ObjectPack

Выпуск 2.0.20.4

BARS Group

июн. 18, 2017

Оглавление

Глава 1

Установка и настройка окружения

1. Установить зависимости:

```
pip install m3-core m3-ext3 objectpack django==1.4
```

2. Если django-проект еще не создан, то создаем и в `INSTALLED_APS` добавляем приложения:

```
INSTALLED_APPS = (

    ...
    'm3_ext',
    'm3_ext.ui',
    'objectpack',
)
```

3. #TODO: Подключить desktop view (м.б. сделать ссылку на m3-ext3)
4. Инициализировать контроллер:

```
# controller.py
from objectpack.observer import ObservableController, Observer

observer = Observer()
controller = ObservableController(url="actions", observer=observer)
```

5. #TODO: Расширить urlpatterns
6. PROFIT!

Глава 2

Подробно про ObjectPack

ObjectPack - это пак, который реализует основные CRUD операции для модели и содержит следующие экшены:

- *ObjectListWindowAction* - возвращает окно со списком объектов
- *ObjectRowsAction* - возвращает JSON-строки для окна со списком объектов
- *ObjectAddWindowAction* - возвращает ExtJS окно добавления нового объекта
- *ObjectEditWindowAction* - возвращает ExtJS окно редактирования объекта
- *ObjectSaveAction* - сохранение нового и обновление существующего объектов
- *ObjectDeleteAction* - удаляет объекты

Для того чтобы сконфигурировать свой пак, минимально требуется лишь указать модель, по которой он будет строиться. Так предыдущий пример можно было записать как:

```
# actions.py

class PersonPack(ObjectPack):

    model = Person

    # разрешим добавлять ссылку на list_window в меню Desktop'а
    add_to_menu = True
```

Всё по прежнему работает, но вместо колонок с полями модели в гриде отображается всего одна колонка “Наименование” и пропали кнопки Добавить/Редактировать/Удалить. Так мы получили простой список объектов.

Список объектов

Настройки колонок в окне со списком объектов хранятся в атрибуте `columns`. По умолчанию он имеет значение:

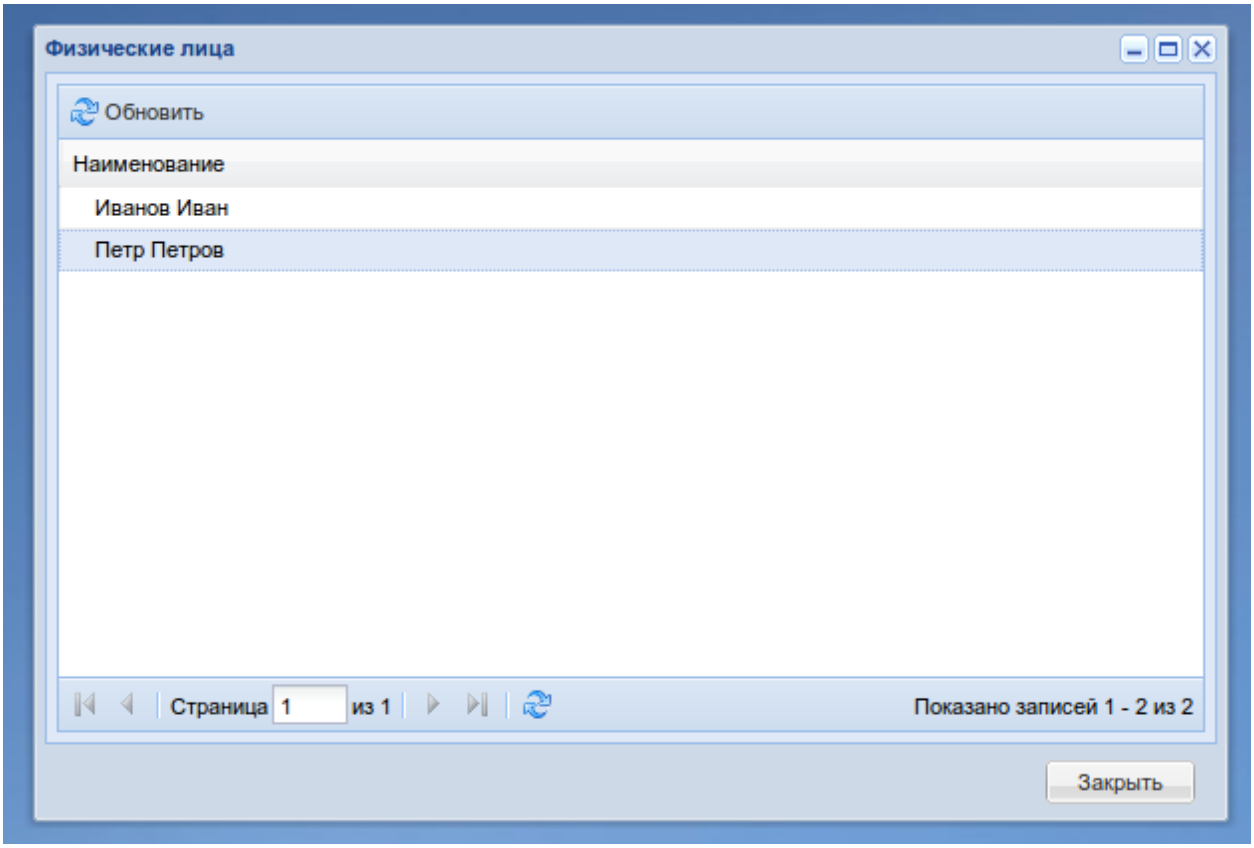

Рис. 2.1: List Window - Окно со списком объектов

```
columns = [
    {
        'data_index': '__unicode__',
        'header': u'Наименование',
    }
]
```

`columns` - это список словарей, где каждый словарь соответствует одной колонке. Колонки в гриде будут расположены в том же порядке.

data_index

Для задания колонки достаточно задать ключ *data_index*, значением которого могут быть атрибут объекта, property или callable объект, который можно вызвать без передачи аргументов. Так же можно получить доступ к атрибутам доступным через композицию, например `'userprofile.user.username'`.

prepare_row

Можно указать значение несуществующего атрибута. В *ObjectPack* есть метод *prepare_row*, который позвоялет установить дополнительные атрибуты в объект перед сериализацией в JSON:

```
# actions.py

class PersonPack(ObjectPack):

    model = Person

    columns = [
        {
            'data_index': '__unicode__',
            'header': u'Имя',
        },
        {
            'data_index': 'birthday',
            'header': u'Дата рождения',
        },
        {
            'data_index': 'is_adult',
            'header': u'Достигнул совершеннолетия',
        }
    ]

    def prepare_row(self, obj, request, context):
        today = datetime.date.today()
        is_adult = ((today - obj.birthday).days // 365 >= 18)
        obj.is_adult = '<div class="x-grid3-check-col%s"/>' % (
            '-on' if is_adult else '')
        return obj
```

Сортировка и поиск

Чтобы включить поиск и сортировку по колонке, нужно добавить в `columns`:

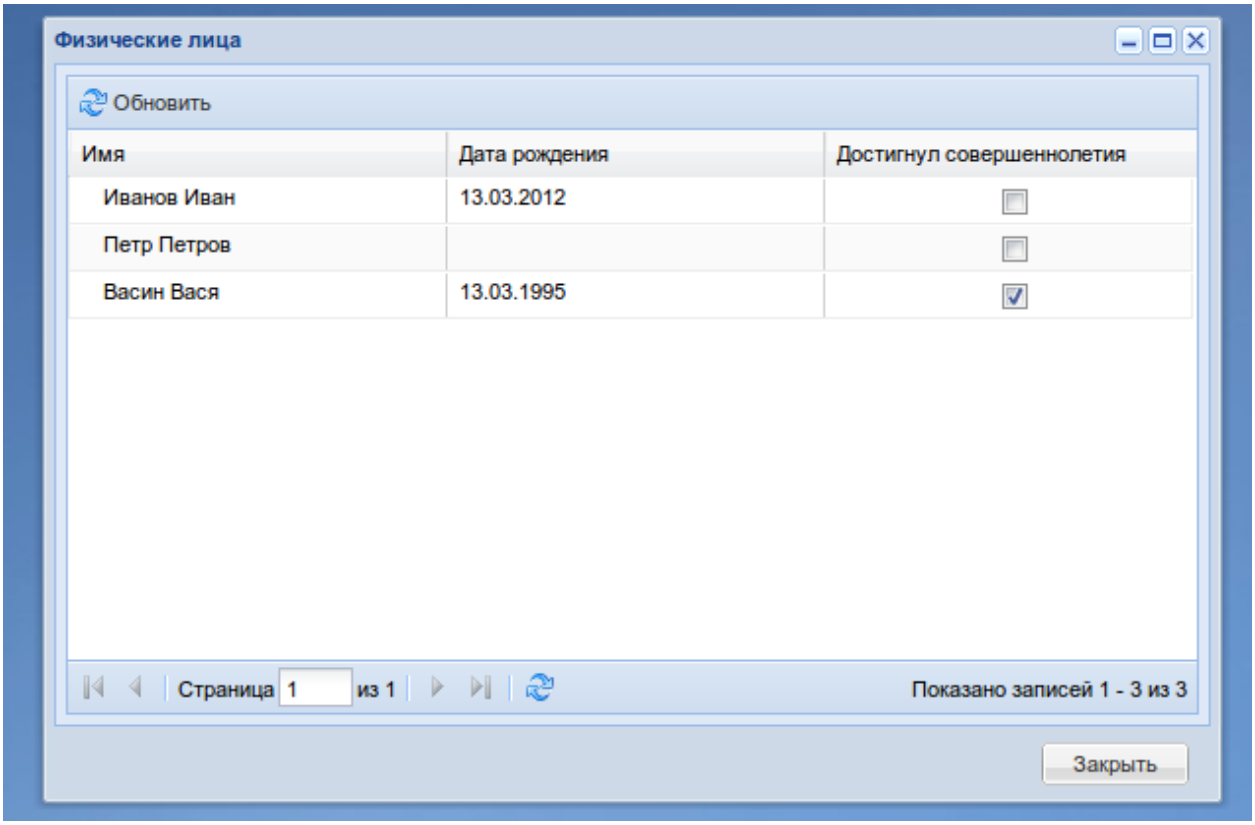

Рис. 2.2: Физ. лица достигнувшие совершеннолетие

```
columns = [

    {
        'data_index': '__unicode__',
        'header': u'Имя',
        'searchable': True,
        'search_fields': ('name', 'surname'),
        'sortable': True,
        # Сортировка сперва по имени, потом по фамилии
        'sort_fields': ('name', 'surname'),
    }

]
```

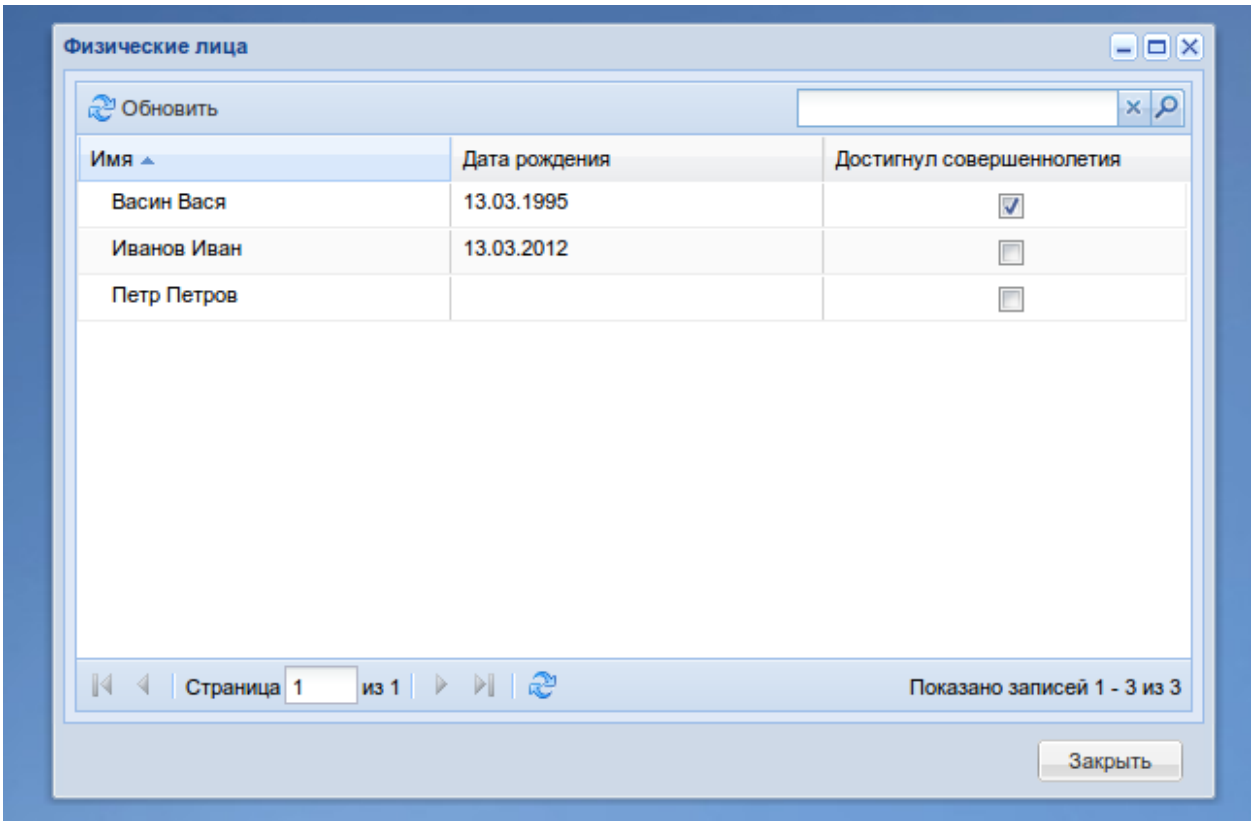

Рис. 2.3: Поиск и сортировка

Значениями по ключам `search_fields` и `sort_fields` должнен быть кортеж из лукапов полей django модели. Например:

```
{
    'search_fields': (
        'userprofile__user__username',
        'userprofile__person__name',
        'userprofile__person__surname')
}
```

Примечание: Если `data_index` колонки соотвествует полям модели, то ключи `search_fields` и

`sort_fields` можно опустить.

