view to generate the default html page. This view search for all the available apps to render the menus and apps according to the defined environment configurations.

10.2.3 HTML templates layer

The templates layer is where the html is configured. Here is where the layout is defined.

10.2.4 Javascript & CSS layer

10.2.4.1 CSS

This layer includes all the css used to render the layout.

10.2.4.2 Javascript

This layer includes all the js required to render the layout and set its behaviour. It is here also that the rendered sections are defined, so for each *LAYOUT_POSITION flag* value, there is a corresponding javascript function that render the app for the specific section.

10.3 Why such architecture?

The reason why the Orquestra application layer was create, was because like that the layout is separated from the apps execution. Having the exact same apps without touching any code I can refactor all the layout without just by switching the Orquestra layer by another.

Example:

This is a representation of where the Orquestra layer is positioned in relation to the pyforms framework.

Now if we switch the Orquestra layer by another layer called Kuloneco which implements rearranges the layout, the result will be the next:

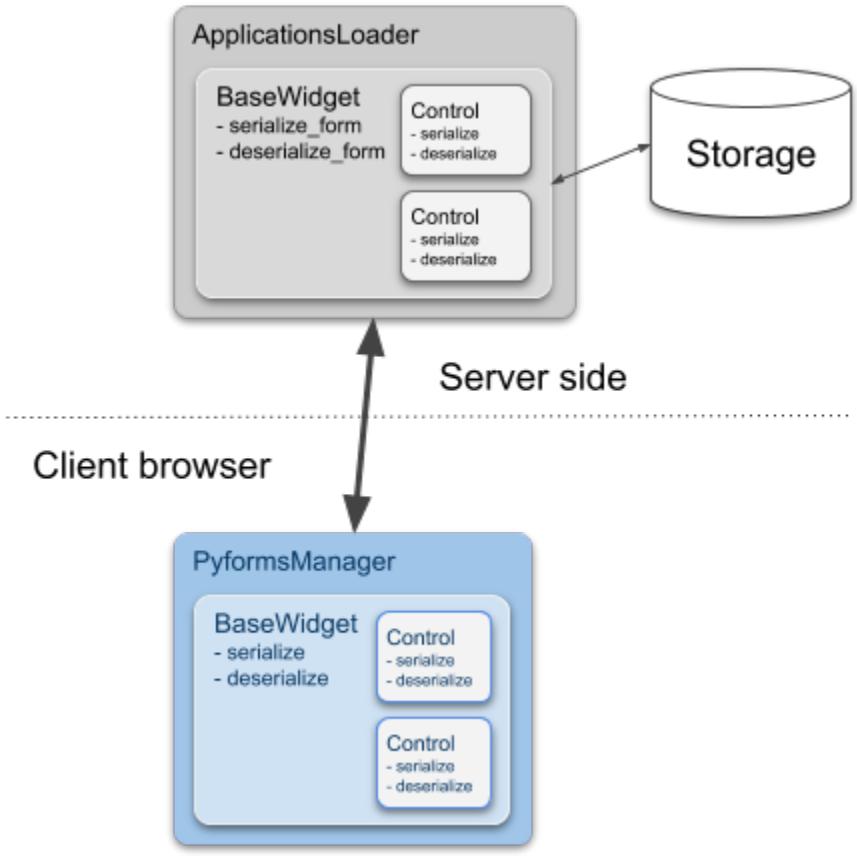
CHAPTER

ELEVEN

OVERVIEW

Pyforms web works by having on the client side a javascript layer handeling all the Widgets and Controls updates occurred on the server side. The Widgets and Controls updates are exchanged using the functions **serialize** and **deserialize** on the client side and the **serialize_form** and **deserialize_form** on the server side. Only updated controls are communicated from one side to the other.

Python when running in a Apache2 does not maintain in memory the objects created in each HTTP request, therefore everytime there is an update to a Widget, one is dumped to the disk and loaded when a new HTTP request arrive.

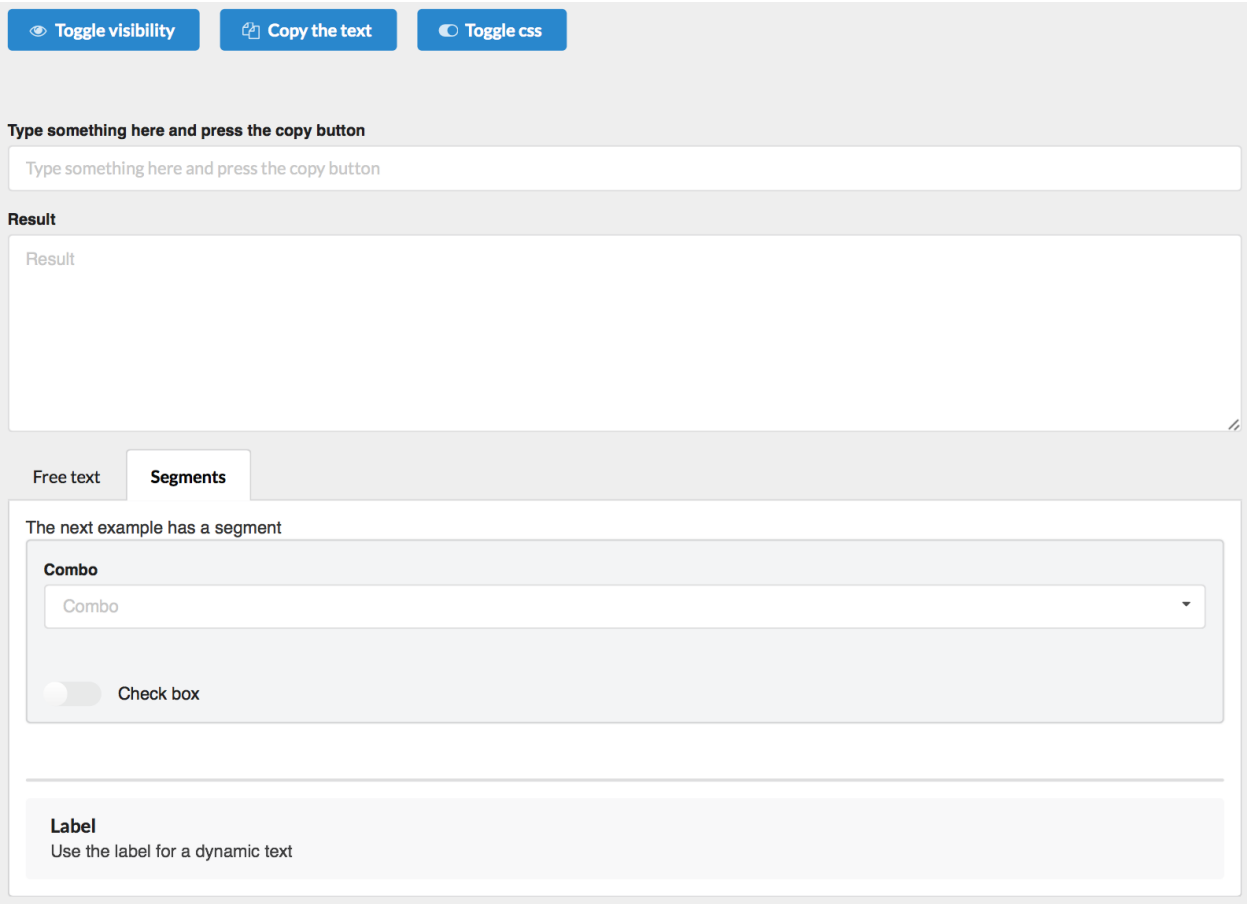


CHAPTER

TWELVE

PYTHON

12.1 BaseWidget



12.1.1 Overview

The BaseWidget class is the base class of all pyforms applications.

12.1.2 API

class pyforms_web.basewidget.**BaseWidget**(**args*, ***kwargs*)

Bases: object

The class implements a application form

Parameters

- **title** (*str*) – Title of the app. By default will assume the value in the class variable TITLE.
- **parent_win** (BaseWidget) – Parent BaseWidget

Example:

```
class FeedViewerApp(BaseWidget):

    TITLE = 'Feed viewer'

    def __init__(self, *args, **kwargs):

        self._likebtn    = ControlButton(label_visible=False, labeled=True)
        self._htmlviewer = ControlTemplate('Html', template=self.VIEWER_TEMPLATE)

        self.formset = ['_likebtn', '_htmlviewer']
```

URLS = None

list(str): Django urls to be added to the urls.py file

TITLE = None

str: Title of the application.

LAYOUT_POSITION = None

int or str: Id of the layout handler function registered in the javascript by the function [pyforms.register_layout_place] or Element DOM id in the HTML where the application should be shown.

REFRESH_TIMEOUT = None

str: Time in milliseconds to refresh the application.

AUTHORIZED_GROUPS = None

list(str): List of django groups authorized to run the application

CSS = ''

str: Css classes to add to the form.

STATIC_FILES = []

str: Static files to include in the page

TRIGGER_ON_KEYDOWN_CODES = []

list(int): List of keys codes to detect

init_form(*parent=None*)

Generate the application Form. Return the dict:

```
{
     'code': ...,           # HTML code that will initialize the application.
     'title': ...,          # Title of the application.
     'css': ...,            # Application CSS.
     'app_id': ...,         # Application id.
     'refresh_timeout': ... # Application refresh time.
}
```

generate_column(*row*)
Generate the html to organize the formset in segments

generate_nocolumns(*formset*)
Generate the html for the no_columns organizer

generate_panel(*formset*)
Generate a panel for the application form with all the controls:

Parameters **formset** (*list*) – formset configuration, used to generate the panel.

