Odoo9.0模块开发全流程

2016年04月20日

# **构建Odoo模块**

## **模块组成**

* 业务对象

业务对象声明为Python类, 由Odoo自动载入.

* 数据文件

XML或CSV文件格式, 在其中声明了元数据(视图或工作流)、配置数据(模块参数)、演示数据等.

* Web控制器

处理Web浏览器发来的requests.

* 静态web数据

Web用到的图像, CSS或JavaScript文件.

## **模块结构**

一个Odoo模块也是一个Python模块, 存放在一个目录中, 包含一个\_\_init\_\_.py文件, 用于导入其他Python模块.

from . import mymodule

odoo.py提供了一个子命令scaffold可以方便地创建一个空的模块.

$ odoo.py scaffold <module name> <where to put it>

命令执行后, 将会创建一个子目录并且其中包括了Odoo模块所需的一些基本文件.

### **练习 #1**

执行 ./odoo.py scaffold openacademy addons, 在addons目录下创建一个名为openacademy的模块, 生成的目录文件结构如下.

openacademy

├── \_\_init\_\_.py

├── \_\_openerp\_\_.py

├── controllers.py

├── demo.xml

├── models.py

├── security

│ └── ir.model.access.csv

└── templates.xml

各文件内容请查看文件或查看[原文](https://www.odoo.com/documentation/8.0/howtos/backend.html), 然后对\_\_openerp\_\_.py中的几种标识文本进行修改，

至少需要添加'installable':True, 'application':True。

## **对象关系映射**

ORM层是Odoo的一个关键组件, 它可以避免大部分的SQL语句编写从而提高扩展性和安全性.

业务对象用派生自Model的Python类(模型)来编写, 该类的\_name属性定义了模型在Odoo系统中的名称.

from openerp import models

class MinimalModel(models.Model):

\_name = 'test.model'

## **字段**

字段定义模型能够存储什么以及在哪里存储, 字段在模型类中用属性来定义.

from openerp import models, fields

class LessMinimalModel(models.Model):

\_name = 'test.model2'

name = fields.Char()

### **通用属性**

与模型类似, 字段也可以通过参数传递对其进行设定:

name = field.Char(required=True)

字段的常用属性有:

* string (unicode, default: field’s name)

字段标签名称，会显示在界面上（对用户可见）。

* required (bool, default: False)

如果值为True，此字段值不能为空，设置默认值或者在创建记录时提供。

* help (unicode, default: ‘’)

界面上显示提示语。

* index (bool, default: False)

如果值为True，创建表时将为此列添加索引。

### **简单字段**

字段可以分为两类: 简单字段和关系字段. 前者为原子值, 直接保存在模型对应的数据库表中; 后者连接到其他的记录上(可以是相同的模型也可以是不同的模型).

Boolean, Date, Char这些都是简单字段.

### **保留字段**

Odoo在模型中自动创建并维护一些字段, 这些字段就是保留字段, 这些字段数据不需要也不应该手动去修改.

* id (Id)

一条记录的唯一id。

* create\_date (Datetime)

记录创建时间。

* create\_uid (Many2one)

谁创建的记录。

* write\_date (Datetime)

最后修改时间。

* write\_uid (Many2one)

谁最后修改的记录。

### **特殊字段**

默认情况下, Odoo要求模型中有一个*name*字段, 用于显示和搜索, 通过设置\_rec\_name也可以达到这样的目的.

### **练习 #2**

在openacademy模块中定义一个新的模型*Course*, openacademy/models.py内容如下:

*# -\*- coding: utf-8 -\*-*

from openerp import models, fields, api

class Course(models.Model):

\_name = 'openacademy.course'

name = fields.Char(string="Title", required=True)

description = fields.Text()

## **数据文件**

Odoo是一个高度数据驱动的系统, 虽然使用Python代码来定制模块行为, 但很多模块数据是在其载入时setup的, 并且有些模块仅仅为Odoo添加数据.

通过数据文件来定义模块数据, 例如可以使用XML文件中的<record>元素定义数据, 每一个<record>元素创建或者更新数据库中的一条记录, 形式如下:

<openerp>

<data>

<record model="{model name}" id="{record identifier}">

<field name="{a field name}">{a value}</field>

</record>

</data>

<openerp>

* model

Odoo模型名.

* id

外部ID(External Identifier), 通过它可以引用到记录(并且不需要知道记录所在的数据库ID).

* 元素

name属性用于确定字段名称(例如description), 该元素的body给出字段的值.

数据文件必须在模块载入清单文件列表中, 也就是\_\_openerp\_\_.py的’data’列表(全部载入)或’demo’列表(只有设定为载入演示数据才会载入)中.

### **练习 #3**

创建一个数据文件来向Course中添加数据, 编辑openacademy/demo.xml, 并确认\_\_openerp\_\_.py的’demo’列表中有该文件.

<openerp>

<data>

<record model="openacademy.course" id="course0">

<field name="name">Course 0</field>

<field name="description">Course 0's description

Can have multiple lines

</field>

</record>

<record model="openacademy.course" id="course1">

<field name="name">Course 1</field>

*<!-- no description for this one -->*

</record>

<record model="openacademy.course" id="course2">

<field name="name">Course 2</field>

<field name="description">Course 2's description</field>

</record>

</data>

</openerp>

## **动作和菜单**

在Odoo中, 动作和菜单都是定义在数据库中的数据记录, 一般通过数据文件来定义.

动作可以由三种方式触发:

* 点击菜单项(菜单项链接到特定动作)
* 点击视图上的按钮(如果按钮连接到动作)
* 作为对象的上下文动作

使用<menuitem>声明一个ir.ui.menu并将其连接到一个action, 可以用下面的形式的代码.