Установкой атрибута `list_sort_order` можно задать сортировку по умолчанию:

```
class PersonPack(ObjectPack):

    list_sort_order = ('name', 'surname')
```

Фильтрация на сервере

Часто бывает необходимо ограничить изначальную выборку данных. Для этого необходимо в паке перегрузить метод *get_rows_query*:

```
def get_rows_query(self, request, context):
    query = super(PersonPack, self).get_rows_query(request, context)
    query = query.filter(birthday__isnull=False)
    return query
```

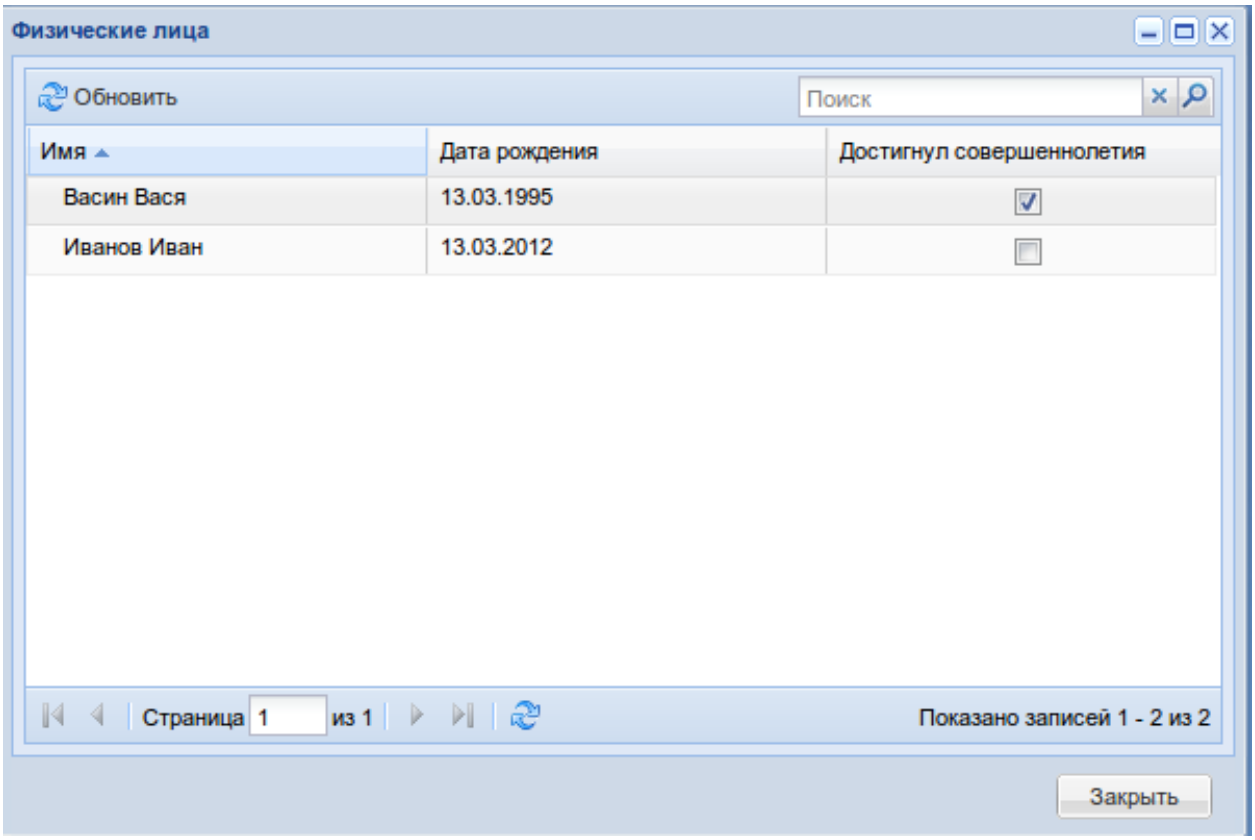

Рис. 2.4: Только физ. лица с указанной датой рождения

Колоночные фильтры

Иногда общей строки поиска по гриду бывает недостаточно и нужны отдельные фильтры по колонкам. В objectpack есть два вида колоночных фильтров: встроенные в контекстное меню заголовка колонки

и контролы расположенные непросредтвенно в заголовке. По умолчанию включен первый тип. Рассмотрим на примере:

```
class PersonPack(ObjectPack):

    model = Person

    columns = [
        {
            'data_index': '__unicode__',
            'header': u'Фамилия Имя',
            'width': 2,
            'filter': {
                'type': 'string',
                'custom_fields': ('name', 'surname')
            }
        },
        {
            'data_index': 'gender',
            'header': u'Пол',
            'width': 1,
            'filter': {
                'type': 'list',
                'options': model.GENDERS
            }
        },
        {
            'data_index': 'birthday',
            'header': u'Дата рождения',
            'width': 1,
            'filter': {
                'type': 'date',
            }
        }
    ]
```

```
from functools import partial
from objectpack.filters import ColumnFilterEngine, FilterByField


class PersonPack(objectpack.ObjectPack):

    model = models.Person

    filter_engine_clz = ColumnFilterEngine

    f = partial(FilterByField, model)

    columns = [
        {
            'data_index': '__unicode__',
            'header': u'Фамилия Имя',
            'width': 2,
            'filter': (
                f('name', 'name__icontains')
                & f('surname', 'surname__icontains')
            )
        },
```

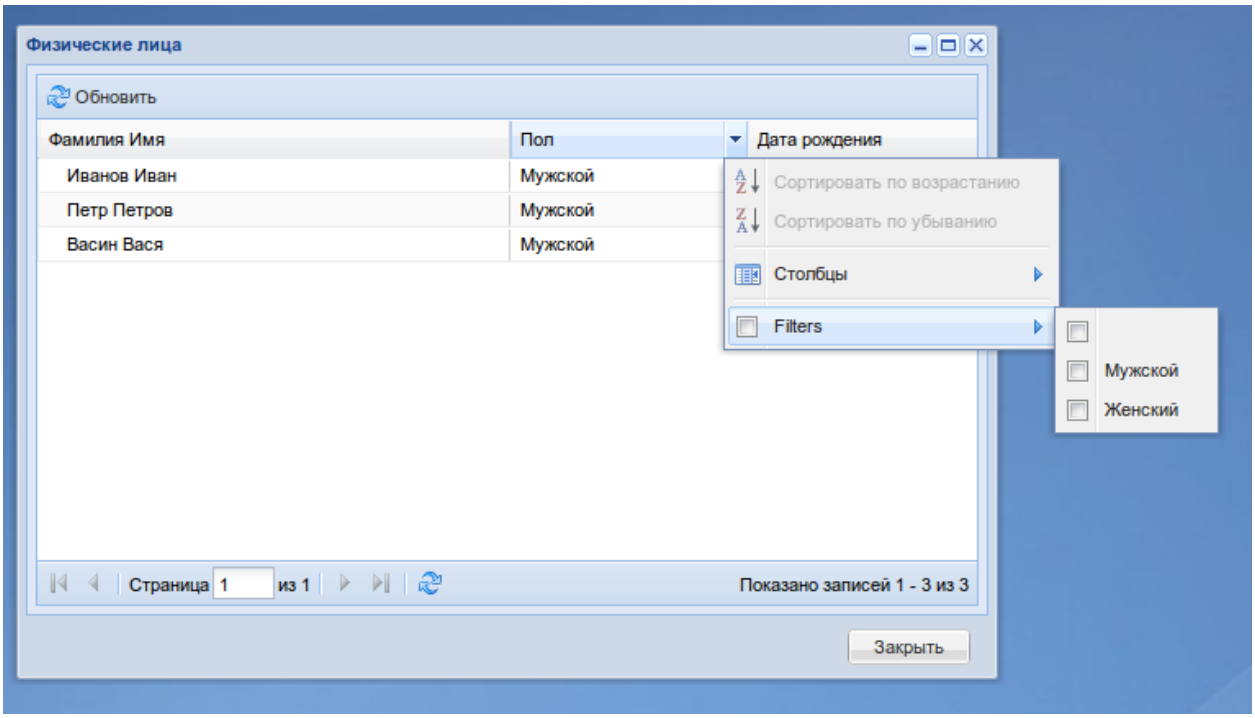

Рис. 2.5: Фильтр встроенный в контекстное меню

```
        {
            'data_index': 'gender',
            'header': u'Пол',
            'width': 1,
            'filter': f('gender')
        },
        {
            'data_index': 'birthday',
            'header': u'Дата рождения',
            'width': 2,
            'filter': (
                f('birthday', 'birthday__gte', tooltip=u'С')
                & f('birtday', 'birthday__lte', tooltip=u'По')
            )
        }
    ]
```

Окно со списком объектов

По умолчанию в качестве окна со списком объектов используется *BaseListWindow*. Отнаследовавшись от него можно конфигурировать свои окна со списками или можно перегрузить методы пака *create_list_window* и *get_list_window_params*.

Создание объекта

Теперь добавим в наш справочник возможность создавать новые объекты.

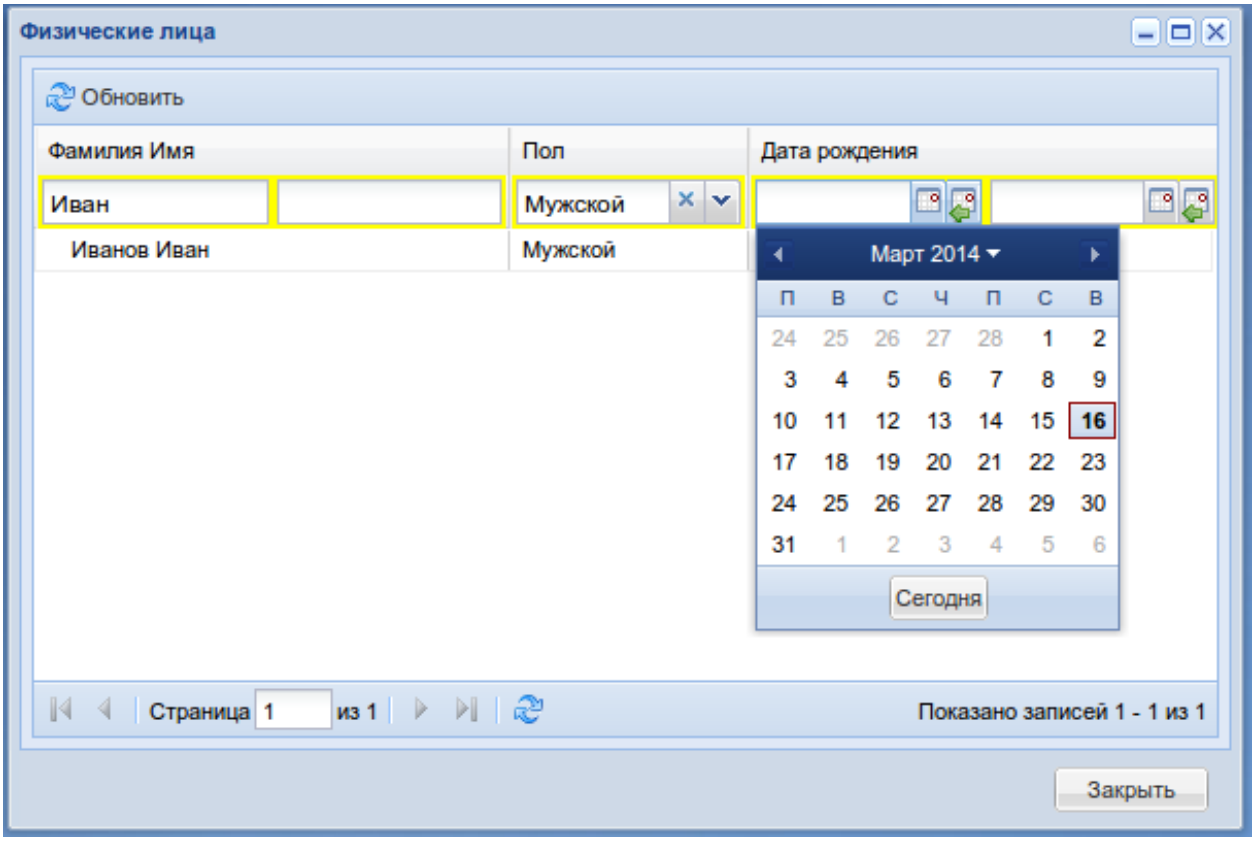

Рис. 2.6: Колоночный фильтр

Окно добавления

Для этого необходимо установить в атрибут `add_window` класс окна. Это может быть любой класс унаследованный от *BaseEditWindow*.