Example:

```
[
     no_columns('_toggle_btn','_copy_btn', '_css_btn'),
     ' ',
     ('empty:twelve','_input'),
     '_text',
     {
         'a:Free text': [
             'h1:Header 1',
             'h2:Header 2',
             'h3:Header 3',
             'h4:Header 4',
             'h5:Header 5',
             'h1-right:Header 1',
             'h2-right:Header 2',
             'h3-right:Header 3',
             'h4-right:Header 4',
             'h5-right:Header 5',
             '-',
             'Free text here',
             'msg:Message text',
             'info:Info message',
             'warning:Warning message',
             'alert:Alert message'
         ],
         'b:Segments': [
             'The next example has a segment',
             segment(
                 '_combo',
                 '_check',
                 css='secondary'
             ),
             '_list',
             '_label'
         ]
```

(continues on next page)

(continued from previous page)

```
    }
]
```

- **tuple**: displays the controls horizontally.
- **list**: displays the controls vertically.
- **dict**: displays the controls in Tabs.
  - Use [a:,b:,c:] prefix to sort the tabs.
- **'-'**: Draw a vertical line.
- **' '**: Empty column.
- **Empty column**: Use ' ', or the prefix 'empty:' + size of the column (ex: one, two, ..., sixteen) to add a empty column.
- **segment**: Wraps the formset around a segment (Semantic UI segment).
  - Call the parameter **css**, to add extra classes to the segment.
- **no_columns**: Do not apply the fields columns alignments.
- **Free text**: Do not apply the fields columns alignments.
- **Message**: By using the prefixes [msg:,info:,warning:,alert:] you will wrap a free message on message box.
- **Headers**: Use the prefixes [h1:,h2:,h3:,h4:,h5:,h1-right:,h2-right:,h3-right:,h4-right:,h5-right:] on free text.

`close()`
Close the application

`message`(*msg*, *title=None*, *msg_type=None*)
Write a simple message.

Parameters

- **msg** (*str*) – Message to show.
- **title** (*str*) – Message title.
- **msg_type** (*str*) – Message box css class.

`success`(*msg*, *title=None*)
Write a success message

Parameters

- **msg** (*str*) – Message to show.
- **title** (*str*) – Message title.

`info`(*msg*, *title=None*)
Write a info message

Parameters

- **msg** (*str*) – Message to show.
- **title** (*str*) – Message title.

warning(*msg*, *title=None*)
Write a warning message

Parameters

- **msg** (*str*) – Message to show.
- **title** (*str*) – Message title.

alert(*msg*, *title=None*)
Write a alert message

Parameters

- **msg** (*str*) – Message to show.
- **title** (*str*) – Message title.

message_popup(*msg*, *title=''*, *buttons=None*, *handler=None*, *msg_type=''*, *buttons_css=[]*)
Show a popup message window

Parameters

- **msg** (*str*) – Message to show.
- **title** (*str*) – Message title.
- **buttons** (*list(str)*) – List of buttons labels to create in the popup window.
- **msg_type** (*str*) – Message box css class.
- **handler** (*buttons_css*) – Method that will handle the press of the buttons.
- **handler** – CSS class of the buttons.

```
# Handler
def button_pressed_btn(popup=[Popup instance], button=[Label of the pressed␣
↪button]):
    ...
```

success_popup(*msg*, *title=''*, *buttons=None*, *handler=None*, *buttons_css=[]*)
Show a popup success message window

Parameters

- **msg** (*str*) – Message to show.
- **title** (*str*) – Message title.
- **buttons** (*list(str)*) – List of buttons labels to create in the popup window.
- **handler** (*buttons_css*) – Method that will handle the press of the buttons.
- **handler** – CSS class of the buttons.

info_popup(*msg*, *title=''*, *buttons=None*, *handler=None*, *buttons_css=[]*)
Show a popup info message window

Parameters

- **msg** (*str*) – Message to show.
- **title** (*str*) – Message title.
- **buttons** (*list(str)*) – List of buttons labels to create in the popup window.
- **handler** (*buttons_css*) – Method that will handle the press of the buttons.

- **handler** – CSS class of the buttons.

warning_popup(*msg*, *title=''*, *buttons=None*, *handler=None*, *buttons_css=[]*)
Show a popup warning message window

Parameters

- **msg** (*str*) – Message to show.
- **title** (*str*) – Message title.
- **buttons** (*list(str)*) – List of buttons labels to create in the popup window.
- **handler** (*buttons_css*) – Method that will handle the press of the buttons.
- **handler** – CSS class of the buttons.

alert_popup(*msg*, *title=''*, *buttons=None*, *handler=None*, *buttons_css=[]*)
Show a popup alert message window

Parameters

- **msg** (*str*) – Message to show.
- **title** (*str*) – Message title.
- **buttons** (*list(str)*) – List of buttons labels to create in the popup window.
- **handler** (*buttons_css*) – Method that will handle the press of the buttons.
- **handler** – CSS class of the buttons.

add_timeout(*milliseconds*, *call_function*)

commit(*user=None*)
Save all the application updates to a file, so it can be used in the next session.

execute_js(*code*)
This function executes a javascript remotely on the client side.

Parameters **code** (*str*) – Javascript code to execute.

mark_to_update_client()
Used to flag pyforms that the application was updated and the updates should be sent to the client side

deserialize_form(*params*)
Load the json parameters sent by the client side

Parameters **params** (*dict*) – Data to load.

serialize_form()
Serialize the Form to a control.

Returns: dict: Data representings the current state of the application.

stream_status(*user=None*)

stream(*func*)

classmethod has_permissions(*user*)
This class method, verifies if a user has permissions to execute the application

Parameters **params** (*User*) – User to availuate the permissions.

has_session_permissions(*user*)
It verifies if a user has permissions to execute the application during the runtime.

Parameters **params** (*User*) – User to availuate the permissions.

refresh_event()
Event called every X time defined by REFRESH_TIMEOUT variable.

keydown_event()

property controls
Returns all the form controls from the the module

property form
Return the basewidget html. The html is based on the 'basewidget-template.html' template

property title
Return and set the title of the application

property formset
Return and set the controls organization in the form

property uid
Return and set the application unique identifier

property refresh_timeout
Return a boolean indicating if the form is visible or not

property parent

property form_loaded
Return True if the Widget already executed the init_form() function or not.

12.2 Controls

12.2.1 ControlBase

class pyforms_web.controls.control_base.**ControlBase**(**args*, ***kwargs*)
Bases: object

Basis class from where all the Controls inherit from.

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

update_control_event()

init_form()
Render the control js

show()
Show the control

hide()
Hide the control

serialize()
Serialize the control data.

Returns: dict: Serialized data.

deserialize(*properties*)
Serialize the control data.

Parameters **properties** (*dict*) – Serialized data to load.

mark_to_update_client()
Mark the control to update in the client side.

changed_event()
Event called when the control value is changed.

property enabled
Set the control enabled or disabled.

property value
Set or return de control value.

property name
Set or return the name of the control.

property label
Set or return the label of the control.

property parent
Set or return the control window.

property visible
Set if the control is visible.

property help
Set or return the help text of the control.

property error
Set or return the error of the control.

property label_visible
Set or return the label visibility of the control.

property readonly
Set or return the control readonly mode.

property css
Set or return the extra css of the control.

property field_css
Set or return the extra css of the field div that encapsulates the control.

property style
Set or return the style of the control.

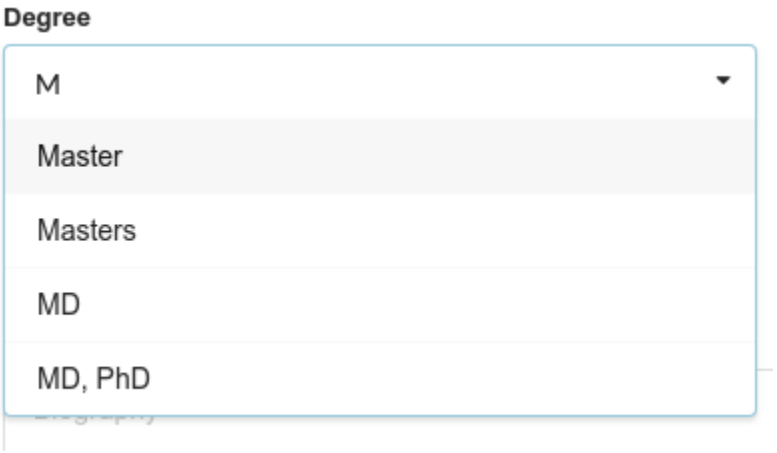
property field_style
Set or return the style of the field div that encapsulate the control.

property control_id
Return the control id.

property place_id
Return the place control id.

property was_updated
Return the if the control is marked as updated.

12.2.2 ControlAutoComplete



class pyforms_web.controls.control_autocomplete.**ControlAutoComplete**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

This control implements an auto complete text box. The control gets its values from a url or from a queryset.

Parameters

- **items_url** (*str*) – (Mandatory) Configure the Combo to get its items from an URL. Default=None.
- **autocomplete_search** (*function*) – Set the function to query the items in the case we are using the url mode. Default=self.autocomplete_search.
- **queryset** (*str*) – Set the queryset to which the autocomplete will get the values. Default=None.
- **multiple** (*boolean*) – Allow multiple choices. Default=False.
- **queryset_filter** (*function*) – Function to filter the queryset. Default=self.queryset_filter.