<record model="ir.actions.act\_window" id="action\_list\_ideas">

<field name="name">Ideas</field>

<field name="res\_model">idea.idea</field>

<field name="view\_mode">tree,form</field>

</record>

<menuitem id="menu\_ideas" parent="menu\_root" name="Ideas" sequence="10"

action="action\_list\_ideas"/>

注意: action必须先于menu的连接使用定义, 数据文件在载入时顺序地执行, 所以动作的ID必须首先存在于数据库中才能使用.

### **练习 #4**

定义一个新的菜单项访问OpenAcademy课程.

创建openacademy/views/openacademy.xml文件, 并在其中添加动作和菜单.

<?xml version="1.0" encoding="UTF-8"?>

<openerp>

<data>

*<!-- window action -->*

*<!--*

*The following tag is an action definition for a "window action",*

*that is an action opening a view or a set of views*

*-->*

<record model="ir.actions.act\_window" id="course\_list\_action">

<field name="name">Courses</field>

<field name="res\_model">openacademy.course</field>

<field name="view\_type">form</field>

<field name="view\_mode">tree,form</field>

<field name="help" type="html">

<p class="oe\_view\_nocontent\_create">Create the first course

</p>

</field>

</record>

*<!-- top level menu: no parent -->*

<menuitem id="main\_openacademy\_menu" name="Open Academy"/>

*<!-- A first level in the left side menu is needed*

*before using action= attribute -->*

<menuitem id="openacademy\_menu" name="Open Academy"

parent="main\_openacademy\_menu"/>

*<!-- the following menuitem should appear \*after\**

*its parent openacademy\_menu and \*after\* its*

*action course\_list\_action -->*

<menuitem id="courses\_menu" name="Courses" parent="openacademy\_menu"

action="course\_list\_action"/>

*<!-- Full id location:*

*action="openacademy.course\_list\_action"*

*It is not required when it is the same module -->*

</data>

</openerp>

在\_\_openerp\_\_.py中添加这个数据文件名到’data’.

'data': [

*# 'security/ir.model.access.csv',*

'templates.xml',

'views/openacademy.xml',

],

更新模块后可以看到菜单, 操作看看效果.

# **基本视图**

视图定义了模型数据如何显示, 每种类型的视图代表一种数据可视化模式.

## **基本的视图定义**

一个视图是以一条ir.ui.view模型数据的形式定义的.

<record model="ir.ui.view" id="view\_id">

<field name="name">view.name</field>

<field name="model">object\_name</field>

<field name="priority" eval="16"/>

<field name="arch" type="xml">

*<!-- view content: <form>, <tree>, <graph>, ... -->*

</field>

</record>

## **Tree views**

Tree view也被称为list views, 在一个表格中显示记录. 根元素是<tree>, 最简形式的tree view只是简单地列出每条记录的多个字段, 每个字段为一列.

<tree string="Idea list">

<field name="name"/>

<field name="inventor\_id"/>

</tree>

## **Form views**

Form用于创建或编辑单条记录, 根元素是<form>, 可以在form中组合各种高层结构元素(如groups, notebooks)以及交互元素(如buttons, fields).

<form string="Idea form">

<group colspan="4">

<group colspan="2" col="2">

<separator string="General stuff" colspan="2"/>

<field name="name"/>

<field name="inventor\_id"/>

</group>

<group colspan="2" col="2">

<separator string="Dates" colspan="2"/>

<field name="active"/>

<field name="invent\_date" readonly="1"/>

</group>

<notebook colspan="4">

<page string="Description">

<field name="description" nolabel="1"/>

</page>

</notebook>

<field name="state"/>

</group>

</form>

### **练习 #5**

为openacademy创建form view, views/openacademy.xml数据文件中增加<record model=”ir.ui.view”…>内容.

<?xml version="1.0" encoding="UTF-8"?>

<openerp>

<data>

<record model="ir.ui.view" id="course\_form\_view">

<field name="name">course.form</field>

<field name="model">openacademy.course</field>

<field name="arch" type="xml">

<form string="Course Form">

<sheet>

<group>

<field name="name"/>

<field name="description"/>

</group>

</sheet>

</form>

</field>

</record>

*<!-- window action -->*

*<!--*

*The following tag is an action definition for a "window action",*

更新模块, 创建一个Course, 可以看到form view变了.

### **练习 #6**

使用*notebook*. 在form view中, 将description字段放在一个tab中, 方便随后添加其他tabs, 对练习#5的form view数据做如下修改.

<sheet>

<group>

<field name="name"/>

</group>

<notebook>

<page string="Description">

<field name="description"/>

</page>

<page string="About">

This is an example of notebooks

</page>

</notebook>

</sheet>

</form>

</field>

更新模块, 看效果.

### **More**

还可以使用HTML为form view提供更加灵活的布局, 例如下面的例子.

<form string="Idea Form">

<header>

<button string="Confirm" type="object" name="action\_confirm"

states="draft" class="oe\_highlight" />

<button string="Mark as done" type="object" name="action\_done"

states="confirmed" class="oe\_highlight"/>

<button string="Reset to draft" type="object" name="action\_draft"

states="confirmed,done" />

<field name="state" widget="statusbar"/>

</header>

<sheet>

<div class="oe\_title">

<label for="name" class="oe\_edit\_only" string="Idea Name" />

<h1><field name="name" /></h1>

</div>

<separator string="General" colspan="2" />

<group colspan="2" col="2">

<field name="description" placeholder="Idea description..." />

</group>

</sheet>

</form>

## **Search views**

Search views用来自定义list views及其它统计/多条记录视图中的搜索字段. 根元素为<search>, 其子元素定义了在哪些字段上进行搜索.

<search>

<field name="name"/>

<field name="inventor\_id"/>

</search>

如果一个模型没有定义对应的Search view, odoo自动创建一个仅搜索name字段的search view.

### **练习 #7**

添加title以及description搜索, 在views/openacademy.xml中定义search view.