Этот класс реализует каркас для окна и предоставляет некоторый интерфейс, который следует соблюдать:

```
from objectpack.ui import BaseEditWindow, make_combo_box
from m3_ext.ui import all_components as ext

from models import Person


class PersonAddWindow(BaseEditWindow):

    def _init_components(self):
        """
        Здесь следует инициализировать компоненты окна и складывать их в
        :attr:`self`.
        """
        super(PersonAddWindow, self)._init_components()

        self.field__name = ext.ExtStringField(
            label=u'Имя',
            name='name',
            allow_blank=False,
            anchor='100%')

        self.field__surname = ext.ExtStringField(
            label=u'Фамилия',
            name='surname',
            allow_blank=False,
            anchor='100%')

        self.field__gender = make_combo_box(
            label=u'Пол',
            name='gender',
            allow_blank=False,
            anchor='100%',
            data=Person.GENDERS)

        self.field__birthday = ext.ExtDateField(
            label=u'Дата рождения',
            name='birthday',
            anchor='100%')

    def _do_layout(self):
        """
        Здесь размещаем компоненты в окне
        """
        super(PersonAddWindow, self)._do_layout()
        self.form.items.extend((
            self.field__name,
            self.field__surname,
            self.field__gender,
            self.field__birthday,
        ))
```

```
    def set_params(self, params):
        """
        Установка параметров окна

        :params: Словарь с параметрами, передается из пака
        """
        super(PersonAddWindow, self).set_params(params)
        self.height = 'auto'
```

Теперь скажем паку какое окно нужно использовать:

```
class PersonPack(ObjectPack):

    model = Person

    add_window = ui.PersonAddWindow

    ...
```

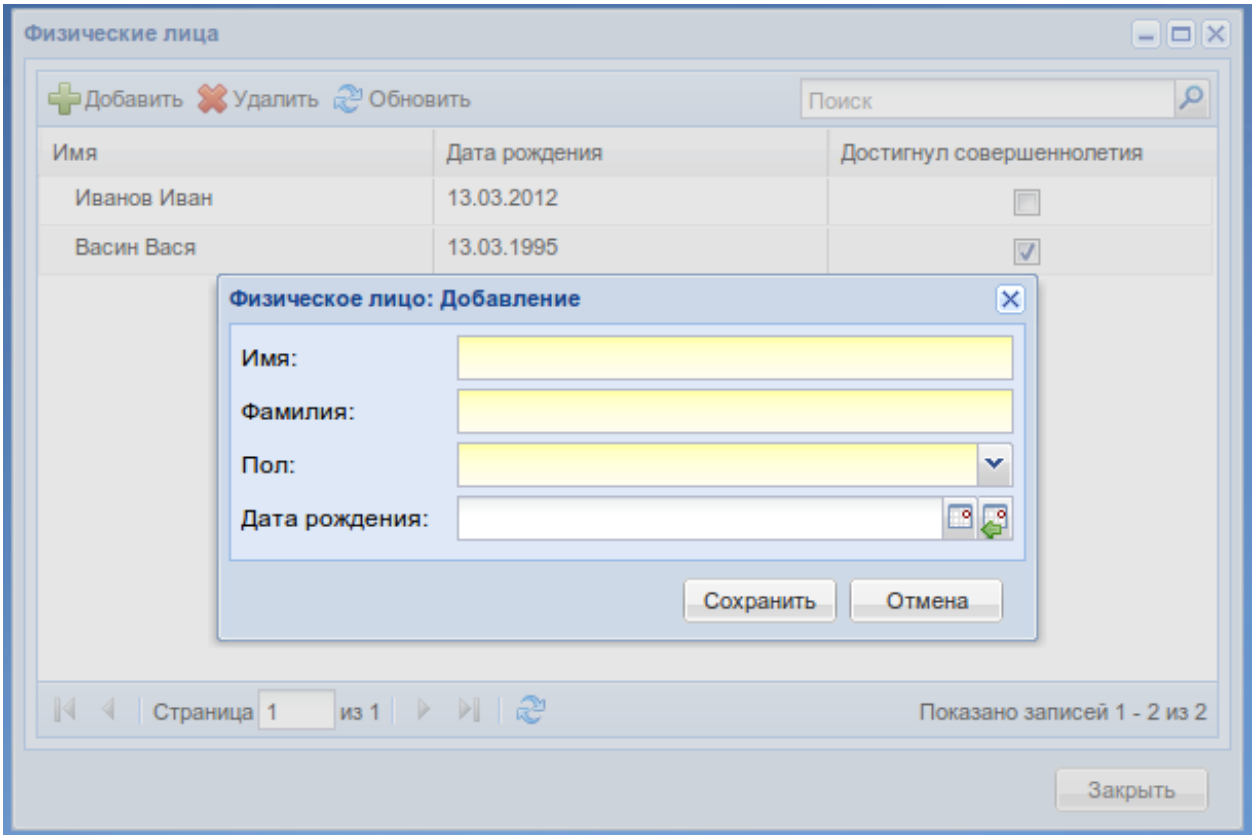

Рис. 2.7: Окно создания физ. лица

Генерация окон

Описыние компонент окна занятие утомительное и скушное. К счастью в objectpack есть убер-фича - генерация окон редактирования для модели. Так окно из предыдущего примера полностью идентично слудующему:

```
from objectpack import ModelEditWindow

add_window = ModelEditWindow.fabricate(model=Person)
```

Тонкая настройка окон

Часто бывает нужно дополнительно сконфигурировать окно, особенно это актуально в случае с генерированными окнами. Для этого удобно использовать два метода в паке: *create_edit_window* и *get_edit_window_params*

Редактирование

Теперь добавим возможность редактировать объекты. Для этого нужно паку задать атрибут `edit_window`. В нашем случае окно редактирования идентично окну создания, поэтому мы пишем:

```
add_window = edit_window = ModelEditWindow.fabricate(model=Person)
```

Окно редактирование может быть сложным, например, когда у модели есть зависимые модели. В таких случаях можно использовать окно с вкладками *TabbedWindow*.

Конфигурирование окна осуществляется *так же* как и для окна создания.

Сохранение

ObjectSaveAction будет доступен в паке после задания либо окна создания, либо окна редактирования объекта.

При сохранении значения из формы окна добавления/редактирования сопоставляются с полями модели по атрибутам `name` элементов формы.

Непосредственное сохранение объекта модели происходит в методе *save_row*. Перегрузив этот метод можно дополнительно управлять сохранением объекта:

```
def save_row(self, obj, create_new, request, context):
    if not (obj.name.isalpha() and obj.surname.isalpha()):
        raise ApplicationLogicException(
            u'Имя и Фамилимя могут содержать только буквы алфавита!')
    super(PersonPack, self).save_row(obj, create_new, request, context)
```

Удаление

За удаление объекта отвечает атрибут `can_delete`, который может принимать три значения: `True`, `False` или `None`. По умолчанию `None`.

Если установлено значение `None`, то *ObjectDeleteAction* будет добавлен в пак если задоно либо окно добавления, либо окно редактирования. `True` удаление возможно и `False` - не возможно:

```
class PersonPack(ObjectPack):

    model = Person

    can_delete = True
```

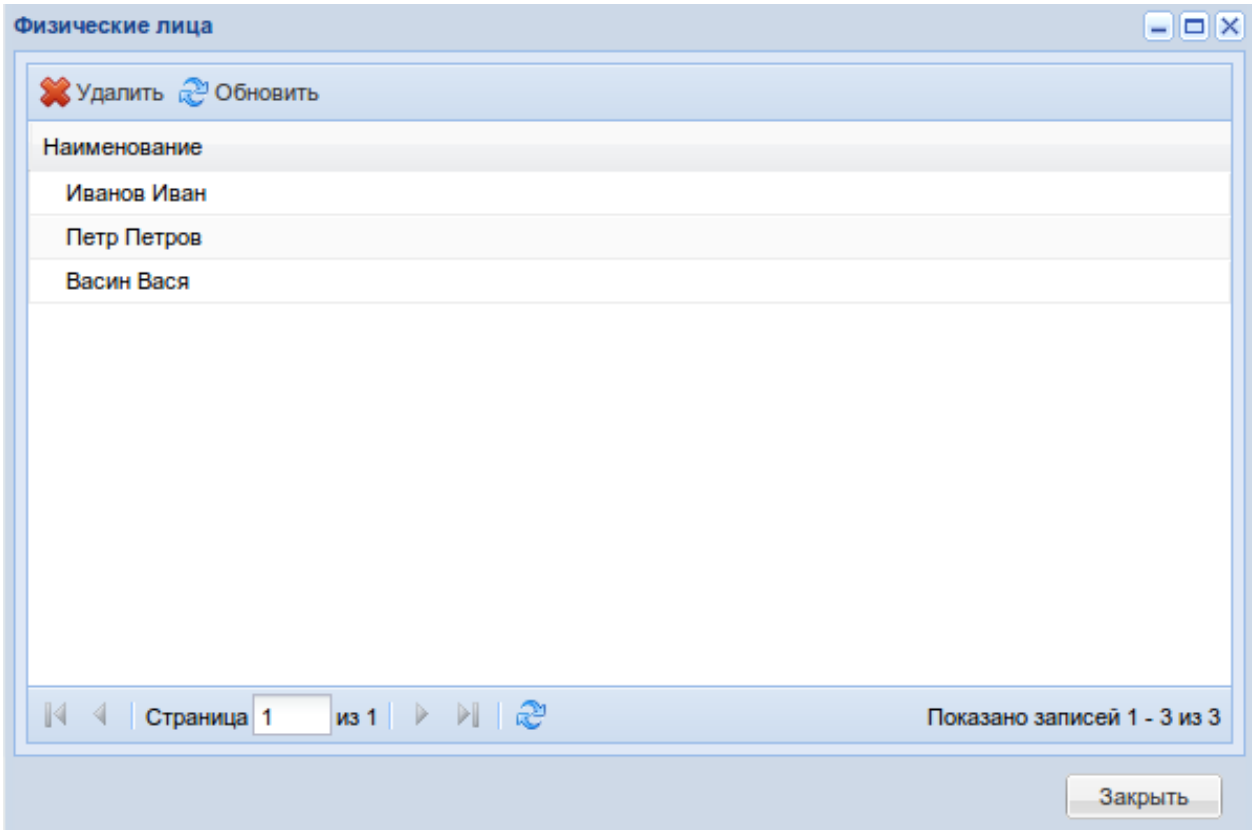

Рис. 2.8: Простой список с можностью удаления

Само удаление объекта модели происходит в методе *delete_row*. По умолчанию тут вызывается метод `safe_delete` модели и, если он не определен, вызывается функция `m3.db.safe_delete()`. Перегрузив его можно управлять удалением объекта:

```
def delete_row(self, obj_id, request, context):
    if date.today().weekday() in (5, 6):
        raise ApplicationLogicException(
            u'Нельзя удалять записи в выходные дни!')

    # не хотим использовать m3.db.safe_delete
    obj = self.model.objects.get(id=obj_id)
    obj.delete()
    return obj
```

Глава 3

Контроллер, наблюдатель и точки расширения

В качестве контроллера в ObjectPack используется *ObservableController*. Особенностью этого контроллера является то, что при регистрации в нём экшена, последний в свою очередь добавляется в реестр слушателей наблюдателя *Observer*.

Observer

Observer позволяет регистрировать в экшенах точки расширения, а также добавляет в каждый экшен две точки расширения *before* и *after*, которые действуют как `m3.actions.Action.pre_run` и `m3.actions.Action.post_run`, но выполняются соответственно до и после них, т.е. если методы *before* и *after* вернут какой-либо результат ActionResult, то результатом выполнения экшена будет он.

Рассмотрим на примере из `objectpack.demo`:

```
# создаём наблюдателя
obs = observer.Observer()
    #logger=logger, verbose_level=observer.Observer.LOG_MORE)

# создаём контроллер
action_controller = observer.ObservableController(obs, "/controller")
```

Далее создаем слушателя, который описывается классом с одним обязательным атрибутом *listen*:

```
@obs.subscribe
class Listener(object):

    # список регулярок, для сопоставления экшенам
    listen = ['.*/.*/ObjectListWindowAction']

    def after(self, request, context, response):
        response.data.title = u'My-xa-xa! %s' % response.data.title
```

Так мы подменили текст заголовка окна, метод *after* слушателя будет вызван после post_run экшена.

Примечание: response - это по сути ActionResult, а мы помним что, ExtUIScriptResult в атрибуте data хранит ExtJS компонент, в данном случае это будет объект окна *objectpack.ui.BaseListWindow*.