  ..code: python

  def queryset_filter(self, qs, keyword, control): return qs

In the case you are using a Model queryset to retrieve the values you can define the fields to lookup for using the staticmethod **autocomplete_search_fields** as it is shown inthe example bellow.

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

init_form()
Render the control js

property queryset

autocomplete_search(*keyword*)

deserialize(*data*)
Serialize the control data.

Parameters properties (*dict*) – Serialized data to load.

serialize()
Serialize the control data.

Returns: dict: Serialized data.

property objects
Return the selected objects.

property parent
Set or return the control window.

12.2.3 ControlBarsChart

#.. image:: /_static/imgs/controls/autocomplete.png

class pyforms_web.controls.control_barschart.**ControlBarsChart**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.

- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

init_form()
Render the control js

remote_data_selected_event()

data_selected_event(*series_index*, *data*)

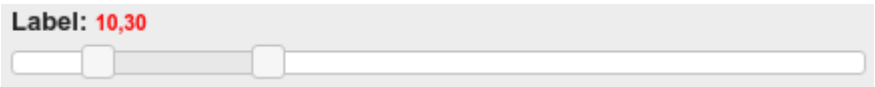
serialize()
Serialize the control data.

Returns: dict: Serialized data.

deserialize(*properties*)
Serialize the control data.

Parameters **properties** (*dict*) – Serialized data to load.

12.2.4 ControlBoundingSlider



class pyforms_web.controls.control_boundingslider.**ControlBoundingSlider**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

property min

property max

12.2.5 ControlBreadcrumb

#.. image:: /_static/imgs/controls/autocomplete.png

class pyforms_web.controls.control_breadcrumb.**ControlBreadcrumb**(**args*, ***kwargs*)

Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

init_form()

Render the control js

pressed()

This event is called when the button is pressed. The correspondent js event is defined in the framework.js file

serialize()

Serialize the control data.

Returns: dict: Serialized data.

deserialize(*properties*)

Serialize the control data.

Parameters **properties** (*dict*) – Serialized data to load.

12.2.6 ControlButton

class pyforms_web.controls.control_button.**ControlButton**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

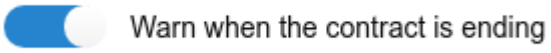
Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

pressed()
This event is called when the button is pressed. The correspondent js event is defined in the framework.js file

---

12.2.7 ControlCheckBox



class pyforms_web.controls.control_checkbox.**ControlCheckBox**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.

- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

12.2.8 ControlCheckBoxList

class pyforms_web.controls.control_checkboxlist.**ControlCheckBoxList**(**args*, ***kwargs*)

Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

item_selection_changed_event()

item_selection_changed_client_event()

property headers

property rows_count

property columns_styles

property columns_align

property selected_row_index

property value

Sets and gets the values.

The values should have the

12.2.9 ControlCheckBoxListQuery

class pyforms_web.controls.control_checkboxlistquery.**ControlCheckBoxListQuery**(**args*, ***kwargs*)

Bases: *pyforms_web.controls.control_checkboxlist.ControlCheckBoxList*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

obj2list(*row*)

remove_event()

add_event()

property queryset

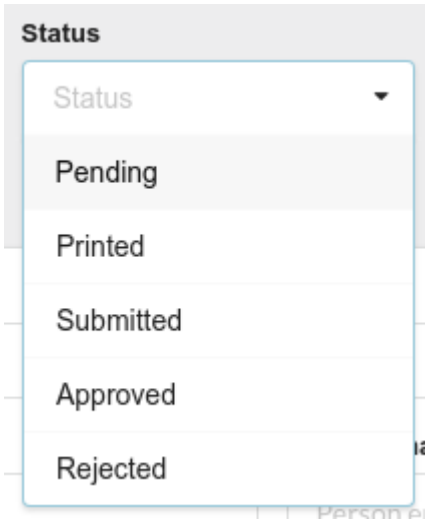
property value

Sets and gets the values.

The values should have the

property items

12.2.10 ControlCombo



class pyforms_web.controls.control_combo.**ControlCombo**(**args*, ***kwargs*)

Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

add_item(*label*, *value=<class 'pyforms_web.controls.control_combo.ValueNotSet'>*)

clear_items()

property items

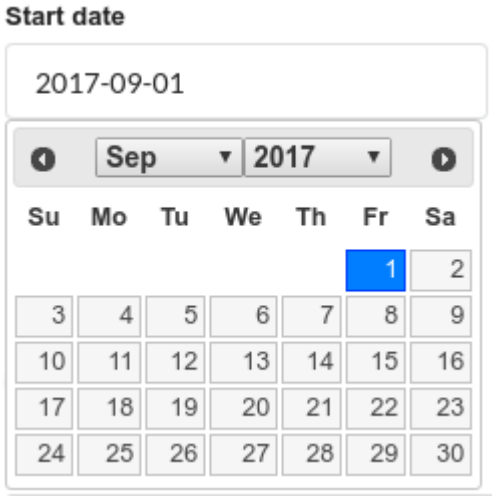
property values

property value

Set or return de control value.

property text

12.2.11 ControlDate



class pyforms_web.controls.control_date.**ControlDate**(**args*, ***kwargs*)

Bases: *pyforms_web.controls.control_base.ControlBase*

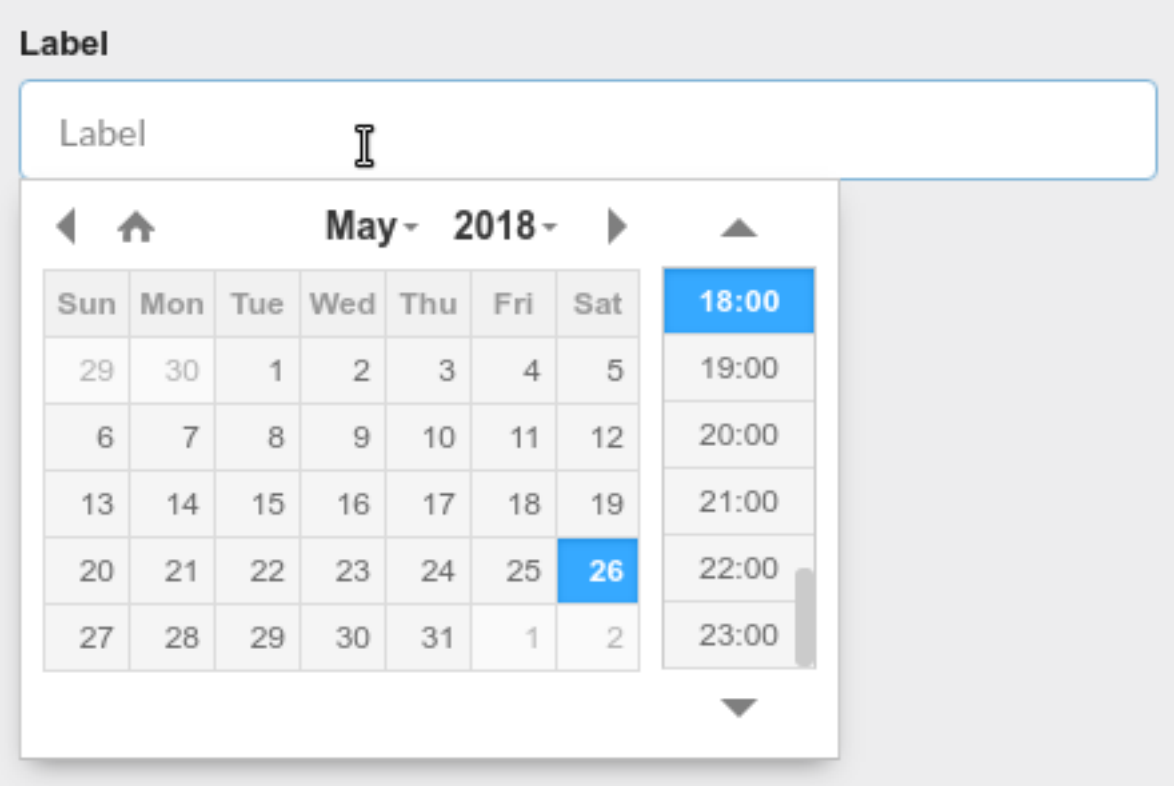
Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

property value

Set or return de control value.

12.2.12 ControlDateTime



class pyforms_web.controls.control_datetime.**ControlDateTime**(**args*, ***kwargs*)

Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

property value

Set or return de control value.

12.2.13 ControlDecimal

class pyforms_web.controls.control_decimal.**ControlDecimal**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

12.2.14 ControlEmail

class pyforms_web.controls.control_email.**ControlEmail**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_text.ControlText*

Parameters **on_enter_event** (*function*) – Event called when the Enter key is pressed.

12.2.15 ControlEmptyWidget

class pyforms_web.controls.control_emptywidget.**ControlEmptyWidget**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

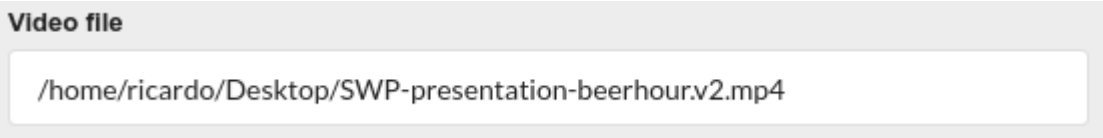
Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.

- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

property value
Set or return de control value.