</field>

</record>

<record model="ir.ui.view" id="course\_search\_view">

<field name="name">course.search</field>

<field name="model">openacademy.course</field>

<field name="arch" type="xml">

<search>

<field name="name"/>

<field name="description"/>

</search>

</field>

</record>

*<!-- window action -->*

*<!--*

*The following tag is an action definition for a "window action",*

更新模块, 搜索框输入字符后可以看到下方能够选择搜索description字段.

# **模型中的关联**

## **概述**

一个模型中的记录可能关联到其他模型的记录, 例如销售订单记录会关联到一个包含客户信息的客户记录.

### **练习 #8**

为了说明数据关联, 首先增加新的模型.

Open Academy模块中, 一个session是一个在特定时间针对特定听众讲授课程的过程. 需要为session创建相应的模型.

session具有name, 开始日期, 持续时间以及座位数量等. 此外还需要添加相应的action和menuitem显示模型数据.

首先在openacademy/models.py中创建Session类.

class Session(models.Model):

\_name = *'openacademy.session'*

name = fields.Char(required=True)

start\_date = fields.Date()

duration = fields.Float(digits=(6, 2), help="Duration in days")

seats = fields.Integer(string="Number of seats")

然后在openacademy/view/openacademy.xml中添加用于访问session模型的action和menuitem定义.

*<!-- Full id location:*

*action="openacademy.course\_list\_action"*

*It is not required when it is the same module -->*

*<!-- session form view -->*

<record model="ir.ui.view" id="session\_form\_view">

<field name="name">session.form</field>

<field name="model">openacademy.session</field>

<field name="arch" type="xml">

<form string="Session Form">

<sheet>

<group>

<field name="name"/>

<field name="start\_date"/>

<field name="duration"/>

<field name="seats"/>

</group>

</sheet>

</form>

</field>

</record>

<record model="ir.actions.act\_window" id="session\_list\_action">

<field name="name">Sessions</field>

<field name="res\_model">openacademy.session</field>

<field name="view\_type">form</field>

<field name="view\_mode">tree,form</field>

</record>

<menuitem id="session\_menu" name="Sessions"

parent="openacademy\_menu"

action="session\_list\_action"/>

</data>

</openerp>

digits=(6,2)确定浮点数的精度, 6表示总的数字位数(不包括小数点), 2表示小数点后的位数. 所以, digits=(6,2)小数点前最多4位.

## **关联字段**

关联字段指向某些记录，或者是相同的model(模型)，或者是不同的model(模型)。

关联字段类型：

**[Many2one(other\_model, ondelete='set null')](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.fields.Many2one)**

**[One2many(other\_model, related\_field)](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.fields.One2many)**

**[Many2many(other\_model)](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.fields.Many2many)**

### **练习 #9**

## **概述**

* 使用many2one修改Course和Session模型(model)，反映出与其他模型(model)的关联：
* 每个Course有一个负责人，other\_model值为**res.users**
* 每个Session有一个老师，other\_model值为**res.partner**
* 一个Session关联一个Course，other\_model值为**openacademy.course**，必填
* 调整view。

1. 添加相关字段Many2One到model

2. 添加到view

openacademy/models.py

**name = fields.Char(string="Title", required=True)**

**description = fields.Text()**

**responsible\_id = fields.Many2one('res.users',**

**ondelete='set null', string="Responsible", index=True)**

**class Session(models.Model):**

**\_name = 'openacademy.session'**

**start\_date = fields.Date()**

**duration = fields.Float(digits=(6, 2), help="Duration in days")**

**seats = fields.Integer(string="Number of seats")**

**instructor\_id = fields.Many2one('res.partner', string="Instructor")**

**course\_id = fields.Many2one('openacademy.course',**

**ondelete='cascade', string="Course", required=True)**

openacademy/views/openacademy.xml

**<sheet>**

**<group>**

**<field name="name"/>**

**<field name="responsible\_id"/>**

**</group>**

**<notebook>**

**<page string="Description">**

**</field>**

**</record>**

**<!-- override the automatically generated list view for courses -->**

**<record model="ir.ui.view" id="course\_tree\_view">**

**<field name="name">course.tree</field>**

**<field name="model">openacademy.course</field>**

**<field name="arch" type="xml">**

**<tree string="Course Tree">**

**<field name="name"/>**

**<field name="responsible\_id"/>**

**</tree>**

**</field>**

**</record>**

**<!-- window action -->**

**<!--**

**The following tag is an action definition for a "window action",**

**<form string="Session Form">**

**<sheet>**

**<group>**

**<group string="General">**

**<field name="course\_id"/>**

**<field name="name"/>**

**<field name="instructor\_id"/>**

**</group>**

**<group string="Schedule">**

**<field name="start\_date"/>**

**<field name="duration"/>**

**<field name="seats"/>**

**</group>**

**</group>**

**</sheet>**

**</form>**

**</field>**

**</record>**

**<!-- session tree/list view -->**

**<record model="ir.ui.view" id="session\_tree\_view">**

**<field name="name">session.tree</field>**

**<field name="model">openacademy.session</field>**

**<field name="arch" type="xml">**

**<tree string="Session Tree">**

**<field name="name"/>**

**<field name="course\_id"/>**

**</tree>**

**</field>**

**</record>**

**<record model="ir.actions.act\_window" id="session\_list\_action">**

**<field name="name">Sessions</field>**

**<field name="res\_model">openacademy.session</field>**

**Exercise**

Inverse one2many relations

Using the inverse relational field one2many, modify the models to reflect the relation between courses and sessions.