Помимо *before* и *after* в экшенах ObjectPack'a, зарегистрировано множество полезных точек расширения, например *prepare_obj* для *objectpack.actions.ObjectRowsAction*, которая делает тоже что и *objectpack.actions.ObjectPack.prepare_row*, только *request* и *context* здесь будут аттрибутами слушателя:

```
@obs.subscribe
class StarToHash(object):

    listen = ['.*/BandedColumnPack/.*']

    def prepare_obj(self, obj):
        obj['field1'] = obj.get('field1', False) and (obj['id'] % 2)
        return obj
```

Ниже приведен полный перечень точек расширения для ObjectPack, но ничего не мешает нам зарегистрировать свои:

```
class DoSomethingAction(objectpack.BaseAction):

    def run(self, request, context):
        message = 'Done'
        self.handle('do_well', message)
        return OperationResult(message=message)

@obs.subscribe
class DoSomethingListener(object):

    listen = ['.*/.*/DoSomethingAction']

    def do_well(self, message):
        message = 'Well Done!'
        return message
```

Результатом выполнения этого экшена будет информационное окошко с текстом *Well Done!*

Примечание: Если слушатели пишутся в одном приложении рядом с экшенами, то проще подключать их через декоратор. В случае если слушателей нужно подключить в другом модуле или в другом приложении, то лучше вынести их в отдельный модуль listeners.py и выполнить их регистрацию в *app_meta.register_action*. Регистрировать можно либо через импорт модуля, если вы используете декоратор, или вызовом функции, которая будет подписывать слушателей в Observer

Когда могут понадобиться точки расширения?

Через точки расширения удобно делать проверки прав доступа и различных условий бизнес логики, тем самым можно разгрузить код экшена и делегировать эти проверки слушателю. Так же через точки расширения можно реализовать механизм плагинов.

Доступные точки расширения

Action	Точка расширения	Тип переда
	query	Выборка д
Следующие три точки технически ничем не отличается от query, но были вынесены отдельно, чтобы не наруша		
Выборка данных QuerySet	Поиск по выборке	
ObjectRowsAction	apply_filter	Выборка д
apply_sort_order	Выборка данных QuerySet	Сортировк
Список строк для сериализации в JSON	Манипуляция с готовым с сериалиазации списком	
prepare_obj	Объект модели	Манипуля сериализа установка
	row_editing	Кортеж с редактиро (Успешно/ ошибки/N
TODO: пока нет		
ObjectSaveAction	save_obj Объект модели	Обработка должен во *AlreadySa* сохранён
	set_window_params	Словарь с

Таблица 3.1 – продолжение

Action	Точка расширения	Тип переда
Манипуляции с компонентом окна (добавление/удаление/редактирование различных элементов, установка параметров и т.д и т.п.		create_wi
TODO: пока нет	*ObjectListWindowAction*	
before	Принимает в аргументах request, context	Выполняет
	after	
Выполняется после post_run экшена		

Глава 4

objectpack API Reference

actions Module

Created 23.07.2012

Author pirogov

Этот модуль содержит главный класс библиотеки и набор actions для него

class objectpack.actions.BaseAction

Базовые классы: m3.actions.Action

Базовый класс для всех actions.

Имеет автоматически-генерируемый url и возможность подключений точек расширения в *Observer*.

context_declaration()

Делегирует декларацию контекста в пак

Результат Правила для DeclarativeActionContext

Тип результата dict

get_perm_code(*subpermission=None*)

Возвращает код права

Параметры subpermission (*str*) – Код подправа доступа

Результат code - Код доступа

Тип результата str

static handle(*verb*, *arg*)

Заглушка для точек расширения. При регистрации в обсервер перекрывается

Параметры

- verb (*str*) – Имя точки расширения

- arg (*any*) – Объект для передачи в точку расширения

Return arg Тот же объект или любой другой

need_check_permission
Небходимость проверки прав

Если определен perm_code, то необходимость проверки прав будет зависеть от присутствия perm_code среди sub_permissions пака и соответствующего флага пака

Тип результата bool

perm_code = **None**
Код подправа, используемый при формировании кода права экшна стандартным способом. Если код не указан - экшн формирует свой код права независимо от пака

url
автоматически генерируемый url

class objectpack.actions.BasePack

Базовые классы: m3.actions.ActionPack

Потомок ActionPack, реализующий автогенерацию short_name, url

classmethod absolute_url()
Получение url для построения внутренних кэшей m3

declare_context(*action*)
Декларация контекста для экшна

```
def declare_context(self, action):
    if action is self.do_right_things_action:
        return {
            'things': {'type': 'list'},
            'done': {'type': 'boolean'}
        }
```

url
Относительный url пака

class objectpack.actions.BaseWindowAction

Базовые классы: *objectpack.actions.BaseAction*

Базовый Action показа окна

configure_window()
Точка расширения, предоставляющая доступ к настроенному экземпляру окна для тонкой настройки.

Примечание: Оставлена для особо тяжёлых случаев, когда не удаётся обойтись set_params

---

```
def configure_window(self):
    self.win.grid.top_bar.items[8].text = _(u'Ух ты, 9 кнопок')
```

create_window()
Метод инстанцирует окно и помещает экземпляр в атрибут self.win

```
def create_window(self):
    self.win = EditWindow()
```

run(*request*, *context*)
Тело Action, вызывается при обработке запроса к серверу.

Параметры

- request (*django.http.HttpRequest*) – Request
- context (*m3.actions.context.DeclarativeActionContext*) – Context

---

Примечание: Обычно не требует перекрытия

---

set_window_params()
Метод заполняет словарь self.win_params, который будет передан в окно. Этот словарь выступает как шина передачи данных от Actions/Packs к окну

```
def set_window_params(self):
    self.win_params['title'] = _(u'Привет из ада')
```

set_windows_params()

Deprecated

TODO Выпилить, ато опечатка портит всю семантику!

class objectpack.actions.ObjectAddWindowAction

Базовые классы: *objectpack.actions.ObjectEditWindowAction*

Базовый Action показа окна добавления объекта.

Примечание: Отдельный action для уникальности short_name

---

perm_code = **'add'**

class objectpack.actions.ObjectDeleteAction

Базовые классы: *objectpack.actions.BaseAction*

Действие по удалению объекта

audit(*obj*)
Обработка успешно удалённых объектов

delete_obj(*id_*)
Удаление объекта по идентификатору @id_

Параметры id – Идентификатор объекта

delete_objs()
Удаляет обекты по ключам из контекста

perm_code = **'delete'**

run(*request*, *context*)

try_delete_objs()
Удаляет обекты и пытается перехватить исключения

Except m3.RelatedError, django.db.utils.IntegrityError:

Raise m3.ApplicationLogicException

class objectpack.actions.ObjectEditWindowAction

Базовые классы: *objectpack.actions.BaseWindowAction*

Базовый Action показа окна редактирования объекта.

`create_window()`

`perm_code` = **'edit'**

`set_window_params()`

class `objectpack.actions.ObjectListWindowAction`

Базовые классы: *objectpack.actions.BaseWindowAction*

Базовый Action показа окна списка объектов

`create_window()`

`is_select_mode` = **False**
Режим показа окна (True - выбор, False - список)

`perm_code` = **'view'**
Код доступа

`set_window_params()`

class `objectpack.actions.ObjectMultiSelectWindowAction`

Базовые классы: *objectpack.actions.ObjectSelectWindowAction*

Базовый Action показа окна списка выбора нескольких объектов из списка

`create_window()`

class `objectpack.actions.ObjectPack`

Базовые классы: *objectpack.actions.BasePack*, `m3.actions.interfaces.IMultiSelectablePack`

Пак с экшенам, реализующими специфичную для работы с моделью действиями по добавлению, редактированию, удалению (CRUD actions)

Примечание: Можно из пака включить добавление элементов в главное меню или на десктоп extjs. По умолчанию эта опция выключена

```
add_to_desktop = True
add_to_menu = True
```

Если методы extend_menu/extend_desktop не реализованы, меню будет расширяться на основе title и get_default_action

Методы extend_X приоритетны

```
def extend_menu(self, menu):
    """
    Расширение главного меню.
    """
    return (
        # добавление пунктов в меню "справочники"
        menu.dicts(
            menu.Item(u'Dict 1', self),
            menu.SubMenu(u'Dict SubMenu',
                menu.Item(u'Dict 2', self.some_action),
            ),
```

```
        ),
        # добавление пунктов в меню "реестры"
        menu.registries(
            menu.Item(u'Reg 1'),
            menu.SubMenu(u'Regs SubMenu',
                menu.Item(u'Reg 2'),
            ),
        ),
        # добавление пунктов в меню "администрирование"
        menu.administry(
            menu.Item(u'Admin item 1')
        ),

        # добавление пунктов в "корень" меню
        menu.Item(name=u'item 1', self.some_action),

        # добавление подменю в "корень" меню
        menu.SubMenu(u'SubMenu',
            menu.Item(u'Item 2', self.some_action),
            menu.SubMenu(u'SubSubMenu',
                menu.Item(u'Item 3', self.some_action),
            ),
        ),
    )
```

Пустые подменю автоматически “схлопываются” (не видны в Главном Меню)

```
def extend_desktop(self, desk):
    """
    Расширение Рабочего Стола
    """

    return (
       desk.Item(u'Ярлык 1', pack=self.list_action),
       ...
    )
```

Любой из элементов можно отключить вернув вместо него None. Например:

```
desk.Item(u'Name', pack=self) if some_condition else None
```

MSG_DOESNOTEXISTS = **u’\u0417\u0430\u043f\u0438\u0441\u044c \u043d\u0435 \u043d\u0430\u0439\u0**

add_window = **None**
Окно для добавления элемента справочника

allow_paging = **True**
Включить пагинацию

apply_default_sort_order(*query*)

Параметры query (*django.db.models.query.QuerySet*) –

Результат Выборка, отсортированная по-умолчанию

Тип результата django.db.models.query.QuerySet

Примечание: Обычно не требует перекрытия

apply_filter(*query*, *request*, *context*)
Применяет фильтрацию к выборке query

Параметры

- query (*django.db.models.query.QuerySet*) –
- request (*django.http.HttpRequest*) –
- context (*m3.actions.context.DeclarativeActionContext*) –

Результат Отфильтрованная выборка query

Тип результата django.db.models.query.QuerySet

Примечание: Обычно не требует перекрытия

apply_search(*query*, *request*, *context*)
Возвращает переданную выборку отфильторованной по параметрам запроса

Параметры query (*django.db.models.query.QuerySet*) –

Результат

Тип результата django.db.models.query.QuerySet

Примечание: Обычно не требует перекрытия

apply_sort_order(*query*, *request*, *context*)
Возвращает переданную выборку отсортированной по параметрам запроса

Параметры query –

Результат

Тип результата

Примечание: Обычно не требует перекрытия

can_delete = **None**
Флаг разрешающий/запрещающий удаление. Если None, то удаление возможно, при наличии add_window/edit_window

column_constructor_fabric(*config*, *ignore_attrs=None*)
Фабрика колонок по данным атрибута 'columns'

Примечание: callable-объект, возвращающий объект с методом 'configure_grid(grid)'

column_name_on_select = **'__unicode__'**
Поле/метод, предоставляющее значение для отображения в DictSelectField

Внимание: ПОКА НЕ РАБОТАЕТ извлечение вложенных полей - конфликт с ExtJS

columns = **[{'header': u'\u041d\u0430\u0438\u043c\u0435\u043d\u043e\u0432\u0430\u043d\u0438\u\u**
Список колонок для добавления в грид, data_index - поле или метод модели

Важно: Для корректной работы полей выбора, необходима колонка с data_index = '__unicode__'

```
columns = [
    {
        'data_index: '__unicode__',
        'hidden': True,
    },
    {
       'data_index':'',
       'width':,
       'header':u'',
       'serchable':True,
       'sortable':True,
       'sort_fields': ('foo','bar'),
    },
    {
       'header':u'Группирующая Колонка 1',
       'columns': [
           {
               'data_index':'school.name',
               'width':200,
               'header':u'Колонка 1',
               'searchable':True,
               'search_fields': ('school.fullname',),
           },
       ]
    },
    {
       'data_index':'school.parent.name',
       'width':200,
       'header':u'Родитель',
       'renderer':'parent_render'
    },
]
```

configure_grid(*grid*)
Конфигурирует grid для работы с этим паком, создает колонки и задает экшены

Подсказка: Удобно использовать в окнах в комбинированных справочниках с несколькими гридами

```
class RightThingsWindow(objectpack.BaseWindow):

    def _init_components(self):
        super(RightThingsWindow, self)._init_components()
        ...
        self.right_things_todo_grid = ext.ExtObjectGrid()
```

```
        self.right_things_done_grid = ext.ExtObjectGrid()

    def _do_layout(self):
        ...

    def set_params(self, params):
        super(RightThingsWindow, self).set_params(params)
        ...
        get_pack_instance('RightThingsTodoPack').configure_grid(
            self.right_things_todo_grid)
        get_pack_instance('RightThingsDonePack').configure_grid(
        self.right_things_done_grid)
```

Параметры `grid` (*m3_ext.ui.panels.grids.ExtObjectGrid*) – Грид

`create_edit_window`(*create_new*, *request*, *context*)
Получить окно редактирования / создания объекта

Параметры

- `create_new` (*bool*) – Признак добавления или редактирования
- `request` (*django.http.HttpRequest*) – Запрос
- `context` (*m3.actions.context.DeclarativeActionContext*) – Контекст

Результат Окно добавления/редактирования

Тип результата *objectpack.ui.BaseEditWindow*

Подсказка: Удобно использовать для добавления/конфигурирования кастомных контролов в окно

```
def create_edit_window(self, create_new, request, context):
    win = super(RightThingsPack, self).create_edit_window(
        create_new, request, context)
    win.top_bar.btn_do_right_thing
```