12.2.16 ControlFile



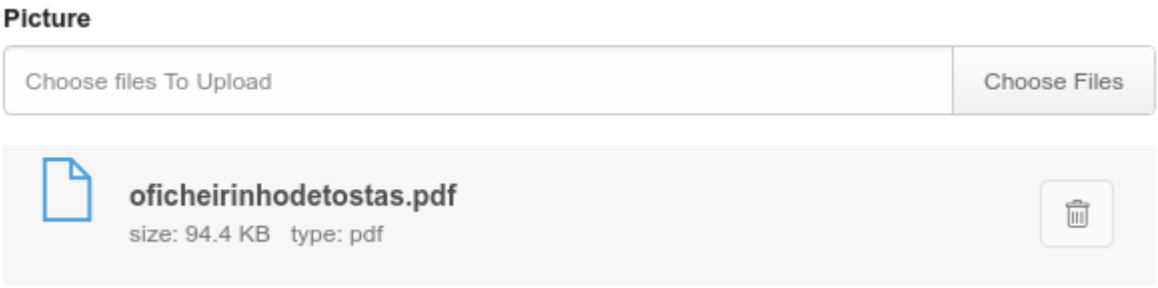
class pyforms_web.controls.control_file.**ControlFile**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

open_file_browser()

12.2.17 ControlFileUpload



class pyforms_web.controls.control_fileupload.**ControlFileUpload**(**args*, ***kwargs*)

Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

property filepath

move_to(*dest_dirpath*)

12.2.18 ControlFloat

class pyforms_web.controls.control_float.**ControlFloat**(**args*, ***kwargs*)

Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.

- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

12.2.19 ControlHtml

class pyforms_web.controls.control_html.**ControlHtml**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

12.2.20 ControlImage

class pyforms_web.controls.control_image.**ControlImage**(**args*, ***kwargs*)

Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

save(*data*)

load(*data*)

repaint()

set_url(*url*)

property value

Set or return de control value.

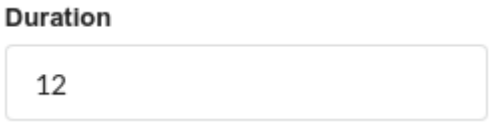
12.2.21 ControlImg

class pyforms_web.controls.control_img.**ControlImg**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

12.2.22 ControlInteger



class pyforms_web.controls.control_integer.**ControlInteger**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

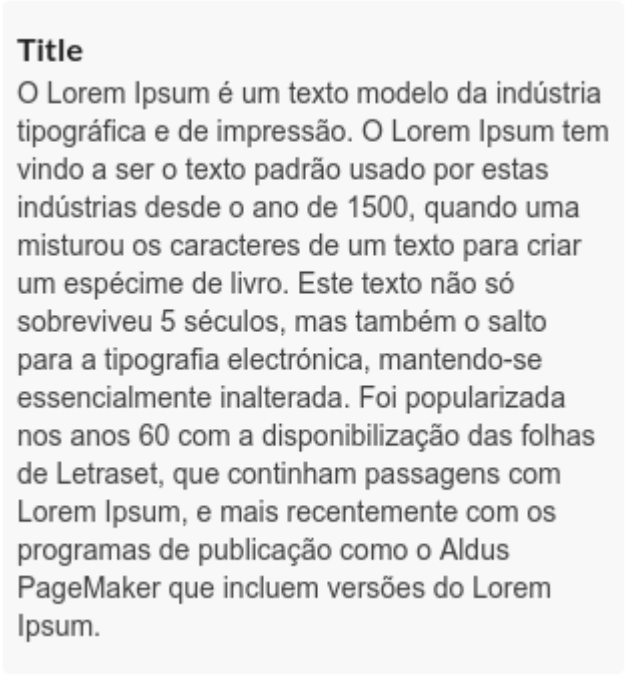
- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.

- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

property value
Set or return de control value.

clean_field()
Validate the value of the Control.

12.2.23 ControlLabel



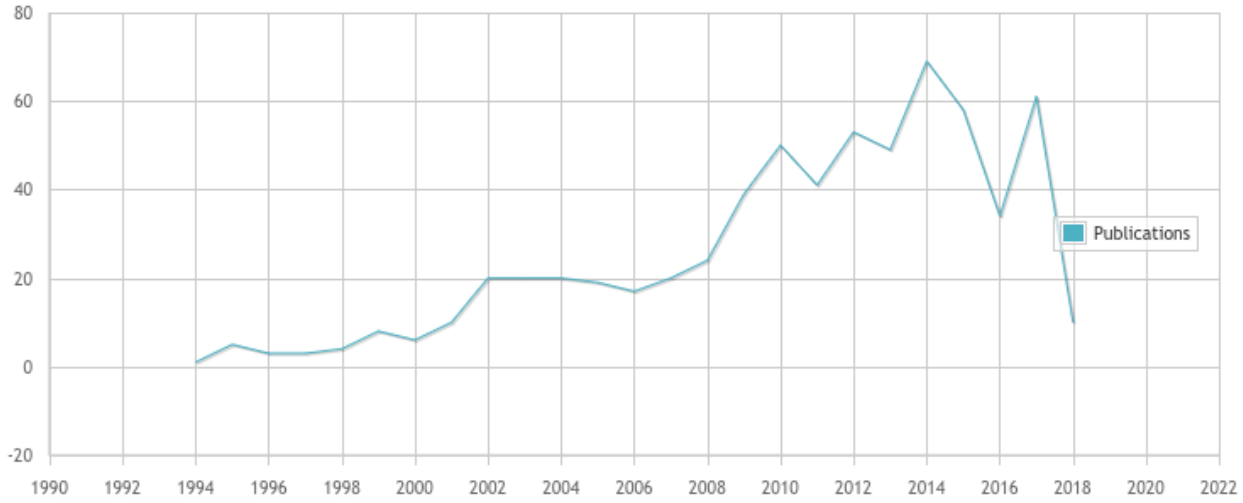
class pyforms_web.controls.control_label.**ControlLabel**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.

- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

12.2.24 ControlLineChart



class pyforms_web.controls.control_linechart.**ControlLineChart**(**args*, ***kwargs*)

Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

remote_data_selected_event()

data_selected_event(*series_index*, *data*)

12.2.25 ControlList

class pyforms_web.controls.control_list.**ControlList**(**args*, ***kwargs*)

Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = True.
- **horizontal_headers** (*list(str)*) – List of headers.
- **select_entire_row** (*bool*) – Flag to select the entire row or individual cells. Default = False.
- **selected_row_index** (*int*) – Default selected row. Default = -1.
- **columns_size** (*list(str)*) – List of css with the size value of each column. Default = None.
- **columns_align** (*list(str)*) – List of css alignment values for each column. Default = [].
- **row_double_click_event** (*function*) – Reference to a function called when the row is double clicked.
- **item_selection_changed_event** (*function*) – Reference to a function called when the selection of an item change.

item_selection_changed_event()

row_double_click_event()

append(*row*)

pop(*row_index*)

property horizontal_headers

property rows_count

property select_entire_row

property readonly

Set or return the control readonly mode.

property columns_style

property selected_row

property selected_row_index

property value
Set or return de control value.

property columns_size

property columns_align

12.2.26 ControlMultipleChecks

class pyforms_web.controls.control_multiplechecks.**ControlMultipleChecks**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

add_item(*label*, *value=None*)

clear_items()

12.2.27 ControlMultipleSelection

class pyforms_web.controls.control_multipleselection.**ControlMultipleSelection**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.

- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

add_item(*label*, *value=None*)

clear_items()

12.2.28 ControlMultipleUpload

class pyforms_web.controls.control_multipleupload.**ControlMultipleUpload**(**args*, ***kwargs*)

Bases: *pyforms_web.controls.control_base.ControlBase*

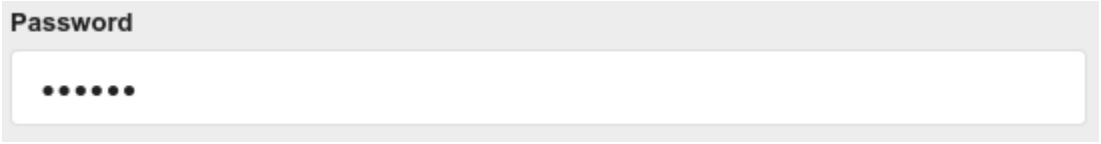
Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

filepaths()

move_to(*dest_dirpath*)

12.2.29 ControlPassword



class pyforms_web.controls.control_password.**ControlPassword**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

12.2.30 ControlPieChart

class pyforms_web.controls.control_piechart.**ControlPieChart**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.

- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

12.2.31 ControlProgress

class pyforms_web.controls.control_progress.**ControlProgress**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

property min

property max

12.2.32 ControlQueryList

class pyforms_web.controls.control_querylist.**ControlQueryList**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.

- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

item_selection_changed_client_event()

item_selection_changed_event()

export_csv_event()
Event called to export the queryset to excel

export_csv_http_response()

property export_csv_columns
Sets and gets the list of columns to be used in the cvs export By default it will assume the self.list_display value

property export_csv
Flag to activate or deactivate the csv export button

property selected_row_id

property columns_size

property columns_align

property value
Set or return de control value.

page_changed_event()

sort_changed_event()

filter_changed_event()

filter_event()

page_event()

sort_event()

format_filter_column(*col_value*)

queryset_to_list(*queryset*, *list_display*, *first_row*, *last_row*)

serialize_filters(*list_filter*, *queryset*)

12.2.33 ControlSimpleLabel

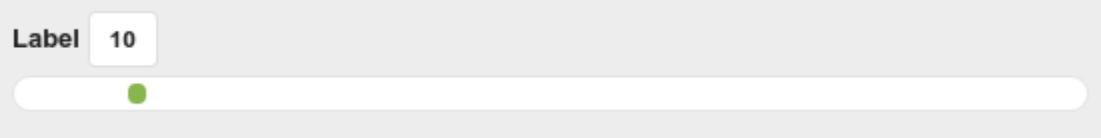
class pyforms_web.controls.control_simplelabel.**ControlSimpleLabel**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.

- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

12.2.34 ControlSlider



class pyforms_web.controls.control_slider.**ControlSlider**(**args*, ***kwargs*)

Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

property min

property max

12.2.35 ControlSpacer

class pyforms_web.controls.control_spacer.**ControlSpacer**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

12.2.36 ControlTemplate

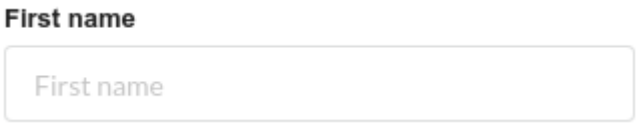
class pyforms_web.controls.control_template.**ControlTemplate**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.
- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

property **template**

12.2.37 ControlText

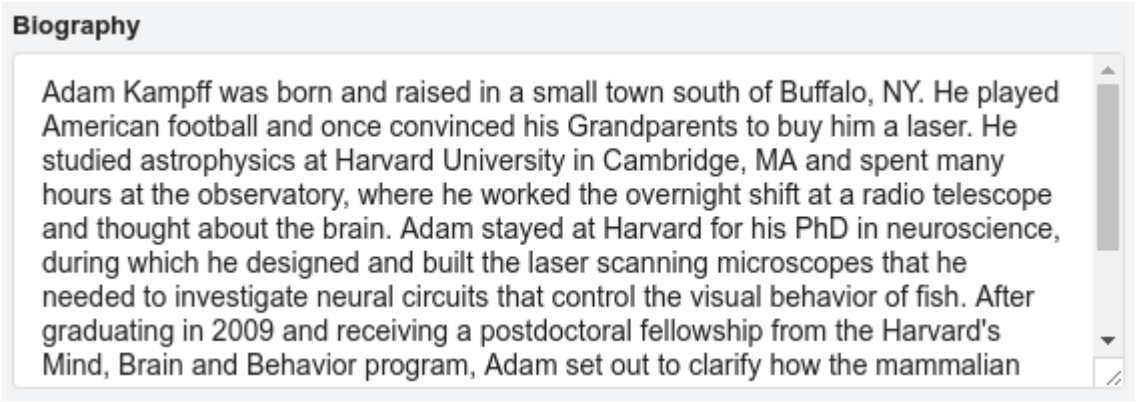


class pyforms_web.controls.control_text.**ControlText**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters **on_enter_event** (*function*) – Event called when the Enter key is pressed.

on_enter_event()
Event called when the Enter key is pressed

12.2.38 ControlTextArea



class pyforms_web.controls.control_textarea.**ControlTextArea**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_text.ControlText*

Parameters **on_enter_event** (*function*) – Event called when the Enter key is pressed.

stream(*func*)

12.2.39 ControlTime

class pyforms_web.controls.control_time.**ControlTime**(**args*, ***kwargs*)
Bases: *pyforms_web.controls.control_base.ControlBase*

Parameters

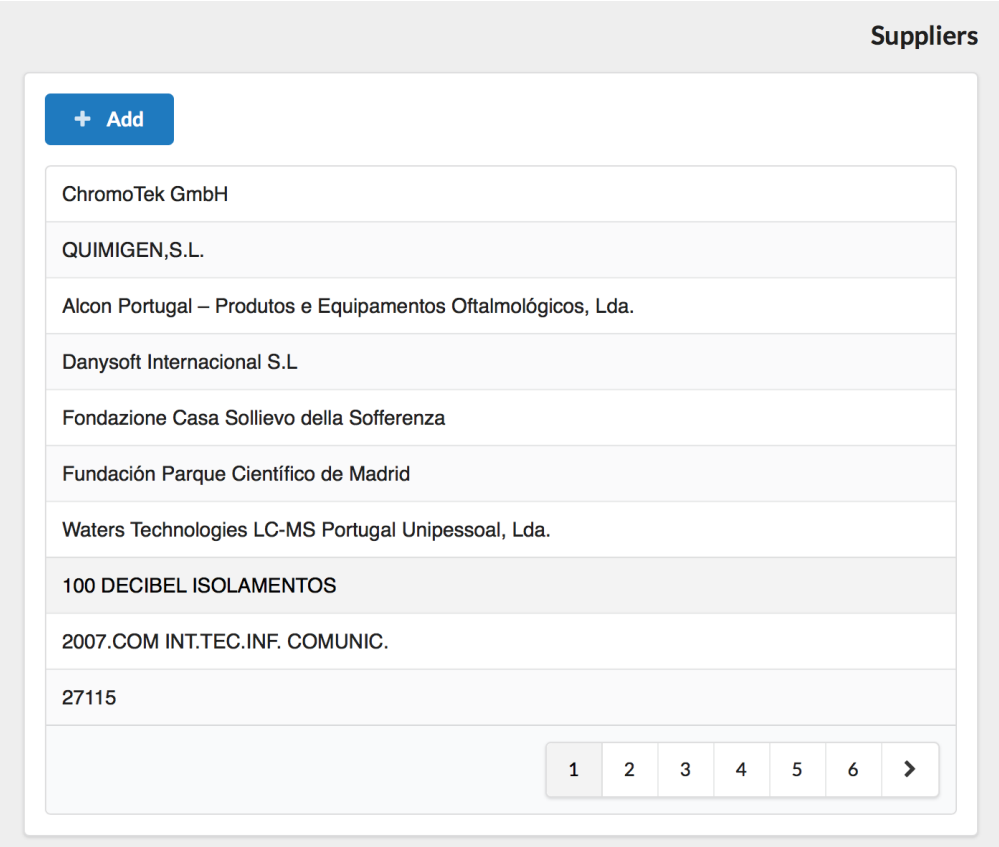
- **label** (*str*) – Control label.
- **helptext** (*str*) – Set the help text. Default = None.
- **default** (*str*) – Set the value. Default = None.
- **visible** (*bool*) – Set the control visible or hidden. Default = True.
- **error** (*bool*) – Mark the control as having and error. Default = False.

- **css** (*str*) – Extra css classes to add to the control.
- **field_css** (*str*) – Extra css classes to add to the field dive that encapsulates the control.
- **style** (*str*) – Extra style to add to the control.
- **field_style** (*str*) – Extra style to add to the field div that encapsulates the control.
- **enabled** (*bool*) – Set the control enabled or disabled. Default = True.
- **readonly** (*bool*) – Set the control as read only. Default = False.
- **label_visible** (*bool*) – Hide or show the label. Default = True.

property value
Set or return de control value.

12.3 Django admin

12.3.1 ModelAdminWidget



class pyforms_web.widgets.django.modeladmin.**ModelAdminWidget**(**args*, ***kwargs*)
Bases: *pyforms_web.basewidget.BaseWidget*

The class is used to generate an admin interface for ModelAdmin.MODEL.

```
from suppliers.models import Suplier

class SupplierAdminApp(ModelAdminWidget):

     MODEL = Supplier
     TITLE = 'Suppliers'
```

Parameters

- **title** (*str*) – Title of the app. By default will assume the value in the class variable TITLE.
- **model** (*django.db.models.Model*) – Model the App will manages. By default will assume the value in the class variable MODEL.
- **editform_class** (*class*) – Class used to generate the edition form. By default will assume the value in the class variable EDITFORM_CLASS.
- **parent_pk** (*int*) – (optional) Used to generate the inline interface. Primary key of the parent model
- **parent_model** (*Model*) – (optional) Used to generate the inline interface. Parent model

MODEL = None
class: Model to manage

TITLE = None
str: Title of the application

EDITFORM_CLASS
class: Edit form class

alias of *pyforms_web.widgets.django.modelform.ModelFormWidget*

ADDFORM_CLASS = None
class: Create form class

USE_DETAILS_TO_ADD = True
boolean: Use the flag to create the ControlEmptyWidget self._details. This control is used to load the ADDFORM_CLASS.

USE_DETAILS_TO_EDIT = True
boolean: Use the flag to create the ControlEmptyWidget self._details. This control is used to load the EDITFORM_CLASS.

INLINES = []
list(class): Sub models to show in the interface

LIST_FILTER = None
list(str): List of filters fields

LIST_DISPLAY = None
list(str): List of fields to display in the table

LIST_HEADERS = None
list(str): Table columns headers. It will override the LIST_DISPLAY

LIST_COLS_SIZES = None
list(str): Table columns sizes. Should use style units.

LIST_COLS_ALIGN = None
list(str): Table columns alignments. Should use style units.