1. Modify the **Course** class, and
2. add the field in the course form view.

openacademy/models.py

**responsible\_id = fields.Many2one('res.users',**

**ondelete='set null', string="Responsible", index=True)**

**session\_ids = fields.One2many(**

**'openacademy.session', 'course\_id', string="Sessions")**

**class Session(models.Model):**

openacademy/views/openacademy.xml

**<page string="Description">**

**<field name="description"/>**

**</page>**

**<page string="Sessions">**

**<field name="session\_ids">**

**<tree string="Registered sessions">**

**<field name="name"/>**

**<field name="instructor\_id"/>**

**</tree>**

**</field>**

**</page>**

**</notebook>**

**</sheet>**

**Exercise**

Multiple many2many relations

Using the relational field many2many, modify the Session model to relate every session to a set of attendees. Attendees will be represented by partner records, so we will relate to the built-in model **res.partner**. Adapt the views accordingly.

1. Modify the **Session** class, and
2. add the field in the form view.

openacademy/models.py

**instructor\_id = fields.Many2one('res.partner', string="Instructor")**

**course\_id = fields.Many2one('openacademy.course',**

**ondelete='cascade', string="Course", required=True)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

openacademy/views/openacademy.xml

**<field name="seats"/>**

**</group>**

**</group>**

**<label for="attendee\_ids"/>**

**<field name="attendee\_ids"/>**

**</sheet>**

**</form>**

**</field>**

## Inheritance

### Model inheritance

Odoo provides two inheritance mechanisms to extend an existing model in a modular way.

The first inheritance mechanism allows a module to modify the behavior of a model defined in another module:

* add fields to a model,
* override the definition of fields on a model,
* add constraints to a model,
* add methods to a model,
* override existing methods on a model.

The second inheritance mechanism (delegation) allows to link every record of a model to a record in a parent model, and provides transparent access to the fields of the parent record.

**See also**

* **[\_inherit](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.models.Model._inherit)**
* **[\_inherits](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.models.Model._inherits)**

### View inheritance

Instead of modifying existing views in place (by overwriting them), Odoo provides view inheritance where children "extension" views are applied on top of root views, and can add or remove content from their parent.

An extension view references its parent using the **inherit\_id** field, and instead of a single view its **arch** field is composed of any number of **xpath** elements selecting and altering the content of their parent view:

**<!-- improved idea categories list -->**

**<record id="idea\_category\_list2" model="ir.ui.view">**

**<field name="name">id.category.list2</field>**

**<field name="model">idea.category</field>**

**<field name="inherit\_id" ref="id\_category\_list"/>**

**<field name="arch" type="xml">**

**<!-- find field description and add the field**

**idea\_ids after it -->**

**<xpath expr="//field[@name='description']" position="after">**

**<field name="idea\_ids" string="Number of ideas"/>**

**</xpath>**

**</field>**

**</record>**

**expr**

An [**XPath**](http://w3.org/TR/xpath) expression selecting a single element in the parent view. Raises an error if it matches no element or more than one

**position**

Operation to apply to the matched element:

**inside**

appends **xpath**'s body at the end of the matched element

**replace**

replaces the matched element by the **xpath**'s body

**before**

inserts the **xpath**'s body as a sibling before the matched element

**after**

inserts the **xpaths**'s body as a sibling after the matched element

**attributes**

alters the attributes of the matched element using special **attribute** elements in the **xpath**'s body

**Tip**

When matching a single element, the **position** attribute can be set directly on the element to be found. Both inheritances below will give the same result.

**<xpath expr="//field[@name='description']" position="after">**

**<field name="idea\_ids" />**

**</xpath>**

**<field name="description" position="after">**

**<field name="idea\_ids" />**

**</field>**

**Exercise**

Alter existing content

* Using model inheritance, modify the existing Partner model to add an **instructor** boolean field, and a many2many field that corresponds to the session-partner relation
* Using view inheritance, display this fields in the partner form view

**Note**

This is the opportunity to introduce the developer mode to inspect the view, find its external ID and the place to put the new field.

1. Create a file **openacademy/partner.py** and import it in **\_\_init\_\_.py**
2. Create a file **openacademy/views/partner.xml** and add it to **\_\_openerp\_\_.py**

openacademy/\_\_init\_\_.py

**# -\*- coding: utf-8 -\*-**

**from . import controllers**

**from . import models**

**from . import partner**

openacademy/\_\_openerp\_\_.py

**# 'security/ir.model.access.csv',**

**'templates.xml',**

**'views/openacademy.xml',**

**'views/partner.xml',**

**],**

**# only loaded in demonstration mode**

**'demo': [**

openacademy/partner.py

**# -\*- coding: utf-8 -\*-**

**from openerp import fields, models**

**class Partner(models.Model):**

**\_inherit = 'res.partner'**

**# Add a new column to the res.partner model, by default partners are not**

**# instructors**

**instructor = fields.Boolean("Instructor", default=False)**

**session\_ids = fields.Many2many('openacademy.session',**

**string="Attended Sessions", readonly=True)**

openacademy/views/partner.xml

**<?xml version="1.0" encoding="UTF-8"?>**

**<openerp>**

**<data>**

**<!-- Add instructor field to existing view -->**

**<record model="ir.ui.view" id="partner\_instructor\_form\_view">**

**<field name="name">partner.instructor</field>**

**<field name="model">res.partner</field>**

**<field name="inherit\_id" ref="base.view\_partner\_form"/>**

**<field name="arch" type="xml">**

**<notebook position="inside">**

**<page string="Sessions">**

**<group>**

**<field name="instructor"/>**

**<field name="session\_ids"/>**

**</group>**

**</page>**

**</notebook>**

**</field>**

**</record>**

**<record model="ir.actions.act\_window" id="contact\_list\_action">**

**<field name="name">Contacts</field>**

**<field name="res\_model">res.partner</field>**

**<field name="view\_mode">tree,form</field>**

**</record>**

**<menuitem id="configuration\_menu" name="Configuration"**

**parent="main\_openacademy\_menu"/>**

**<menuitem id="contact\_menu" name="Contacts"**

**parent="configuration\_menu"**

**action="contact\_list\_action"/>**

**</data>**

**</openerp>**

#### Domains

In Odoo, **[Domains](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "reference-orm-domains)** are values that encode conditions on records. A domain is a list of criteria used to select a subset of a model's records. Each criteria is a triple with a field name, an operator and a value.