`create_list_window`(*is_select_mode*, *request*, *context*)
Получить окно списка / выбора объектов

Параметры

- `is_select_mode` (*bool*) – Режим показа окна (True - выбор, False - список)
- `request` (*django.http.HttpRequest*) – Запрос
- `context` (*m3.actions.context.DeclarativeActionContext*) – Контекст

Результат Окно списка/выбора объектов

Тип результата *objectpack.ui.BaseListWindow*

`declare_context`(*action*)
Декларирует контекст для экшна

Параметры `action` (*objectpack.BaseAction*) – Экземпляр экшена

Результат Правила для декларации контекста DeclarativeActionContext

Тип результата dict

delete_action = **None**
Экшен удаления объектовы

delete_row(*obj_id*, *request*, *context*)
Удаляет объект по :obj_id Возвращает удалённый объект - т.е. объект модели, который уже не представлен в БД

Параметры obj_id (*int*) – pk объекта

Результат Удалённый объект

Тип результата django.db.models.Model

edit_window = **None**
Окно для редактирования элемента справочника

edit_window_action = **None**
Экшен показа окна редактирования объекта

filter_engine_clz
Класс конструктора фильтра для грида

Примечание: Подробнее смотри в objectpack.demo

псевдоним класса MenuFilterEngine

format_window_title(*action*)
Возвращает отформатированный заголовка окна. Заголовок примет вид “Модель: Действие”

Подсказка: Например “Сотрудник: Добавление”

Параметры action (*unicode*) – Действие характеризующее экшен

Результат Заголовок окна

Тип результата unicode

get_autocomplete_url()
Возвращает адрес для запроса элементов, подходящих введенному в поле тексту

Результат url экшена

Тип результата str

get_default_action()
Возвращает действие по умолчанию (действие для значка на раб.столе/пункта меню)

Примечание: Используется пи упрощенном встраивании в UI (add_to_XXX=True)

Результат Экземпляр экшена

Тип результата objectpack.BaseAction

get_display_dict(*key*, *value_field='id'*, *display_field='name'*)

get_display_text(*key*, *attr_name=None*)
Возвращает отображаемое значение записи (или атрибута attr_name) по ключу key

Параметры

- key (*basestring or int*) – ID объекта
- attr_name (*str*) – Имя атрибута модели

Результат Отображаемое текстовое представление объекта

Тип результата basestring

get_edit_url()
Возвращает адрес формы редактирования элемента справочника

Результат url экшена показа окна редактирования объекта

Тип результата str

get_edit_window_params(*params*, *request*, *context*)
Возвращает словарь параметров, которые будут переданы окну редактирования

```
def get_edit_window_params(self, params, request, context):
    params = super(RightThingsPack, self).get_edit_window_params(
        params, request, context)
    params.update({
        'user': request.user,
        'height': 800,
        'width': 600,
    })
    return params
```

Параметры

- params (*dict*) – Словарь параметров
- request (*django.http.HttpRequest*) – Запрос
- context (*m3.actions.context.DeclarativeActionContext*) – Контекст

Результат Словарь параметров

Тип результата dict

get_filter_plugin()
Возвращает плагин фильтрации

Результат js-код с плагином фильтрации

Тип результата basestring

get_list_url()
Возвращает адрес формы списка элементов справочника.

Примечание: Используется для присвоения адресов в прикладном приложении

Результат url экшена показа окна со списком объектов

Тип результата str

get_list_window_params(*params*, *request*, *context*)
Возвращает словарь параметров, которые будут переданы окну списка

```
def get_list_window_params(self, params, request, context):
    params = super(RightThingsPack, self).get_list_window_params(
        params, request, context)
    params.update({
        'title': u'Right things done by user: %s'
        % request.user.username,
        'height': 800,
        'width': 600,
    })
    return params
```

Параметры

- `params` (*dict*) – Словарь параметров
- `request` (*django.http.HttpRequest*) – Запрос
- `context` (*m3.actions.context.DeclarativeActionContext*) – Контекст

Результат Словарь параметров

Тип результата dict

`get_multi_select_url()`
Возвращает адрес формы выбора из списка элементов справочника.

Примечание: Используется для присвоения адресов в прикладном приложении

Результат url экшена показа окна выбора из списка объектов

Тип результата str

`get_not_found_exception()`
Возвращает класс исключения 'объект не найден'

Результат Класс исключения модели django

Тип результата django.core.exceptions.ObjectDoesntExist

`get_obj(request, context)`
Получает id объекта из контекста и возвращает кортеж (объект модели, create_new), где create_new признак создания или редактирования

Результат (Объект модели self.model, create_new)

Тип результата tuple

`get_row(row_id)`
Функция возвращает объект по :row_id Если id нет, значит нужно создать новый объект

Примечание: Используется в ExtDictSelectField'ax

Параметры `row_id` (*int*) – id объекта

Результат Объект модели self.model

Тип результата django.db.models.Model

get_rows_query(*request*, *context*)
Возвращает выборку из БД для получения списка данных

Параметры

- request (*django.http.HttpRequest*) – Запрос
- context (*m3.actions.context.DeclarativeActionContext*) – Контекст

Результат Кварисет

Тип результата django.db.models.query.QuerySet

```
def get_rows_query(self, request, context):
    query = super(RightThingsDonePack, self).get_rows_query(
        request, context)
    return query.filter(done=True)
```

get_rows_url()
Возвращает адрес, по которому запрашиваются элементы грида

Результат url экшена с данными для грида

Тип результата str

get_search_fields(*request=None*, *context=None*)

Параметры

- request (*django.http.HttpRequest*) –
- context (*m3.actions.context.DeclarativeActionContext*) –

Результат Список значений ‘data_index’ из колонок self.columns, по которым будет производиться поиск

Тип результата list

Примечание: Обычно не требует перекрытия

---

get_select_url()
Возвращает адрес формы выбора из списка элементов справочника.

Примечание: Используется для присвоения адресов в прикладном приложении

Результат url экшена показа окна выбора из списка объектов

Тип результата str

get_sort_order(*data_index*, *reverse=False*)
:param data_index :type data_index: str :param reverse: Обратный порядок :type reverse: bool :return: Ключи сортировки для указанного data_index :rtype: list or tuple

---

Примечание: Обычно не требует перекрытия

---

handle_row_editing(*request*, *context*, *data*)
Метод принимает данные из редактируемого грида и возвращает результат редактирования кортежем вида (удачно/неудачно, “сообщение”/None) :param

request: :type request: django.http.HttpRequest :param context: :type context: m3.actions.context.DeclarativeActionContext :param data: :type data:

`height` = **400**

`id_field` = **'id'**
data_index колонки, идентифицирующей объект. Этот параметр будет браться из модели и передаваться как ID в ExtDataStore, т.е в post запросе редактирования будет лежать `{id_param_name: obj.id_field}`

`id_param_name`

Результат Название поля, идентифицирующего объект и название параметра, который будет передаваться в запросе на модификацию/удаление

Тип результата str

`list_sort_order` = **None**
Порядок сортировки элементов списка. Работает следующим образом:

- Если в list_columns модели списка есть поле code, то устанавливается сортировка по возрастанию этого поля
- Если в list_columns модели списка нет поля code, но есть поле name, то устанавливается сортировка по возрастанию поля name

```
list_sort_order = ['code', '-name']
```

`list_window`
Класс отвечающий за отображение окна со списком объектов

псевдоним класса `BaseListWindow`

`list_window_action` = **None**
Экшен показа окна со списком объектов

`model` = **None**
Класс django-модели, для которой будет формироваться справочник

`multi_select_window`
Класс отвечающий за отображение окна множественного выбора из списка объектов

псевдоним класса `BaseMultiSelectWindow`

`multi_select_window_action` = **None**
Экшен с получения данных объектов / редактирование строк

`new_window_action` = **None**
Экшен показа окна добавления объекта

`prepare_row`(*obj*, *request*, *context*)
Установка дополнительных атрибутов объекта перед возвратом json'а строк грида или может вернуть proxy_object

Параметры `obj` (*`django.db.models.Model`*) – Объект из выборки, полученной в get_rows_query

Результат

Тип результата

```
columns = [
    {
        'data_index': 'title',
```

```
        'header': 'Title',
    },
    {
        'data_index': 'date',
        'header': 'Date',
    },
    {
        'data_index': 'done_checkbox',
        'header': 'Done',
    }
]

def prepare_row(self, obj, request, context):
    """
    Добавляет в объект атрибут, для отображения булевого
    поля модели как чек-бокс
    """
    obj = super(RightThingsPack, self).prepare_row(
        obj, request, context)
    obj.done_checkbox = (
        '<div class="x-grid3-check-col-on%s"></div>'
         % '-on' if obj.done else ''
    )
```

`read_only` = **False**
Пак будет настраивать грид на возможность редактирования

`replace_action`(*action_ attr_ name*, *new_ action*)
Заменяет экшен в паке

Параметры

- `action_attr_name` (*str*) – Имя атрибута пака для экшена
- `new_action` (*objectpack.BaseAction*) – Экземпляр экшена

`rows_action` = **None**
Экшен с получения данных объектов / редактирование строк

`save_action` = **None**
Экшен сохранения объекта

`save_row`(*obj*, *create_ new*, *request*, *context*)
Сохраняет объект. При необходимости возбуждается ValidationError, или OverlapError, которые затем отлавливаются в ObjectSaveAction.save_obj

Параметры

- `obj` (*django.db.models.Model*) – Объект модели self.model
- `create_new` (*bool*) – Признак создания нового объекта

`search_fields` = **None**
Список дополнительных полей модели по которым будет идти поиск основной список береться из colums по признаку searchable

`select_window`
Класс отвечающий за отображение окна выбора из списка объектов

псевдоним класса `BaseSelectWindow`

`select_window_action` = **None**
Экшен показа окна со списком для выбора объектов

`title`
Заголовок окна справочника, если не перекрыт в потомках - берется из модели

`width` = **600**

class `objectpack.actions.ObjectRowsAction`

Базовые классы: *objectpack.actions.BaseAction*

Базовый Action получения данных для отображения в окне списка объектов

`apply_filter()`
Метод применяет к выборке self.query фильтр, как правило поступающий от "колоночных фильтров"/фильтров в контекстных меню в окне списка

Примечание: Регистрирует точку расширения *apply_filter* в Observer

---

`apply_limit()`
Метод применяет к выборке self.query операцию оганичения по количеству элементов (для порционной загрузки в окно списка).

`apply_search()`
Метод применяет к выборке self.query фильтр по тексту из поля "Поиск" окна списка

Примечание: Регистрирует точку расширения *apply_search* в Observer

---

`apply_sort_order()`
Метод применяет к выборке self.query сортировку по выбранному в окне списка столбцу

`get_column_data_indexes()`

Результат Список data_index колонок, для формирования json

Тип результата list

`get_rows()`
Метод производит преобразование QuerySet в список. При этом объекты сериализуются в словари

Результат Список сериализованных объектов

Тип результата list

`get_total_count()`
Возвращает общее кол-во объектов

Результат Количество объектов в выборке

Тип результата int

`handle_row_editing`(*request*, *context*, *data*)
Обрабатывает inline-редактирование грида Метод должен вернуть кортеж (удачно/неудачно, "сообщение"/None)

Параметры

- `request` (*django.http.HttpRequest*) – Request
- `context` (*m3.actions.context.DeclarativeActionContext*) – Context

- data (*dict*) – Данные редактирования

Результат (True/False, message/None)

Тип результата tuple

prepare_object(*obj*)

Возвращает словарь, для составления результирующего списка

Параметры obj (*django.db.models.Model*) – Объект, полученный из QuerySet'a

Результат Словарь для сериализации в json

Тип результата dict

Примечание: Регистрирует в Observer точку расширения *prepare_ obj*

---

run(*request*, *context*)

set_query()

Метод получает первоначальную выборку данных в виде QuerySet и помещает в атрибут self.query

---

Примечание: Регистрирует точку расширения *query* в Observer

---

class objectpack.actions.ObjectSaveAction

Базовые классы: *objectpack.actions.BaseAction*

Базовый Action сохранения отредактированного объекта

exception AlreadySaved

Базовые классы: exceptions.Exception

Исключение, с помощью которого расширение, перекрывшее сохранение объекта, может сообщить, что объект сохранен и больше ничего делать не нужно.