SEARCH_FIELDS = None
list(str): Fields to be used in the search

EXPORT_CSV = False
boolean: Flag to activate the export of data to csv. The value of this flag is overwritten by the function has_export_csv_permissions

EXPORT_CSV_COLUMNS = None
list(str): List of fields to export to the csv file. By default it will assume the fields in the LIST_DISPLAY variable

EXPORT_CSV_HEADERS = {}
dict(str: str): Provide custom header labels to fields listed in EXPORT_CSV_COLUMNS, e.g. {'date': 'Procedure Date'}

CONTROL_LIST
class: Control to be used in to list the values

alias of *pyforms_web.controls.control_querylist.ControlQueryList*

FIELDSETS = None
Formset of the edit form

READ_ONLY = []
list(str): List of readonly fields

LIST_ROWS_PER_PAGE = 10
int: number of rows to show per page

LIST_N_PAGES = 5
int: number of pages to show in the list bottom

ADD_BTN_LABEL = ' Add'
str: Label of the add button

property selected_row_object
django.db.models.Model: Return the current selected row object. If no row is selected return None.

get_toolbar_buttons(*has_add_permission=False*)
This function generate the formset configuration for the top buttons,

Returns: list(str): Returns the formset configuration that will be append to the end of the fieldsets.

populate_list()
Function called to configure the CONTROL_LIST to display the data

get_queryset(*request*, *queryset*)
The function retrives the queryset used to polulate the list.

Parameters **queryset** (*django.db.models.query.QuerySet*) – Default queryset used to populate the list. This queryset may have already applied the next filters: - If this class is being used as a inline app, the filters to select only the rows related with the parent app are applied. - If the model being managed by this class has a function called get_queryset(request, queryset), the filters applied by this function are applied. (this helps maintaining the visualization rules on the side of the model)

Returns: django.db.models.query.QuerySet: Returns the queryset used to populate the list.

get_related_field_queryset(*request*, *list_queryset*, *field*, *queryset*)
Called to return the main list filters for the ForeignKeys and ManyToMany fields.

Parameters

- **request** (*django.http.request.HttpRequest*) – HttpRequest originating the call of this function.
- **list_queryset** (*django.db.models.query.QuerySet*) – Queryset of the main list.
- **field** (*django.db.models.fields.Field*) – Related django field.
- **queryset** (*django.db.models.query.QuerySet*) – Default queryset for the related field.

Returns: django.db.models.query.QuerySet: Results.

hide_form()
Function called to hide the form

show_create_form()
Show an empty for for creation

show_edit_form(*obj_pk=None*)
Show the edition for for a specific object

Parameters **obj_pk** (*int*) – Primary key of the object to be show in the edit form.

get_editmodel_class(*obj*)
Gets the pyforms app to edit the object

Parameters **obj** (*django.db.models.Model*) – Object to be edited

set_parent(*parent_model*, *parent_pk*)
Function called to set prepare the Application to work as an inline

Parameters

- **parent_model** (*django.db.models.Model*) – Model of the parent Edition form
- **parent_pk** (*int*) – Primary key of the parent object

has_add_permissions()
Function called to check if one has permission to add new objects.

Returns: bool: True if has add permission, False otherwise.

has_view_permissions(*obj*)
Function called to check if one has permission to view the current queryset.

Parameters **obj** (*django.db.models.Model*) – object to view.

Returns: bool: True if has view permissions, False otherwise.

has_remove_permissions(*obj*)
Function called to check if one has permission to remove the current queryset.

Parameters **obj** (*django.db.models.Model*) – object to remove.

Returns: bool: True if has remove permissions, False otherwise.

has_update_permissions(*obj*)
Function called to check if one has permission to update the current queryset.

Parameters **obj** (*django.db.models.Model*) – object to update.

Returns: bool: True if has update permissions, False otherwise.

has_export_csv_permissions(*user*)
Function called to check if one has permission to export the objects to csv.

Parameters **django.contrib.auth.models.User** – User to check the permission.

Returns: bool: True if has permissions, False otherwise.

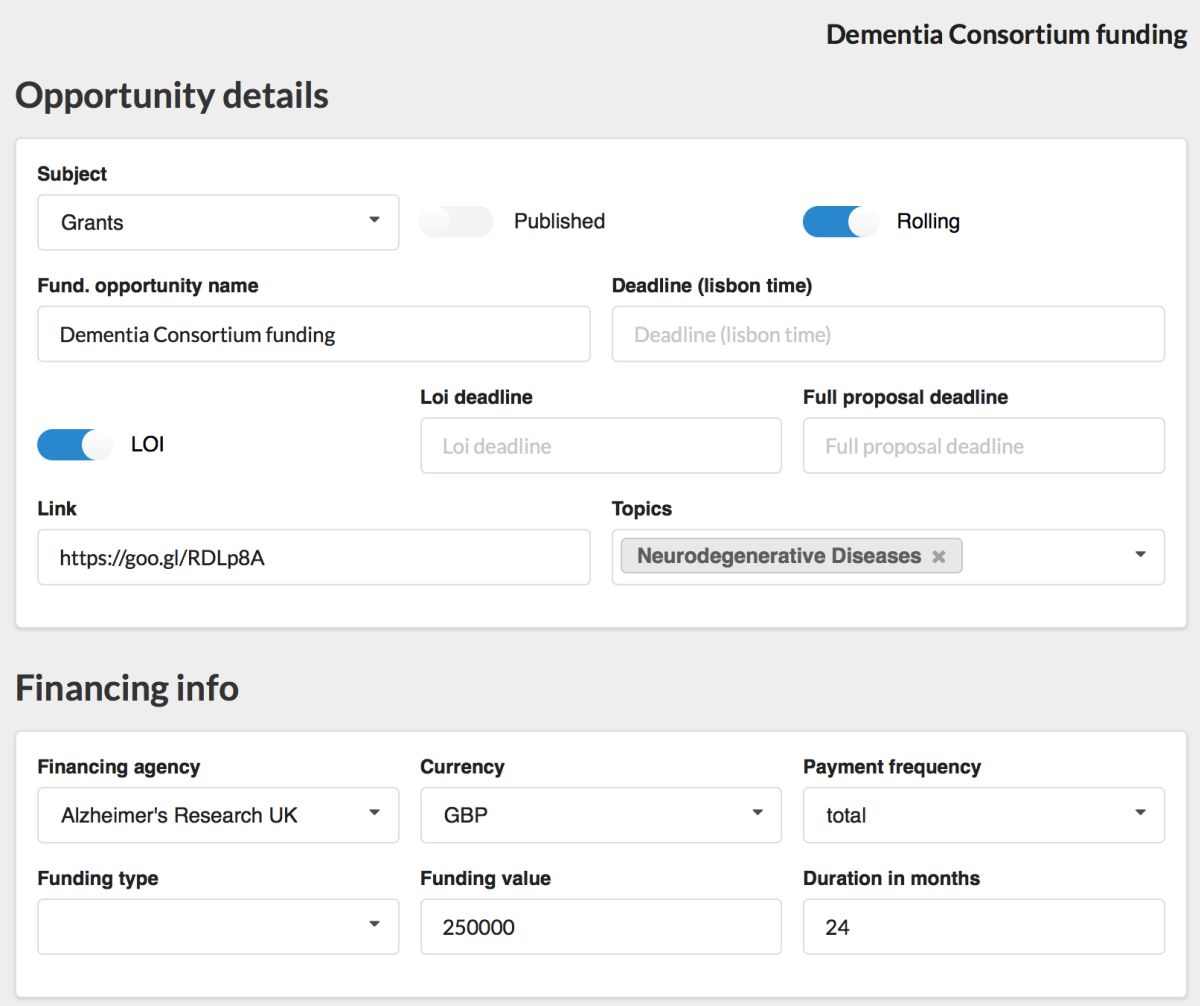
get_export_csv_columns(*user*)
Function called to get the columns for the csv export.

Parameters **django.contrib.auth.models.User** – User to check the permission.

Returns: list(str): List of columns names.

filter_event()
Function called when the filter is updated.

12.3.2 ModelFormWidget



class pyforms_web.widgets.django.modelform.**ModelFormWidget**(**args*, ***kwargs*)
Bases: *pyforms_web.basewidget.BaseWidget*

When a Pyforms application inherit from this class a form for the model ModelFormWidget.MODEL is created.

Usage example:

```
from funding.models import FundingOpportunity

class EditFundingOpportunitiesApp(ModelFormWidget):

    TITLE = "Edit opportunities"
    MODEL = FundingOpportunity

    FIELDSETS = [
        'h2:Opportunity details',
        segment([
            ('subject','fundingopportunity_published','fundingopportunity_rolling
↪'),
            ('fundingopportunity_name','fundingopportunity_end'),
            ('_loi','fundingopportunity_loideadline', 'fundingopportunity_
↪fullproposal'),
            ('fundingopportunity_link','topics'),
        ]),
        'h2:Financing info',
        segment([
            ('financingAgency','currency','paymentfrequency'),
            ('fundingtype','fundingopportunity_value','fundingopportunity_duration
↪'),
        ]),
        'h2:Description',
        segment([
            'fundingopportunity_eligibility',
            'fundingopportunity_scope',
            'fundingopportunity_brifdesc',
        ])
    ]
```

Parameters

- **title** (*str*) – Title of the app. By default will assume the value in the class variable TITLE.
- **model** (*django.db.models.Model*) – Model with the App will represent. By default will assume the value in the class variable MODEL.
- **inlines** (*list(ModelAdminWidget)*) – Sub models to show in the interface
- **fieldsets** (*list(str)*) – Organization of the fields
- **parent_pk** (*int*) – Parent model key
- **parent_model** (*django.db.models.Model*) – Parent model class
- **pk** (*int*) – Model register to manage

MODEL = None
class: Model to manage

TITLE = None
str: Title of the application

INLINES = []
list(class): Sub models to show in the interface

FIELDSETS = None
Formset of the edit form

READ_ONLY = []
list(str): List of readonly fields

HAS_CANCEL_BTN_ON_EDIT = True
bool: Flag to show or hide the cancel button

HAS_CANCEL_BTN_ON_ADD = True
bool: Flag to show or hide the cancel button

CLOSE_ON_REMOVE = False
bool: Close the application on remove

CLOSE_ON_CANCEL = False
bool: Close the application on cancel

POPULATE_PARENT = True
bool: Call populate_list function of the parent application when object is saved, updated or deleted

SAVE_BTN_LABEL = ' Save'
str: Label for the save button

CREATE_BTN_LABEL = ' Create'
str: Label for the create button

CANCEL_BTN_LABEL = ' Close'
str: Label for the cancel button

REMOVE_BTN_LABEL = ' Remove'
str: Label for the delete button

POPUP_REMOVE_TITLE = 'The next objects are going to be affected or removed'
str: Label for the popup window for the delete confirmation

property parent_object

property model_object
django.db.models.Model object: Return the current object in edition.

update_permissions_variables()

get_readonly(*default*)
The function returns the readonly fields to be set in the form.