For instance, when used on the Product model the following domain selects all services with a unit price over 1000:

**[('product\_type', '=', 'service'), ('unit\_price', '>', 1000)]**

By default criteria are combined with an implicit AND. The logical operators **&** (AND), **|** (OR) and **!** (NOT) can be used to explicitly combine criteria. They are used in prefix position (the operator is inserted before its arguments rather than between). For instance to select products "which are services OR have a unit price which is NOT between 1000 and 2000":

**['|',**

**('product\_type', '=', 'service'),**

**'!', '&',**

**('unit\_price', '>=', 1000),**

**('unit\_price', '<', 2000)]**

A **domain** parameter can be added to relational fields to limit valid records for the relation when trying to select records in the client interface.

**Exercise**

Domains on relational fields

When selecting the instructor for a Session, only instructors (partners with **instructor** set to **True**) should be visible.

openacademy/models.py

**duration = fields.Float(digits=(6, 2), help="Duration in days")**

**seats = fields.Integer(string="Number of seats")**

**instructor\_id = fields.Many2one('res.partner', string="Instructor",**

**domain=[('instructor', '=', True)])**

**course\_id = fields.Many2one('openacademy.course',**

**ondelete='cascade', string="Course", required=True)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

**Note**

A domain declared as a literal list is evaluated server-side and can't refer to dynamic values on the right-hand side, a domain declared as a string is evaluated client-side and allows field names on the right-hand side

**Exercise**

More complex domains

Create new partner categories Teacher / Level 1 and Teacher / Level 2. The instructor for a session can be either an instructor or a teacher (of any level).

1. Modify the Session model's domain
2. Modify **openacademy/view/partner.xml** to get access to Partner categories:

openacademy/models.py

**seats = fields.Integer(string="Number of seats")**

**instructor\_id = fields.Many2one('res.partner', string="Instructor",**

**domain=['|', ('instructor', '=', True),**

**('category\_id.name', 'ilike', "Teacher")])**

**course\_id = fields.Many2one('openacademy.course',**

**ondelete='cascade', string="Course", required=True)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

openacademy/views/partner.xml

**<menuitem id="contact\_menu" name="Contacts"**

**parent="configuration\_menu"**

**action="contact\_list\_action"/>**

**<record model="ir.actions.act\_window" id="contact\_cat\_list\_action">**

**<field name="name">Contact Tags</field>**

**<field name="res\_model">res.partner.category</field>**

**<field name="view\_mode">tree,form</field>**

**</record>**

**<menuitem id="contact\_cat\_menu" name="Contact Tags"**

**parent="configuration\_menu"**

**action="contact\_cat\_list\_action"/>**

**<record model="res.partner.category" id="teacher1">**

**<field name="name">Teacher / Level 1</field>**

**</record>**

**<record model="res.partner.category" id="teacher2">**

**<field name="name">Teacher / Level 2</field>**

**</record>**

**</data>**

**</openerp>**

## Computed fields and default values

So far fields have been stored directly in and retrieved directly from the database. Fields can also be computed. In that case, the field's value is not retrieved from the database but computed on-the-fly by calling a method of the model.

To create a computed field, create a field and set its attribute **compute** to the name of a method. The computation method should simply set the value of the field to compute on every record in **self**.

**Danger**

**self** is a collection

The object **self** is a recordset, i.e., an ordered collection of records. It supports the standard Python operations on collections, like **len(self)** and **iter(self)**, plus extra set operations like **recs1 + recs2**.

Iterating over **self** gives the records one by one, where each record is itself a collection of size 1. You can access/assign fields on single records by using the dot notation, like **record.name**.

**import random**

**from openerp import models, fields, api**

**class ComputedModel(models.Model):**

**\_name = 'test.computed'**

**name = fields.Char(compute='\_compute\_name')**

**@api.multi**

**def \_compute\_name(self):**

**for record in self:**

**record.name = str(random.randint(1, 1e6))**

### Dependencies

The value of a computed field usually depends on the values of other fields on the computed record. The ORM expects the developer to specify those dependencies on the compute method with the decorator **[depends()](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.api.depends)**. The given dependencies are used by the ORM to trigger the recomputation of the field whenever some of its dependencies have been modified:

**from openerp import models, fields, api**

**class ComputedModel(models.Model):**

**\_name = 'test.computed'**

**name = fields.Char(compute='\_compute\_name')**

**value = fields.Integer()**

**@api.depends('value')**

**def \_compute\_name(self):**

**for record in self:**

**record.name = "Record with value %s" % record.value**

**Exercise**

Computed fields

* Add the percentage of taken seats to the Session model
* Display that field in the tree and form views
* Display the field as a progress bar

1. Add a computed field to Session
2. Show the field in the Session view:

openacademy/models.py

**course\_id = fields.Many2one('openacademy.course',**

**ondelete='cascade', string="Course", required=True)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

**taken\_seats = fields.Float(string="Taken seats", compute='\_taken\_seats')**

**@api.depends('seats', 'attendee\_ids')**

**def \_taken\_seats(self):**

**for r in self:**

**if not r.seats:**

**r.taken\_seats = 0.0**

**else:**

**r.taken\_seats = 100.0 \* len(r.attendee\_ids) / r.seats**

openacademy/views/openacademy.xml

**<field name="start\_date"/>**

**<field name="duration"/>**

**<field name="seats"/>**

**<field name="taken\_seats" widget="progressbar"/>**

**</group>**

**</group>**

**<label for="attendee\_ids"/>**

**<tree string="Session Tree">**

**<field name="name"/>**

**<field name="course\_id"/>**

**<field name="taken\_seats" widget="progressbar"/>**

**</tree>**

**</field>**

**</record>**

### Default values

Any field can be given a default value. In the field definition, add the option **default=X** where **X** is either a Python literal value (boolean, integer, float, string), or a function taking a recordset and returning a value:

**name = fields.Char(default="Unknown")**

**user\_id = fields.Many2one('res.users', default=lambda self: self.env.user)**

**Note**

The object **self.env** gives access to request parameters and other useful things:

* **self.env.cr** or **self.\_cr** is the database cursor object; it is used for querying the database
* **self.env.uid** or **self.\_uid** is the current user's database id
* **self.env.user** is the current user's record
* **self.env.context** or **self.\_context** is the context dictionary
* **self.env.ref(xml\_id)** returns the record corresponding to an XML id
* **self.env[model\_name]** returns an instance of the given model

**Exercise**

Active objects – Default values

* Define the start\_date default value as today (see **[Date](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.fields.Date)**).
* Add a field **active** in the class Session, and set sessions as active by default.

openacademy/models.py

**\_name = 'openacademy.session'**

**name = fields.Char(required=True)**

**start\_date = fields.Date(default=fields.Date.today)**

**duration = fields.Float(digits=(6, 2), help="Duration in days")**

**seats = fields.Integer(string="Number of seats")**

**active = fields.Boolean(default=True)**

**instructor\_id = fields.Many2one('res.partner', string="Instructor",**

**domain=['|', ('instructor', '=', True),**

openacademy/views/openacademy.xml

**<field name="course\_id"/>**

**<field name="name"/>**

**<field name="instructor\_id"/>**

**<field name="active"/>**

**</group>**

**<group string="Schedule">**

**<field name="start\_date"/>**

**Note**

Odoo has built-in rules making fields with an **active** field set to **False**invisible.

## Onchange

The "onchange" mechanism provides a way for the client interface to update a form whenever the user has filled in a value in a field, without saving anything to the database.

For instance, suppose a model has three fields **amount**, **unit\_price** and **price**, and you want to update the price on the form when any of the other fields is modified. To achieve this, define a method where **self** represents the record in the form view, and decorate it with **[onchange()](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.api.onchange)** to specify on which field it has to be triggered. Any change you make on **self** will be reflected on the form.

**<!-- content of form view -->**

**<field name="amount"/>**

**<field name="unit\_price"/>**

**<field name="price" readonly="1"/>**

**# onchange handler**

**@api.onchange('amount', 'unit\_price')**

**def \_onchange\_price(self):**

**# set auto-changing field**

**self.price = self.amount \* self.unit\_price**

**# Can optionally return a warning and domains**

**return {**

**'warning': {**

**'title': "Something bad happened",**

**'message': "It was very bad indeed",**

**}**

**}**

For computed fields, valued **onchange** behavior is built-in as can be seen by playing with the Session form: change the number of seats or participants, and the **taken\_seats** progressbar is automatically updated.

**Exercise**

Warning

Add an explicit onchange to warn about invalid values, like a negative number of seats, or more participants than seats.

openacademy/models.py

**r.taken\_seats = 0.0**

**else:**

**r.taken\_seats = 100.0 \* len(r.attendee\_ids) / r.seats**

**@api.onchange('seats', 'attendee\_ids')**

**def \_verify\_valid\_seats(self):**

**if self.seats < 0:**

**return {**

**'warning': {**

**'title': "Incorrect 'seats' value",**

**'message': "The number of available seats may not be negative",**

**},**

**}**

**if self.seats < len(self.attendee\_ids):**

**return {**

**'warning': {**

**'title': "Too many attendees",**

**'message': "Increase seats or remove excess attendees",**

**},**

**}**

## Model constraints

Odoo provides two ways to set up automatically verified invariants: **[Python constraints](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.api.constrains)** and **[SQL constraints](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.models.Model._sql_constraints)**.

A Python constraint is defined as a method decorated with **[constrains()](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.api.constrains)**, and invoked on a recordset. The decorator specifies which fields are involved in the constraint, so that the constraint is automatically evaluated when one of them is modified. The method is expected to raise an exception if its invariant is not satisfied:

**from openerp.exceptions import ValidationError**

**@api.constrains('age')**

**def \_check\_something(self):**

**for record in self:**

**if record.age > 20:**

**raise ValidationError("Your record is too old: %s" % record.age)**

**# all records passed the test, don't return anything**

**Exercise**

Add Python constraints

Add a constraint that checks that the instructor is not present in the attendees of his/her own session.

openacademy/models.py

**# -\*- coding: utf-8 -\*-**

**from openerp import models, fields, api, exceptions**

**class Course(models.Model):**

**\_name = 'openacademy.course'**

**'message': "Increase seats or remove excess attendees",**

**},**

**}**

**@api.constrains('instructor\_id', 'attendee\_ids')**

**def \_check\_instructor\_not\_in\_attendees(self):**

**for r in self:**

**if r.instructor\_id and r.instructor\_id in r.attendee\_ids:**

**raise exceptions.ValidationError("A session's instructor can't be an attendee")**

SQL constraints are defined through the model attribute **[\_sql\_constraints](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.models.Model._sql_constraints)**. The latter is assigned to a list of triples of strings**(name, sql\_definition, message)**, where **name** is a valid SQL constraint name, **sql\_definition** is a [**table\_constraint**](http://www.postgresql.org/docs/9.3/static/ddl-constraints.html)expression, and **message** is the error message.

**Exercise**

Add SQL constraints

With the help of [**PostgreSQL's documentation**](http://www.postgresql.org/docs/9.3/static/ddl-constraints.html) , add the following constraints:

1. CHECK that the course description and the course title are different
2. Make the Course's name UNIQUE

openacademy/models.py

**session\_ids = fields.One2many(**

**'openacademy.session', 'course\_id', string="Sessions")**

**\_sql\_constraints = [**

**('name\_description\_check',**

**'CHECK(name != description)',**

**"The title of the course should not be the description"),**

**('name\_unique',**

**'UNIQUE(name)',**

**"The course title must be unique"),**

**]**

**class Session(models.Model):**

**\_name = 'openacademy.session'**

**Exercise**

Exercise 6 - Add a duplicate option

Since we added a constraint for the Course name uniqueness, it is not possible to use the "duplicate" function anymore (**Form ‣ Duplicate**).