ObjectSaveAction.bind_to_obj()

Заполнение объекта данными из полей окна

ObjectSaveAction.bind_win()

Заполнение полей окна по данным из request

ObjectSaveAction.create_obj()

Метод делегирует паку загрузку объекта из БД / создание нового объекта модели

ObjectSaveAction.create_window()

Создаёт окно для дальнейшего биндинга в форму из реквеста

ObjectSaveAction.run(*request*, *context*)

Тело Action, вызывается при обработке запроса к серверу

Параметры

- request (*django.http.HttpRequest*) – Request
- context (*m3.actions.context.DeclarativeActionContext*) – Context

---

Примечание: Обычно не требует перекрытия

ObjectSaveAction.save_obj()
Сохранение объекта в БД

Raise m3.ApplicationLogicException

class objectpack.actions.ObjectSelectWindowAction

Базовые классы: *objectpack.actions.ObjectListWindowAction*

Базовый Action показа окна списка выбора объекта из списка

Совет: Используется с m3_ext.ui.fields.complex.ExtDictSelectField

---

is_select_mode = **True**

set_window_params()

class objectpack.actions.SelectorWindowAction

Базовые классы: *objectpack.actions.BaseAction*

Экшн показа окна выбора с пользовательским экшном обработки выбранных элементов. Например, множественный выбор элементов справочника, для последующего создания связок с ними.

callback_url = **None**
url экшна обработки результата выбора

configure_action(*request*, *context*)
Настройка экшна. Здесь нужно назначать пак и callback

```
def configure_action(self, request, context):
    super(UserPack, self).configure_action(request, context)
    self.data_pack = get_pack_instance('GroupPack')
    self.callback_url = (
        self.parent.selector_save_action.get_absolute_url())
```

configure_context(*request*, *context*)
В данном методе происходит конфигурирование контекста для окна выбора. Возвращаемый результат должен быть экземпляром ActionContext.

Параметры

- request (*django.http.HttpRequest*) – Request
- context (*m3.actions.context.DeclarativeActionContext*) – Context

Тип результата m3.actions.context.ActionContext

configure_window(*win*, *request*, *context*)
В данном методе происходит конфигурирование окна выбора

Параметры

- win (objectpack.ui.BaseSelectWindow) – Окно выбора из справочника
- request (*django.http.HttpRequest*) – Request
- context (*m3.actions.context.DeclarativeActionContext*) – Context

data_pack = **None**
Пак, объекты модели которого выбираются

multi_select = **True**
Признак показа окна множественного выбора

run(*request*, *context*)
Выполнение экшна

Параметры

- request (*django.http.HttpRequest*) – Request
- context (*m3.actions.context.DeclarativeActionContext*) – Context

Результат Результат с окном ExtJS

Тип результата m3_ext.ui.results.ExtUIScriptResult

Raise AssertionError, m3.ApplicationLogicException

Примечание: Без крайней необходимости не перекрывать

---

url = **'/selector_window'**
Жестко определяет url для экшена

TODO: выпылить, использовать проперти из BaseAction

objectpack.actions.multiline_text_window_result(*data*, *success=True*, *title=u''*, *width=600*, *height=500*)
Формирование OpersionResult в виде многострочного окна, с размерами :width x :height и заголовком :title, отображающего текст :data

Параметры

- data (*basestring or Iterable*) – Текст или список со строками
- success (*bool*) – Результат выполнения операции в контексте ExtJS
- title (*basestring*) – Заголовок окна
- width (*int*) – Ширина окна
- height (*int*) – Высота окна

Результат Результат операции в контексте ExtJS

Тип результата m3.actions.results.OperationResult

ui Module

created 23.07.12

author pirogov

class objectpack.ui.BaseEditWindow(**args*, ***kwargs*)

Базовые классы: m3_ext.ui.windows.edit_window.ExtEditWindow, *objectpack.ui.BaseWindow*

Базовое окно редактирования (с формой и кнопкой сабмита)

form
Форма окна

set_params(*params*)

См.также:

objectpack.ui.BaseWindow.set_params()

class objectpack.ui.BaseListWindow(**args*, ***kwargs*)

Базовые классы: *objectpack.ui.BaseWindow*

Базовое окно списка объектов

add_grid_column_filter(*column_name*, *filter_control=None*, *filter_name=None*, *tooltip=None*)

Метод добавляет колоночный фильтр в грид

Параметры

- **column_name** (*str*) – Имя колонки
- **filter_control** – Ext-компонент фильтра
- **filter_name** (*str*) – Имя фильтра
- **tooltip** (*unicode*) – Всплывающая подсказка

del_grid_column_filter(*column_name*, *filter_name=None*)

Метод удаляет колоночный фильтр

Параметры

- **column_name** (*str*) – Имя колонки
- **filter_name** (*str*) – Имя фильтра

render()

Рендеринг окна

set_params(*params*)

Принимает в параметрах пак и делегирует ему конфигурирование грида

См.также:

objectpack.ui.BaseWindow.set_params()

class objectpack.ui.BaseMultiSelectWindow(**args*, ***kwargs*)

Базовые классы: *objectpack.ui.BaseSelectWindow*

Окно множественного выбора в ExtMultiSelectWindow

set_params(*params*)

class objectpack.ui.BaseSelectWindow(**args*, ***kwargs*)

Базовые классы: *objectpack.ui.BaseListWindow*

Окно выбора из списка объектов

set_params(*params*)

См.также:

objectpack.ui.BaseWindow.set_params()

class objectpack.ui.BaseWindow

Базовые классы: m3_ext.ui.windows.window.ExtWindow

Базовое окно

set_params(*params*)

Метод принимает словарь, содержащий параметры окна, передаваемые в окно слоем экшнов

Примечание: Параметры могут содержать общие настройки окна (title, width, height, maximized) и флаг режима для чтения (read_only)

Параметры params (*dict*) – Словарь с параметрами

class objectpack.ui.ColumnsConstructor(*items=None*)

Базовые классы: object

Конструктор колонок для сложных гридов с banded-колонками

Имеет 2 дочерних класса: - Col - простая колонка - BandedCol - группирующая колонка.

Пример использования:

```
# создание колонок inline

cc = ColumnsConstructor()
cc.add(
    cc.Col(header='1'),

    cc.BandedCol(header='2', items=(
        cc.Col(header='3'),
        cc.Col(header='4'),

        cc.BandedCol(header='5', items=(
            cc.Col(header='6'),

            cc.BandedCol(header='7', items=(
                cc.Col(header='8'),
                cc.Col(header='9'),
                cc.BandedCol(),
            )),

            cc.Col(header='10')
        ))
    )),
    cc.Col(header='11')
)

# динамическое создание колонок
for grp_idx in 'ABCD':
    grp = cc.BandedCol(header=grp_idx)

    for col_idx in 'ABCD':
        grp.add(
            cc.Col(header=grp_idx + col_idx)
        )

    cc.add(grp)

cc.configure_grid(grid)
```

class BandedCol(*items=None*, ***kwargs*)

Базовые классы: object

Группирующая колонка

add(**args*)
Добавление колонок
Параметры args (*list*) – Колонки

class ColumnsConstructor.Col(***kwargs*)

Базовые классы: object

Простая колонка

ColumnsConstructor.add(**args*)
Добавление колонок

ColumnsConstructor.configure_grid(*grid*)
Конфигурирование грида

classmethod ColumnsConstructor.from_config(*config*, *ignore_ attrs=None*)
Создание экземпляра на основе конфигурации config

Параметры

- config (*dict*) –
- ignore_attrs –

class objectpack.ui.ComboBoxWithStore(*data=None*, *url=None*, ***kwargs*)

Базовые классы: m3_ext.ui.fields.complex.ExtDictSelectField

Потомок m3-комбобокса со втроенным стором

Примечание: Установка артибутов data или url конфигурирует стор контрола

---

data
Фиксированный стор вида ((id, name),....)

url
URL для динамической загрузки

exception objectpack.ui.GenerationError

Базовые классы: exceptions.Exception

ошибка возникает при проблемы генерации контрола

class objectpack.ui.ModelEditWindow(**args*, ***kwargs*)

Базовые классы: *objectpack.ui.BaseEditWindow*

Простое окно редактирования модели

classmethod fabricate(*model*, ***kwargs*)
Генерирует класс-потомок для конкретной модели

Использование:

```
class Pack(...):
    add_window = ModelEditWindow.fabricate(
        SomeModel,
```

```
        field_list=['code', 'name'],
        model_register=observer,
    )
```

Параметры

- `model` (*django.db.models.Model*) – Модель django
- `kwargs` (*dict*) – Параметры для передачи в field_fabric_params

Результат Субкласс objectpack.ui.ModelEditWindow

`field_fabric_params` = **None**
Словарь kwargs для model_fields_to_controls ("field_list", и т.д.)

`model` = **None**
Модель, для которой будет строится окно

`set_params`(*params*)

См.также:

objectpack.ui.BaseWindow.set_params()

class `objectpack.ui.ObjectGridTab`

Базовые классы: *objectpack.ui.WindowTab*

Вкладка с гридом

`do_layout`(*win*, *tab*)

classmethod `fabricate`(*model*, *model_register*, *tab_class_name=None*)
Возвращает класс вкладки, построенной на основе основного пака для модели `model`. В процессе настройки вкладки экземпляр пака получается посредством вызова `model_register`.get для `model_name`

Параметры

- `model` (*django.db.models.Model*) – Модель django
- `model_register` – Реестр моделей
- `tab_class_name` (*str*) – Имя класса вкладки (если не указано, то генерируется на основе имени класса модели пака)

classmethod `fabricate_from_pack`(*pack_name*, *pack_register*, *tab_class_name=None*)
Возвращает класс вкладки, построенной на основе пака с именем `pack_name`. В процессе настройки вкладки экземпляр пака получается посредством вызова `pack_register`.get_pack_instance для `pack_name`

Параметры

- `pack_name` – Имя пака
- `pack_register` – Реестр паков
- `tab_class_name` – Имя класса вкладки (если не указано, то генерируется на основе имени класса модели пака)

`get_pack`()
Возвращает экземпляр ObjectPack для настройки грида

`init_components`(*win*)
Создание грида

Параметры `win` – Окно

`set_params`(*win*, *params*)

`title`
Заголовок вкладки

class `objectpack.ui.ObjectTab`

Базовые классы: *objectpack.ui.WindowTab*

Вкладка редактирования полей объекта

`do_layout`(*win*, *tab*)

classmethod `fabricate`(*model*, ***kwargs*)
Генерирует класс-потомок для конкретной модели

Использование:

```
class Pack(...):
    add_window = ObjectTab.fabricate(
        SomeModel,
        field_list=['code', 'name'],
        model_register=observer,
    )
```

`field_fabric_params` = **None**
Словарь kwargs для model_fields_to_controls ("field_list", и т.д.)

`init_components`(*win*)

`model` = **None**
Модель, для которой будет строится окно

`set_params`(*win*, *params*)

`title`
Заголовок вкладки

class `objectpack.ui.TabbedEditWindow`(**args*, ***kwargs*)

Базовые классы: *objectpack.ui.TabbedWindow*, *objectpack.ui.BaseEditWindow*

Окно редактирования с вкладками

class `objectpack.ui.TabbedWindow`

Базовые классы: *objectpack.ui.BaseWindow*

Окно со вкладками

`set_params`(*params*)

`tabs` = **None**

class `objectpack.ui.WindowTab`

Базовые классы: `object`

Прототип конструктора таба

do_layout(*win*, *tab*)
Здесь задаётся расположение компонентов. Компоненты должны быть расположены на табе tab окна win

Параметры

- win – Окно
- tab – Вкладка

init_components(*win*)
Здесь создаются компоненты, но не задаётся расположение Компоненты создаются, как атрибуты окна win

Параметры win – Окно

set_params(*win*, *params*)
Установка параметров

Параметры

- win – Окно
- params – Параметры

template = **None**

title = **u'‘**

objectpack.ui.allow_blank(*ctl*)
Устанавливает allow_blank=True у контрола и возвращает его (контрол)

Пример использования:

```
controls = map(allow_blank, controls)
```

objectpack.ui.anchor100(*ctl*)
Устанавливает anchor в 100% у контрола и восвращает его (контрол)

Пример использования:

```
controls = map(anchor100, controls)
```

objectpack.ui.deny_blank(*ctl*)
Устанавливает allow_blank=False у контрола и возвращает его (контрол)

Пример использования:

```
controls = map(allow_blank, controls)
```

objectpack.ui.make_combo_box(***kwargs*)
Создает и возвращает ExtComboBox

Параметры kwargs (*dict*) – Передаются в конструктор комбобокса

objectpack.ui.model_fields_to_controls(*model*, *window*, *field_list=None*, *exclude_list=None*, *model_register=None*, ***kwargs*)
Добавление на окно элементов формы по полям модели

Примечание: exclude_list игнорируется при указанном field_list

---

Примечание: Списки включения/исключения полей могут содержать wildcards вида $x*$ или $*x$, которые трактуются как префиксы и суффиксы

Примечание: При создании полей для связанных моделей ActionPack для модели ищется в реестре моделей `model_register` по имени класса модели (передачей имени в метод "get" реестра)

Параметры

- `model` – Модель django
- `window` (*m3_ext.ui.windows.window.ExtWindow*) – Окно
- `field_list` (*list*) – Список полей
- `exclude_list` (*list*) – Список полей-исключений
- `model_register` – Реестр моделей-паков
- `kwargs` (*dict*) – Дополнительные параметры для передачи в конструктор элементов

Результат Список контролов для полей модели

Тип результата list

`models` Module

Виртуальная модель и proxy-обертка для работы с группой моделей

class `objectpack.models.ModelProxy`(*obj=None*)

Базовые классы: `object`

Proxy-объект инкапсулирующий в себе несколько моделей (для случая, когда одна модель - основная, о другие - её поля)

`model` = **None**

`relations` = **None**

`safe_delete()`

`save()`

class `objectpack.models.ModelProxyMeta`

Базовые классы: `type`

Метакласс для ModelProxy

class `objectpack.models.VirtualModel`

Базовые классы: `object`

Виртуальная модель, реализующая Django-ORM-совместимый API, для работы с произвольными данными.