Parameters **default** (*list(str)*) – Default readonly configuration.

Returns: list(str): Read only fields. Check class variable READ_ONLY to know more about it.

get_fieldsets(*default*)
The function returns the fieldsets organization to be set in the form.

Parameters **default** (*list(str)*) – Default fieldsets configuration.

Returns: list(str): fieldsets. Check class variable FIELDSETS to know more about it.

get_buttons_row()
This function generate the formset configuration for the save, create, cancel and remove buttons,

Returns: list(str): Returns the formset configuration that will be append to the end of the fieldsets.

hide_form()
This functions hides the create and edit form.

show_form()
This shows the create and edit form.

cancel_btn_event()
Event called when the cancel button is pressed

autocomplete_search(*queryset*, *keyword*, *control*)
Function used by a combobox to get the items dynamically

Parameters

- **queryset** (*django.db.models.query.QuerySet*) – Queryset from where to filter the results
- **keyword** (*str*) – Keyword for filter the results
- **pyforms.controls.BaseControl** – Control calling the autocomplete

Returns: django.db.models.query.QuerySet: Queryset used to update the autocomplete control

get_related_field_queryset(*field*, *queryset*)
Function called to manages the query for related fields like ForeignKeys and ManyToMany.

Parameters

- **field** (*django.db.models.fields.Field*) – Related django field.
- **queryset** (*django.db.models.query.QuerySet*) – Default queryset for the related field.

Returns: django.db.models.query.QuerySet: Results for the search in the format.

update_related_field(*field*, *pyforms_field*, *queryset*)
Function called update the related fields like ForeignKeys and ManyToMany.

Parameters

- **field** (*django.db.models.fields.Field*) – Related django field.
- **pyforms_field** (ControlBase) – Pyforms field that will be updated.
- **queryset** (*django.db.models.query.QuerySet*) – Default queryset for the related field.

show_create_form()
This function prepares the fields to be shown as create form.

update_callable_fields()
Update the callable fields after the form is saved.

update_autonumber_fields()
Update the auto number fields after the form is saved.

show_edit_form(*pk=None*)
This function prepares the fields to be shown as edit form.

Parameters **pk** (*int*) – Primary key of the object to be show in the edit form.

Returns:

django.db.models.Model object Returns the object in edition.

`delete_event()`
Function called to delete the current object in edition.

Returns:

bool True if the object was deleted with success, False if not.

`popup_remove_handler(popup=None, button=None)`
Function that handles the buttons events of the object delete confirmation popup.

Parameters

- **popup** (`BaseWidget`) – Popup application.
- **button** (`str`) – Label of the pressed button.

`create_newobject()`
Function called to create a new object of the model.

Returns:

django.db.models.Model object Created object

`save_object(obj, **kwargs)`
Function called to save the object It validates the form fields values.

Parameters

- **obj** (*django.db.models.Model*) – Object to save.
- **kwargs** (*dict*) – Any named argument passed to this function will be passed to the Model save method. Example: Model.save(**kwargs).

`validate_object(obj)`
Function called the model object

Parameters **obj** (*django.db.models.Model*) – Object to validate.

Returns:

django.db.models.Model object Created object or None if the object was not saved with success.

`update_object_fields(obj)`
Update the obj fields values with the form inputs values

Parameters **obj** (*django.db.models.Model*) – Object to update the values.

Returns:

django.db.models.Model Updated object.

`save_related_fields(obj)`
Save related fields

Parameters **obj** (*django.db.models.Model*) – Parent object to save.

Returns:

django.db.models.Mode Object passed as parameter

save_form_event(*obj*)
Function handling the form save. This function, updates the obj with the form values, validate the obj fields, and call the save_event function.

Parameters **obj** (*django.db.models.Model*) – Model object used for the save.

Returns:

boolean It returns True or False if the save was successfully.

save_event(*obj*, *new_object*)
Function handling the form save. This function, updates the obj with the form values, validate the obj fields, and call the save_event function.

Parameters

- **obj** (*django.db.models.Model*) – Model object used for the save.
- **new_object** (*Boolean*) – Flag indicating if it is an Add or Update.

Returns:

boolean It returns True or False if the save was successfully.

get_visible_fields_names()
Function called to get names of the visible fields.

Returns:

list(str) List names of the visible fields.

create_model_formfields()
Create the model edition form.

save_btn_event()
Event called by the save button

has_add_permissions()
The functions returns if the user has permissions to add objects or not.

Returns: bool: True if has add permissions, False otherwise.

has_view_permissions()
The functions returns if the user has permissions to view the queryset or not.

Returns: bool: True if has view permissions, False otherwise.

has_session_permissions(*user*)
It verifies if a user has permissions to execute the application during the runtime.

Parameters **params** (*User*) – User to availuate the permissions.

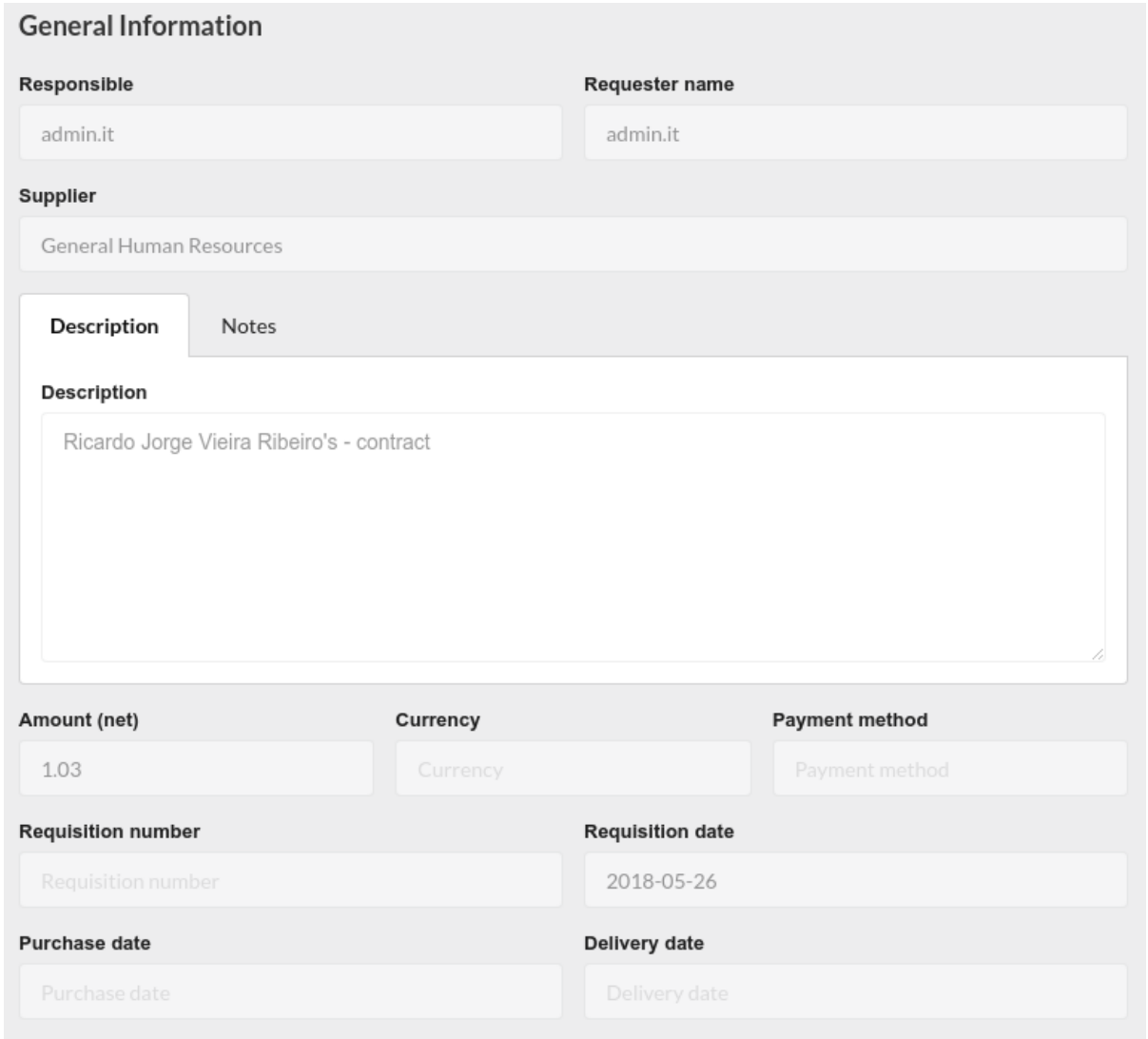
has_remove_permissions()
The functions returns if the user has permissions to remove the current queryset or not.