Re-implement your own "copy" method which allows to duplicate the Course object, changing the original name into "Copy of [original name]".

openacademy/models.py

**session\_ids = fields.One2many(**

**'openacademy.session', 'course\_id', string="Sessions")**

**@api.multi**

**def copy(self, default=None):**

**default = dict(default or {})**

**copied\_count = self.search\_count(**

**[('name', '=like', u"Copy of {}%".format(self.name))])**

**if not copied\_count:**

**new\_name = u"Copy of {}".format(self.name)**

**else:**

**new\_name = u"Copy of {} ({})".format(self.name, copied\_count)**

**default['name'] = new\_name**

**return super(Course, self).copy(default)**

**\_sql\_constraints = [**

**('name\_description\_check',**

**'CHECK(name != description)',**

## Advanced Views

### Tree views

Tree views can take supplementary attributes to further customize their behavior:

**decoration-{$name}**

allow changing the style of a row's text based on the corresponding record's attributes.

Values are Python expressions. For each record, the expression is evaluated with the record's attributes as context values and if **true**, the corresponding style is applied to the row. Other context values are **uid** (the id of the current user) and**current\_date** (the current date as a string of the form **yyyy-MM-dd**).

**{$name}** can be **bf** (**font-weight: bold**), **it** (**font-style: italic**), or any **[bootstrap contextual color](http://getbootstrap.com/components/" \l "available-variations)** (**danger**, **info**,**muted**, **primary**, **success** or **warning**).

**<tree string="Idea Categories" decoration-info="state=='draft'"**

**decoration-danger="state=='trashed'">**

**<field name="name"/>**

**<field name="state"/>**

**</tree>**

**editable**

Either **"top"** or **"bottom"**. Makes the tree view editable in-place (rather than having to go through the form view), the value is the position where new rows appear.

**Exercise**

List coloring

Modify the Session tree view in such a way that sessions lasting less than 5 days are colored blue, and the ones lasting more than 15 days are colored red.

Modify the session tree view:

openacademy/views/openacademy.xml

**<field name="name">session.tree</field>**

**<field name="model">openacademy.session</field>**

**<field name="arch" type="xml">**

**<tree string="Session Tree" decoration-info="duration&lt;5" decoration-danger="duration&gt;15">**

**<field name="name"/>**

**<field name="course\_id"/>**

**<field name="duration" invisible="1"/>**

**<field name="taken\_seats" widget="progressbar"/>**

**</tree>**

**</field>**

### Calendars

Displays records as calendar events. Their root element is **<calendar>** and their most common attributes are:

**color**

The name of the field used for color segmentation. Colors are automatically distributed to events, but events in the same color segment (records which have the same value for their **@color** field) will be given the same color.

**date\_start**

record's field holding the start date/time for the event

**date\_stop (optional)**

record's field holding the end date/time for the event

field (to define the label for each calendar event)

**<calendar string="Ideas" date\_start="invent\_date" color="inventor\_id">**

**<field name="name"/>**

**</calendar>**

**Exercise**

Calendar view

Add a Calendar view to the Session model enabling the user to view the events associated to the Open Academy.

1. Add an **end\_date** field computed from **start\_date** and **duration**

**Tip**

the inverse function makes the field writable, and allows moving the sessions (via drag and drop) in the calendar view

1. Add a calendar view to the Session model
2. And add the calendar view to the Session model's actions

openacademy/models.py

**# -\*- coding: utf-8 -\*-**

**from datetime import timedelta**

**from openerp import models, fields, api, exceptions**

**class Course(models.Model):**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

**taken\_seats = fields.Float(string="Taken seats", compute='\_taken\_seats')**

**end\_date = fields.Date(string="End Date", store=True,**

**compute='\_get\_end\_date', inverse='\_set\_end\_date')**

**@api.depends('seats', 'attendee\_ids')**

**def \_taken\_seats(self):**

**},**

**}**

**@api.depends('start\_date', 'duration')**

**def \_get\_end\_date(self):**

**for r in self:**

**if not (r.start\_date and r.duration):**

**r.end\_date = r.start\_date**

**continue**

**# Add duration to start\_date, but: Monday + 5 days = Saturday, so**

**# subtract one second to get on Friday instead**

**start = fields.Datetime.from\_string(r.start\_date)**

**duration = timedelta(days=r.duration, seconds=-1)**

**r.end\_date = start + duration**

**def \_set\_end\_date(self):**

**for r in self:**

**if not (r.start\_date and r.end\_date):**

**continue**

**# Compute the difference between dates, but: Friday - Monday = 4 days,**

**# so add one day to get 5 days instead**

**start\_date = fields.Datetime.from\_string(r.start\_date)**

**end\_date = fields.Datetime.from\_string(r.end\_date)**

**r.duration = (end\_date - start\_date).days + 1**

**@api.constrains('instructor\_id', 'attendee\_ids')**

**def \_check\_instructor\_not\_in\_attendees(self):**

**for r in self:**

openacademy/views/openacademy.xml

**</field>**

**</record>**

**<!-- calendar view -->**

**<record model="ir.ui.view" id="session\_calendar\_view">**

**<field name="name">session.calendar</field>**

**<field name="model">openacademy.session</field>**

**<field name="arch" type="xml">**

**<calendar string="Session Calendar" date\_start="start\_date"**

**date\_stop="end\_date"**

**color="instructor\_id">**

**<field name="name"/>**

**</calendar>**

**</field>**

**</record>**

**<record model="ir.actions.act\_window" id="session\_list\_action">**

**<field name="name">Sessions</field>**

**<field name="res\_model">openacademy.session</field>**

**<field name="view\_type">form</field>**

**<field name="view\_mode">tree,form,calendar</field>**

**</record>**

**<menuitem id="session\_menu" name="Sessions"**

### Search views

Search view **<field>** elements can have a **@filter\_domain** that overrides the domain generated for searching on the given field. In the given domain, **self** represents the value entered by the user. In the example below, it is used to search on both fields **name**and **description**.