Пример модели: >>> M = VirtualModel.from_data(... lambda: (... {'x': x, 'y': y * 10} ... for x in xrange(5) ... for y in xrange(5) ...), ... auto_ids=True ...)

Теперь с моделью можно работать так: >>> M.objects.count() 25 >>> M.objects.filter(x__gte=2).exclude(y__in=[10, 20, 30]).count() 6 >>> list(M.objects.filter(x=0).order_by("-y").values_list("y", flat=True)) [40, 30, 20, 10, 0]

exception `DoesNotExist`

Базовые классы: `exceptions.Exception`

exception `VirtualModel.MultipleObjectsReturned`

Базовые классы: `exceptions.Exception`

classmethod `VirtualModel.from_data`(*data*, *auto_ids=False*, *class_name='NewVirtualModel'*)

Возвращает субкласс, основанный на переданных данных @data - iterable из словарей @auto_ids - если True, поле id объектов модели

будет генерироваться автоматически

@class_name - имя класса-потомка

`VirtualModel.objects`

Имитация QueryManager'а Django для VirtualModel

class `objectpack.models.VirtualModelManager`(*model_clz=None*, *procs=None*, ***kwargs*)

Базовые классы: `object`

Имитация QueryManager'а Django для VirtualModel

`all()`

`configure`(***kwargs*)

`count()`

`exclude`(**args*, ***kwargs*)

`filter`(**args*, ***kwargs*)

`get`(**args*, ***kwargs*)

`order_by`(**args*)

`select_related()`

`values`(**args*)

`values_list`(**args*, ***kwargs*)

`objectpack.models.kwargs_only`(**keys*)

`objectpack.models.model_proxy_metaclass`

псевдоним класса *ModelProxyMeta*

0

filters.module

Меахнизмы фильтрации справочников/реестров на базе ObjectPack

class `objectpack.filters.AbstractFilter`

Базовые классы: `object`

Прототип класса, описывающего фильтр для потомков AbstractFilterEngine

get_q(*params*)
Метод возвращает Q-объект, построенный на основе данных словаря params

Параметры params (*dict*) – Словарь с лукапами

Результат Ку-объект

Тип результата django.db.models.Q

get_script()
Метод возвращает список строк-js-скриптов, для дополнения колонки грида

class objectpack.filters.AbstractFilterEngine(*columns*)

Базовые классы: object

Прототип механизма фильтрации

apply_filter(*query*, *request*, *context*)

Параметры

- query (*django.db.models.query.QuerySet*) – Кварисет
- request (*django.http.HttpRequest*) – Реквест
- context (*m3.actions.context.DeclarativeActionContext*) – Контекст

Результат Кварисет отфильтрованный на основе параметров запроса

Тип результата django.db.models.query.QuerySet

configure_grid(*grid*)
Метод настраивает переданный grid на использование фильтров

Параметры grid (*m3_ext.ui.panels.grids.ExtObjectGrid*) – Грид

class objectpack.filters.ColumnFilterEngine(*columns*)

Базовые классы: *objectpack.filters.AbstractFilterEngine*

Механизм фильтрации, реализующий UI в виде полей ввода, встроенных в шапку таблицы

apply_filter(*query*, *request*, *context*)

См.также:

objectpack.filters.AbstractFilterEngine.apply_filter

configure_grid(*grid*)

См.также:

objectpack.filters.AbstractFilterEngine.configure_grid

class objectpack.filters.CustomFilter(*xtype*, *parser*, *lookup*, *tooltip=u''*)

Базовые классы: *objectpack.filters.AbstractFilter*

Фильтр, строящийся на основе xtype

get_script()

См.также:

objectpack.filters.AbstractFilter.get_script

class objectpack.filters.FilterByField(*model*, *field_name*, *lookup=None*, *tooltip=None*, ***field_fabric_params*)

Базовые классы: *objectpack.filters.AbstractFilter*

Фильтр на основе поля модели

field

get_script()

См.также:

objectpack.filters.AbstractFilter.get_script

parsers_map = **[(<class 'django.db.models.fields.DateField'>, 'date', None), (<class 'django.db.models.**
Отображение стандартных полей модели в парсеры и лукапы

class objectpack.filters.FilterGroup(*items*, *op=1*)

Базовые классы: *objectpack.filters.AbstractFilter*

Группа фильтров, являющихся частью булева выражения

AND = **1**
И

OR = **2**
Или

get_q(*params*)

См.также:

objectpack.filters.AbstractFilter.get_q

get_script()

См.также:

objectpack.filters.AbstractFilter.get_script

class objectpack.filters.MenuFilterEngine(*columns*)

Базовые классы: *objectpack.filters.AbstractFilterEngine*

Механизм фильтрации, реализующий UI в виде выпадающих меню колонок

apply_filter(*query*, *request*, *context*)

configure_grid(*grid*)

column_filters Module

Фабрики фильтров для колонок гридов

objectpack.column_filters.choices(*field*, *data*)
Возвращает списковый фильтр для поля @field с указанными вариантами @data (Django model choices)

objectpack.column_filters.within(*field_from*, *field_to*)
Возвращает фильтр, проверяющий попадание указанного значения в диапазон, ограниченный значениями полей @field_from, @field_to

objectpack.column_filters.yes_no(*field*)
Возвращает списковый фильтр с вариантами "Да"/"Нет" для boolean-поля @field

desktop Module

Created 23.07.2012

Author pirogov

class objectpack.desktop.Desktop

Базовые классы: objectpack.desktop._UIFabric

Класс для работы с Рабочим Столом

pack_flag = **'add_to_desktop'**

pack_method = **'extend_desktop'**

static ui_extend_method(*metarole*, **items*)
Добавление элементов на Рабочий Стол

class objectpack.desktop.MainMenu(**args*, ***kwargs*)

Базовые классы: objectpack.desktop._BaseMenu

Класс для работы с главным меню

TO_ADMINISTRY = **3**

TO_DICTS = **1**

TO_REGISTRIES = **2**

TO_ROOT = **None**

administry(**items*)
Элементы для меню "администрирование"

dicts(**items*)
Добавление элементов в меню "Справочники"

pack_flag = **'add_to_menu'**

pack_method = **'extend_menu'**

registries(**items*)
Добавление элементов в меню "Реестры"

static ui_extend_method(*metarole*, **items*)
Добавление элементов в главное меню

class objectpack.desktop.TopMenu(**args*, ***kwargs*)

Базовые классы: *objectpack.desktop.MainMenu*

Класс для работы с верхним меню

pack_flag = **'add_to_top_menu'**

pack_method = **'extend_top_menu'**

static ui_extend_method(*metarole*, **items*)

Добавление элементов в верхнее меню

objectpack.desktop.uificate_the_controller(*controller*, *metarole='none'*, *icon_collection=None*, *menu_root=None*, *top_menu_root=None*)

Интеграция в интерфейс рабочего стола паков контроллера

Параметры

- controller (*m3.actions.ActionController*) – Контроллер
- metarole (*str*) – Метароль
- icon_collection –
- menu_root –
- top_menu_root –

tools Module

Created on 23.07.2012 @author: pirogov

class objectpack.tools.ModelCache(*model*, *object_fabric=None*)

Базовые классы: object

Кэш get-ов объектов одной модели. В качестве ключа кэша - набор параметров для get-а Если в конструкторе указана фабрика объектов, то отсутствующие объекты создаются передачей аргументов фабрике.

forget_last()

get(***kwargs*)

class objectpack.tools.QuerySplitter(*query*, *start*, *limit=0*)

Базовые классы: object

Порционный загрузчик выборки в итеративном контексте

```
>>> from django.test.client import RequestFactory
>>> rf = RequestFactory()
>>> request = rf.post('', {'start': 5, 'limit': 10})
>>> QuerySplitter.make_rows(
...     query=range(50),
...     validator=lambda x: x % 2,
...     request=request)
[5, 7, 9, 11, 13, 15, 17, 19, 21, 23]
```

classmethod make_rows(*query*, *row_fabric=<function <lambda>>*, *validator=<function <lambda>>*, *request=None*, *start=0*, *limit=25*)

Формирует список элементов для грида из выборки. Параметры листания берутся из request, или из параметров start/limit. Элементы перед попаданием прогоняются через row_fabric. В результирующий список попадают только те элементы, вызов validator для которых возвращает *True*

Параметры

- query (*django.db.models.query.QuerySet*) – Кварисет

- row_fabric (*types.FunctionType*) –
- validator (*types.FunctionType*) – Функция валидатор
- request (*django.http.HttpRequest*) – Реквест
- start (*int*) – С какой записи начинать
- limit (*int*) – Сколько записей взять

next()

skip_last()
Команда “не учитывать прошлое значение”

class objectpack.tools.TransactionCM(*using=None*, *catcher=None*)

Базовые классы: object

Транизакция в виде ContextManager

objectpack.tools.cached_to(*attr_name*)
Оборачивает простые методы (без аргументов) и property getters, с целью закэшировать первый полученный результат

Параметры attr_name (*str*) – Куда кэшировать

objectpack.tools.collect_overlaps(*obj*, *queryset*, *attr_begin='begin'*, *attr_end='end'*)
Возвращает список объектов из указанной выборки, которые пересекаются с указанным объектом по указанным полям начала и конца интервала

Параметры

- obj – Объект
- queryset (*django.db.models.query.QuerySet*) – Выборка
- attr_begin (*str*) – Атрибут модели с датой начала
- attr_end (*str*) – Атрибут модели с датой конца

objectpack.tools.extract_date(*request*, *key*, *as_date=False*)
Извлечение даты из request‘а в формате DD.MM.YYYY (в таком виде приходит от ExtDateField) и приведение к Django-формату (YYYY-MM-DD)

objectpack.tools.extract_int(*request*, *key*)
Нормальный извлекатель числа

```
>>> from django.test.client import RequestFactory
>>> rf = RequestFactory()
>>> request = rf.post('', {})
>>> extract_int(request, 'NaN')
```

```
>>> request = rf.post('', {'int':1})
>>> extract_int(request, 'int')
1
```

objectpack.tools.extract_int_list(*request*, *key*)
Нормальный извлекатель списка чисел

```
>>> from django.test.client import RequestFactory
>>> rf = RequestFactory()
>>> request = rf.post('', {})
```

```
>>> extract_int_list(request, 'list')
[]
```

```
>>> request = rf.post('', {'list':'1,2,3,4'})
>>> extract_int_list(request, 'list')
[1, 2, 3, 4]
```

objectpack.tools.find_element_by_type(*container*, *cls*)

Поиск экземпляров элементов во всех вложенных контейнерах

Параметры

- container (*m3_ext.ui.containers.containers.ExtContainer*) – Контейнер
- cls (*types.ClassType*) – Класс

objectpack.tools.int_list(*s*)

```
>>> int_list('10,20, 30')
[10, 20, 30]
```

objectpack.tools.int_or_none(*s*)

```
>>> int_or_none('')
None
>>> int_or_none('10')
10
```

objectpack.tools.int_or_zero(*s*)

```
>>> int_or_zero('')
0
>>> int_or_zero('10')
10
```

objectpack.tools.istraversable(*x*)

возвращает True, если объект x позволяет обход себя в цикле *for*

objectpack.tools.modifier(***kwargs*)

Принимает атрибуты со значениями (в виде kwargs) Возвращает модификатор - функцию, модифицирующую передаваемый ей объект указанными атрибутами

```
>>> w10 = modifier(width=10)
>>> controls = map(w10, controls)
>>> class Object(object): pass
>>> w10 = modifier(width=10)
>>> cls = w10(Object())
>>> cls.width
10
```

objectpack.tools.modify(*obj*, ***kwargs*)

Массовое дополнение атрибутов для объекта с его (объекта) возвратом

```
>>> class Object(object): pass
>>> cls = Object()
```

```
>>> cls.param1 = 0
>>> cls = modify(cls, **{'param1':1, })
>>> cls.param1
1
```

objectpack.tools.str_to_date(*s*)

Извлечение даты из строки

```
>>> str_to_date('31.12.2012') == str_to_date('2012-12-31, Happy New Year')
True
```

exceptions Module

Классы исключений, обрабатываемых ObjectPack

exception objectpack.exceptions.OverlapError(*objects*, *header=u'u0418u043cu0435u044eu0442u0441u044f u043fu0435u0440u0435u0441u0435u0447u0435u043du0438u044f u0441u043e u0441u043bu0435u0434u0443u044eu0449u0438u043cu043 u0437u0430u043fu0438u0441u044fu043cu0438:'*)

Базовые классы: exceptions.Exception

Исключние пересечения интервальных моделей

exception objectpack.exceptions.ValidationError(*text*)

Базовые классы: exceptions.Exception

Исключение валидации

objectpack.observer

observer Package

Механизм подписки на события, возникающие при выполнении actions

base Module

Created on 03.08.2012 @author: pirogov

class objectpack.observer.base.ObservableController(*observer*, **args*, ***kwargs*)

Базовые классы: *objectpack.observer.base.ObservableMixin*, m3.actions.ActionController

Контроллер, поддерживающий механизм подписки через Observer

class VerboseDeclarativeContext(*debug*, ***kwargs*)

Базовые классы: m3.actions.context.DeclarativeActionContext

build(*request*, *rules*)

ObservableController.build_context(*request*, *rules*)