Returns: bool: True if has remove permissions, False otherwise.

has_update_permissions()
The functions returns if the user has permissions to update the current queryset or not.

Returns: bool: True if has update permissions, False otherwise.

12.3.3 ModelViewFormWidget



class pyforms_web.widgets.django.modelviewform.**ModelViewFormWidget**(**args*, ***kwargs*)

Bases: *pyforms_web.widgets.django.modelform.ModelFormWidget*

When a Pyforms application inherit from this class a form for the model ModelViewFormWidget.MODEL is created with all the fields in the fieldset in read only mode.

Usage example:

```
from suppliers.models import Order

class OrderView(ModelViewFormWidget):
     MODEL =  Order
     TITLE = 'Order in read-only'

     FIELDSETS = [
         'h3:General Information',
         ('responsible','order_req'),
         'supplier',
         {'a:Description':['order_desc'], 'b:Notes':['order_notes']},
         ('order_amount', 'currency', 'order_paymethod'),
         ('order_reqnum', 'order_reqdate'),
         ('order_podate', 'order_deldate')
     ]
```

Parameters

- **title** (*str*) – Title of the app. By default will assume the value in the class variable TITLE.
- **model** (*django.db.models.Model*) – Model with the App will represent. By default will assume the value in the class variable MODEL.
- **inlines** (*list(ModelAdminWidget)*) – Sub models to show in the interface
- **fieldsets** (*list(str)*) – Organization of the fields
- **parent_pk** (*int*) – Parent model key
- **parent_model** (*django.db.models.Model*) – Parent model class
- **pk** (*int*) – Model register to manage

CHAPTER

THIRTEEN

JAVASCRIPT

13.1 BaseWidget

class BaseWidget(*widget_id*, *widget_name*, *controls*, *parent_id*, *data*)

BaseWidget class handles the communication with the server python application.

Arguments

- **widget_id** (`string()`) – BaseWidget id.
- **widget_name** (`string()`) – BaseWidget name.
- **controls** (`list(ControlBase)()`) – List of the BaseWidget controls.
- **parent_id** (`string()`) – Parent widget id.
- **data** (`object()`) – Object instance with the application data.

`BaseWidget.`**activate_load_event**`()`

Event that check if the loading event should be on or off.

Returns **bool** – .

`BaseWidget.`**app_id**(*name*)

Returns the App id.

Returns **string** –

`BaseWidget.`**close**`()`

Close the app.

`BaseWidget.`**close_sub_apps**`()`

Close child apps

`BaseWidget.`**control_id**(*name*)

Returns the BaseWidget id.

Returns **string** –

`BaseWidget.`**deserialize**(*data*)

Deserializes the application from the data sent by the server.

Arguments

- **data** (`object()`) – Data to deserialize.

`BaseWidget.`**find_control**(*name*)

Returns a control.

Arguments

- **name** (string()) – Control name.

Returns **ControlBase** –

BaseWidget.**fire_event**(*dom_id*, *event*, *show_loading*)
Fire an application event.

Arguments

- **dom_id** (string()) – Python object name.
- **event** (string()) – Event name.
- **show_loading** (bool()) – Flag to activate the loading.

BaseWidget.**jquery**()
Get the jquery object.

Returns **jquery** – .

BaseWidget.**key_pressed**(*evt*)
Function called when the key is pressed.

BaseWidget.**loading**()
Activate the load.

BaseWidget.**not_loading**()
Deactivate the loading.

BaseWidget.**parent_widget**()
Returns the parent BaseWidget.

Returns **BaseWidget** –

BaseWidget.**query_server**(*params*)
Query the server.

Returns **object** – .

BaseWidget.**refresh_timeout_event**()
Function used to call the server refresh_event.

BaseWidget.**serialize**()
Serializes the application to send to the server.

Returns **object** – .

BaseWidget.**serialize_data**(*data*)
Serializes the application to send to the server.

Returns **object** – .

BaseWidget.**update_controls**()
Update controls with the values from the server.

13.2 Controls

Every Control implemented in Python as a **javascript class** responsible for handling the Control in the client side. All these controls inherit from the ControlBase class.

class ControlBase(*name*, *properties*)

ControlBase class implements the basic control functionalities.

Arguments

- **name** (string()) – Name of the control.
- **properties** (string()) – Properties of the control.

ControlBase.**after_init_control**()

Function called after the init_control function. It applies the controls most common configurations.

ControlBase.**app_id**()

Widget id.

Returns **string** – .

ControlBase.**apply_deserialization**(*data*)

Function called after the deserialization of the server data. It applies the controls most common configurations.

Arguments

- **data** (object()) – Data sent by the server.

ControlBase.**control_id**()

Control id.

Returns **string** – .

ControlBase.**deserialize**(*data*)

Deserialize the data from the server. This function is called at the initialization of the control and everytime data is received from the server.

Arguments

- **data** (object()) – Data sent by the server.

ControlBase.**disable**()

Disable the control.

ControlBase.**enable**()

Enable the control.

ControlBase.**get_value**()

Get the value of the control.

Returns **jquery** – .

ControlBase.**hide**()

Hide the control.

ControlBase.**init_control**()

Function called to initialize the control html and events.

ControlBase.**init_help**()
Get the help text as a tag.

Returns **string** – .

ControlBase.**jquery**()
JQuery object of the control html.

Returns **jquery** – .

ControlBase.**jquery_place**()
JQuery object of the div where the control is placed.

Returns **jquery** – .

ControlBase.**place_id**()
Div id where the control is placed.

Returns **string** – .

ControlBase.**serialize**()
Serializes the control data to send back to the server.

Returns **object** – .

ControlBase.**set_css**(*css*)
Sets the css classes for the control. Previously added classes will be removed.

Arguments

- **css** (string()) – String with the css classes to add.

ControlBase.**set_field_css**(*css*)
Sets the css classes for the field where the control is. Previously added classes will be removed.

Arguments

- **css** (string()) – String with the css classes to add.

ControlBase.**set_inline**()
Set the label inline.

Returns **string** – .

ControlBase.**set_label**(*value*)
Sets the label of the control.

Arguments

- **value** (string()) – Label to set.

ControlBase.**set_required**(*value*)
Sets the label of the control.

Arguments

- **value** (string()) – Label to set.

ControlBase.**set_value**(*value*)
Sets the value of the control.

Arguments

- **value** (object()) – Value to set.

ControlBase.**show**()
Show the control.

ControlBase.**update_server**()

Function called before the serialization. It is used to decide if the control should be updated in the server.

13.3 PyformsManager

Note: Located in the **pyforms.js** file.

class PyformsManager()

PyformsManager is the class than manages all the pyforms client side interaction.

PyformsManager.**add_app**(*app*)

Add an app to the manager.

Arguments

- **app** (*BaseWidget()*) – BaseWidget instance.

PyformsManager.**checker_loop**()

check what it do

PyformsManager.**close_layout_place**(*layout_position*)

Close a layout.

Arguments

- **layout_position** (int()) – Layout id.

PyformsManager.**find_app**(*app_id*)

Search an app using the id.

Arguments

- **app_id** (string()) – BaseWidget id.

PyformsManager.**find_control**(*control_id*)

Search for a control by its id.

Arguments

- **control_id** (string()) – Control id.

PyformsManager.**garbage_collector**()

Check if applications are still in use, if not remove them

PyformsManager.**open_application**(*app_data*)

Open an application.

Arguments

- **app_data** (object()) – App data.

PyformsManager.**query_server**(*basewidget*, *data2send*, *show_loading*)

Contact the server to submit and receive updates.

Arguments

- **basewidget** (*BaseWidget()*) – BaseWidget object.
- **data2send** (object()) – Object with the data to send.
- **show_loading** (bool()) – Flag to activate the app loading (optional).

PyformsManager.**register_checkloop**(*check_event*)
Check what it does

PyformsManager.**register_layout_place**(*place_id*, *place_generator*, *place_activator*, *place_closer*)
Register a new layout for the applications.

Arguments

- **place_id** (string()) – App id.
- **place_generator** (function()) – Function that will generate the html container for the applications.
- **place_activator** (function()) – Function to active the layout when it exists.
- **place_closer** (function()) – Function to destroy the layout.

PyformsManager.**remove_app**(*app_id*, *app_index*)
Remove an app from the manager. If the app_index parameter is not defined it will search for the index using the app_id parameter.

Arguments

- **app_id** (string()) – BaseWidget id.
- **app_index** (int()) – BaseWidget index (optional).

PyformsManager.**split_id**(*control_id*)
Parse the control id

Arguments

- **control_id** (string()) – Control id.

Returns **list(str)** – [widget_id, control_name]

13.4 Other functions

Note: Located in the **pyforms-hub.js** file.

error_msg(*msg*)
Function called to show a message. This function can be overwritten.

Arguments

- **msg** (string()) – Message.

run_application(*application*, *constructor_params*, *method_params*)
Function called to execute a pyforms application.

Arguments

- **application** (string()) – Full python module path of the Widget to be executed.
- **constructor_params** (object()) – Widget parameters.
- **method_params** (object()) – Functions and parameters to be executed after the Widget initialization.

pyforms_checkhash()
Function called to check if there is a hash command to be executed. Url hash can be used to open an application.

CHAPTER

FOURTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

D

E

F

G

H

N

O

P

Q

R

S

T

U

V

W