Выполняет построение контекста вызова операции ActionContext на основе переданного request

class objectpack.observer.base.ObservableMixin(*observer*, **args*, ***kwargs*)

Базовые классы: object

Наблюдатель за вызовом actions и кода в точках их (actions) расшрения

append_pack(*pack*)
Добавление ActionPack'а с регистрацией его action'ов в ObserVer'е

class objectpack.observer.base.Observer(*logger=<function <lambda>>*, *verbose_level=1*)

Базовые классы: object

Реестр слушателей, реализующий подписку последних на действия в actions

LOG_CALLS = **2**

LOG_MORE = **3**

LOG_NONE = **0**

LOG_WARNINGS = **1**

configure(*force=False*)
Построение дерева сопоставления экшнов со слушателями Если observer был сконфигурирован ранее и в него ничего не добавили, то построение выполнится, только если передан аргумент *force=True*

Параметры force (*bool*) – Форсировать конфигурирование

get(*model_name*)
Поиск экземпляра ActionPack для модели по имени её класса. Поиск производится среди зарегистрированных Pack'ов, которые являются основными для своих моделей (и привязаны к модели)

get_pack_instance(*pack*)
Возвращает экземпляр зарегистрированного ActionPack. @pack может быть: - классом - строкой с именем класса в формате "package/ClassName"

subscribe(*listener*)
Декоратор, регистрирующий слушателя @listener в реестре слушателей

tools Module

objectpack.observer.tools.name_action(*action*, *pack_name=None*)

Параметры

- action (*objectpack.BaseAction*) – Экшен
- pack_name (*str*) – Имя пака (если не указано - генерится)

Результат Генерация полного имени для action

Тип результата str

Дополнительные паки

slave_object_pack Package

slave_object_pack Package

@author: shibkov

actions Module

Инструменарий для упрощённого создания ActionPack'ов для зависимых моделей

class objectpack.slave_object_pack.actions.SlavePack

Базовые классы: *objectpack.actions.ObjectPack*

"Ведомый" набор действий. Используется чаще всего для грида внутри окна редактирования объекта, отображающего объеты, связанные с редактируемым

declare_context(*action*)
Возвращает декларацию контекста для экшна

get_rows_query(*request*, *context*)

parents = []

save_row(*obj*, *create_new*, *request*, *context*)

tree_object_pack Package

tree_object_pack Package

File: __init__.py Author: Rinat F Sabitov Description:

actions Module

Действия для работы с древовидными справочниками Author: Rinat F Sabitov

class objectpack.tree_object_pack.actions.TreeObjectPack(**args*, ***kwargs*)

Базовые классы: *objectpack.actions.ObjectPack*

Набор действий для работы с объектами, находящимися в древовидной иерархии.

configure_grid(*grid*)

create_edit_window(*create_new*, *request*, *context*)

declare_context(*action*)

get_rows_query(*request*, *context*)

list_window
псевдоним класса BaseTreeListWindow

parent_field = **'parent'**

save_row(*obj*, *create_new*, *request*, *context*)

select_window
псевдоним класса BaseTreeSelectWindow

class objectpack.tree_object_pack.actions.TreeObjectRowsAction

Базовые классы: *objectpack.actions.ObjectRowsAction*

Получение данных для древовидного списка объектов

`prepare_object(obj)`
Сериализация объекта

`run(*args, **kwargs)`

`set_query()`
выборка данных

ui Module

UI для работы с древовидными списками Author: Rinat F Sabitov

class `objectpack.tree_object_pack.ui.BaseObjectTree(*args, **kwargs)`

Базовые классы: `m3_ext.ui.panels.trees.ExtObjectTree`

Визуальный элемент "Дерево"

class `objectpack.tree_object_pack.ui.BaseTreeListWindow(*args, **kwargs)`

Базовые классы: *objectpack.ui.BaseListWindow*

Окно отбражения объектов в виде деревовидного списка

class `objectpack.tree_object_pack.ui.BaseTreeSelectWindow(*args, **kwargs)`

Базовые классы: *objectpack.ui.BaseSelectWindow*

Окно выбора объекта из древовидного списка

`set_params(params)`
установка параметров окна

dictionary_object_pack Package

dictionary_object_package Package

File: __init__.py Author: Rinat F Sabitov Description:

actions Module

File: actions.py Author: Rinat F Sabitov Description:

class `objectpack.dictionary_object_pack.actions.DictionaryObjectPack(*args, **kwargs)`

Базовые классы: *objectpack.actions.ObjectPack*

Набор действий для простых справочников

`add_to_menu` = **True**

`columns` = **[{'header': u'\u043a\u043e\u0434', 'data_index': 'code', 'searchable': True}, {'header': u'\u**

`extend_menu(menu)`
Интеграция в Главное Меню

Глава 5

Быстрый старт

objectpack расширяет возможности m3-core и m3-ext и позволяет экстремально быстро разрабатывать справочники для различных учётных систем.

Например, простой справочник физических лиц:

```
# models.py

class Person(models.Model):

    GENDERS = (
        (0, u''),
        (1, u'Мужской'),
        (2, u'Женский'),
    )

    name = models.CharField(max_length=150, verbose_name=u'Имя')
    surname = models.CharField(max_length=150, verbose_name=u'Фамилия')
    gender = models.PositiveSmallIntegerField(
        choices=GENDERS,
        default=GENDERS[0][0],
        verbose_name=u'Пол')
    birthday = models.DateField(
        null=True, blank=True,
        verbose_name=u'Дата рождения')

    def __unicode__(self):
        return u"%s %s" % (self.surname, self.name)

    class Meta:
        verbose_name = u'Физическое лицо'
        verbose_name_plural = u'Физические лица'
```

```
# actions.py

class PersonPack(objectpack.ObjectPack):
```

```
model = models.Person

add_window = edit_window = objectpack.ModelEditWindow.fabricate(model)

add_to_menu = True

columns = [
    {
        'data_index': 'name',
        'header': u'Имя',
        'width': 2,
    },
    {
        'data_index': 'surname',
        'header': u'Фамилия',
        'width': 2,
    },
    {
        'data_index': 'gender',
        'header': u'Пол',
        'width': 1,
    },
    {
        'data_index': 'birthday',
        'header': u'Дата рождения',
        'width': 1,
    }
]
```

Что мы получим в результате:

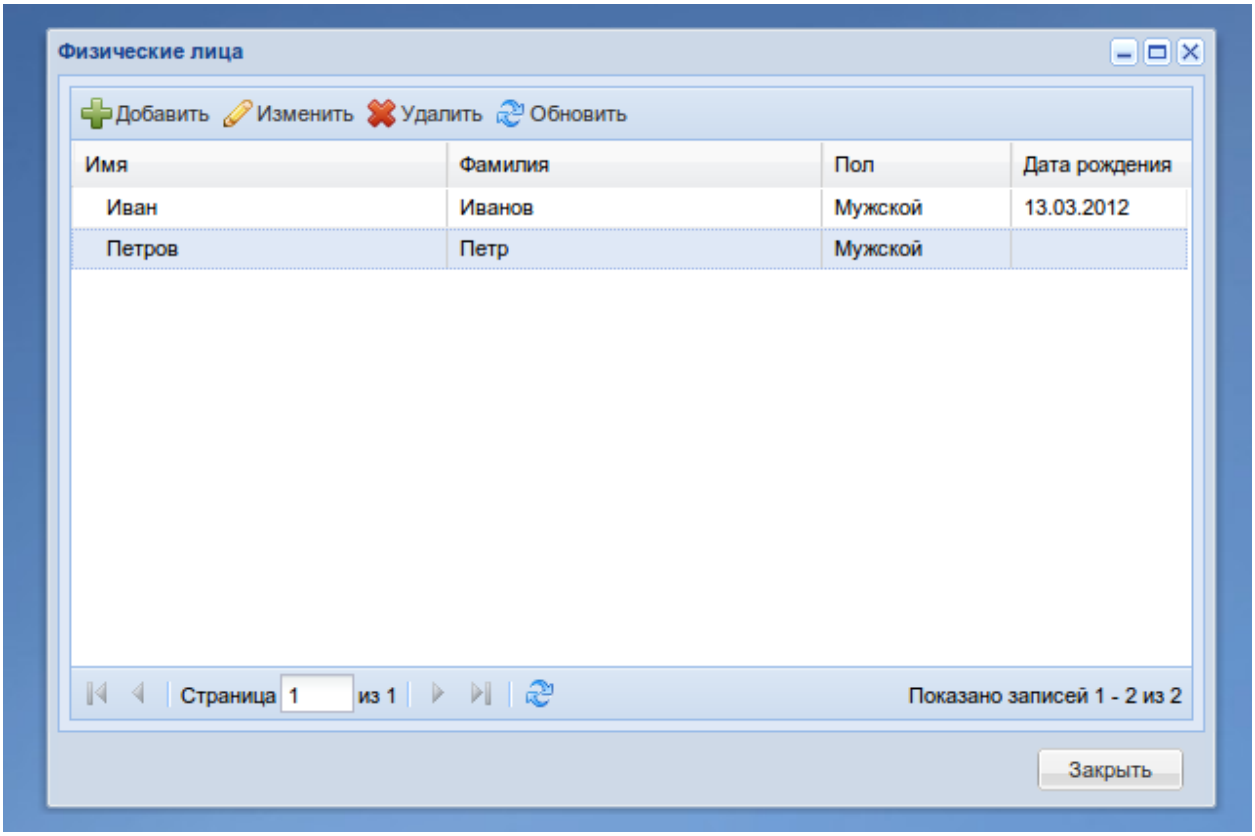

Рис. 5.1: Окно со списком объектов

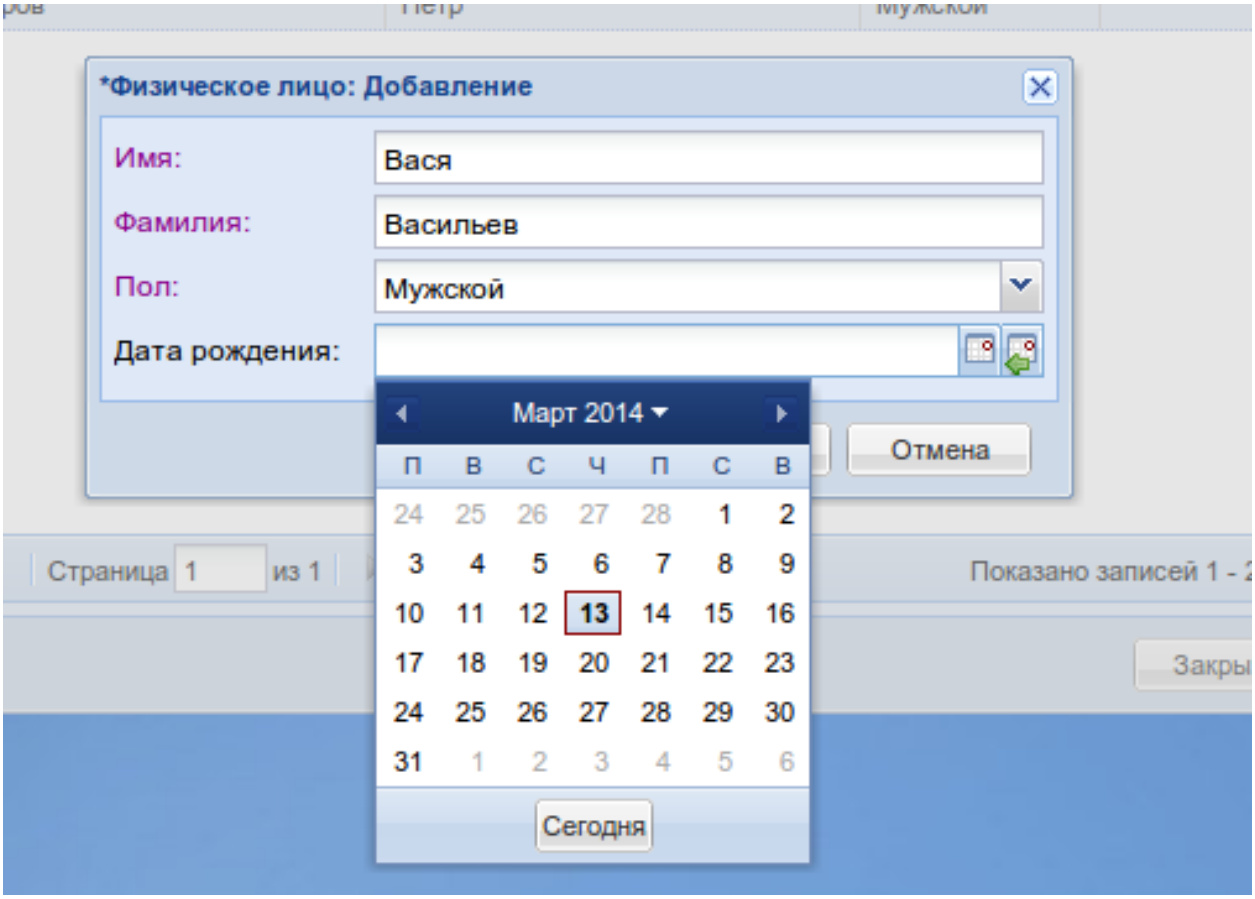

Рис. 5.2: Окно добавления/редактирования объекта

Содержание модулей Python

o

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

T

U

V

W

Y