Search views can also contain **<filter>** elements, which act as toggles for predefined searches. Filters must have one of the following attributes:

**domain**

add the given domain to the current search

**context**

add some context to the current search; use the key **group\_by** to group results on the given field name

**<search string="Ideas">**

**<field name="name"/>**

**<field name="description" string="Name and description"**

**filter\_domain="['|', ('name', 'ilike', self), ('description', 'ilike', self)]"/>**

**<field name="inventor\_id"/>**

**<field name="country\_id" widget="selection"/>**

**<filter name="my\_ideas" string="My Ideas"**

**domain="[('inventor\_id', '=', uid)]"/>**

**<group string="Group By">**

**<filter name="group\_by\_inventor" string="Inventor"**

**context="{'group\_by': 'inventor\_id'}"/>**

**</group>**

**</search>**

To use a non-default search view in an action, it should be linked using the **search\_view\_id** field of the action record.

The action can also set default values for search fields through its **context** field: context keys of the form**search\_default\_field\_name** will initialize field\_name with the provided value. Search filters must have an optional **@name** to have a default and behave as booleans (they can only be enabled by default).

**Exercise**

Search views

1. Add a button to filter the courses for which the current user is the responsible in the course search view. Make it selected by default.
2. Add a button to group courses by responsible user.

openacademy/views/openacademy.xml

**<search>**

**<field name="name"/>**

**<field name="description"/>**

**<filter name="my\_courses" string="My Courses"**

**domain="[('responsible\_id', '=', uid)]"/>**

**<group string="Group By">**

**<filter name="by\_responsible" string="Responsible"**

**context="{'group\_by': 'responsible\_id'}"/>**

**</group>**

**</search>**

**</field>**

**</record>**

**<field name="res\_model">openacademy.course</field>**

**<field name="view\_type">form</field>**

**<field name="view\_mode">tree,form</field>**

**<field name="context" eval="{'search\_default\_my\_courses': 1}"/>**

**<field name="help" type="html">**

**<p class="oe\_view\_nocontent\_create">Create the first course**

**</p>**

### Gantt

Horizontal bar charts typically used to show project planning and advancement, their root element is **<gantt>**.

**<gantt string="Ideas"**

**date\_start="invent\_date"**

**date\_stop="date\_finished"**

**progress="progress"**

**default\_group\_by="inventor\_id" />**

**Exercise**

Gantt charts

Add a Gantt Chart enabling the user to view the sessions scheduling linked to the Open Academy module. The sessions should be grouped by instructor.

1. Create a computed field expressing the session's duration in hours
2. Add the gantt view's definition, and add the gantt view to the Session model's action

openacademy/models.py

**end\_date = fields.Date(string="End Date", store=True,**

**compute='\_get\_end\_date', inverse='\_set\_end\_date')**

**hours = fields.Float(string="Duration in hours",**

**compute='\_get\_hours', inverse='\_set\_hours')**

**@api.depends('seats', 'attendee\_ids')**

**def \_taken\_seats(self):**

**for r in self:**

**end\_date = fields.Datetime.from\_string(r.end\_date)**

**r.duration = (end\_date - start\_date).days + 1**

**@api.depends('duration')**

**def \_get\_hours(self):**

**for r in self:**

**r.hours = r.duration \* 24**

**def \_set\_hours(self):**

**for r in self:**

**r.duration = r.hours / 24**

**@api.constrains('instructor\_id', 'attendee\_ids')**

**def \_check\_instructor\_not\_in\_attendees(self):**

**for r in self:**

openacademy/views/openacademy.xml

**</field>**

**</record>**

**<record model="ir.ui.view" id="session\_gantt\_view">**

**<field name="name">session.gantt</field>**

**<field name="model">openacademy.session</field>**

**<field name="arch" type="xml">**

**<gantt string="Session Gantt" color="course\_id"**

**date\_start="start\_date" date\_delay="hours"**

**default\_group\_by='instructor\_id'>**

**<field name="name"/>**

**</gantt>**

**</field>**

**</record>**

**<record model="ir.actions.act\_window" id="session\_list\_action">**

**<field name="name">Sessions</field>**

**<field name="res\_model">openacademy.session</field>**

**<field name="view\_type">form</field>**

**<field name="view\_mode">tree,form,calendar,gantt</field>**

**</record>**

**<menuitem id="session\_menu" name="Sessions"**

### Graph views

Graph views allow aggregated overview and analysis of models, their root element is **<graph>**.

**Note**

Pivot views (element **<pivot>**) a multidimensional table, allows the selection of filers and dimensions to get the right aggregated dataset before moving to a more graphical overview. The pivot view shares the same content definition as graph views.

Graph views have 4 display modes, the default mode is selected using the **@type** attribute.

**Bar (default)**

a bar chart, the first dimension is used to define groups on the horizontal axis, other dimensions define aggregated bars within each group.

By default bars are side-by-side, they can be stacked by using **@stacked="True"** on the **<graph>**

**Line**

2-dimensional line chart

**Pie**

2-dimensional pie

Graph views contain **<field>** with a mandatory **@type** attribute taking the values:

**row (default)**

the field should be aggregated by default

**measure**

the field should be aggregated rather than grouped on

**<graph string="Total idea score by Inventor">**

**<field name="inventor\_id"/>**

**<field name="score" type="measure"/>**

**</graph>**

**Warning**

Graph views perform aggregations on database values, they do not work with non-stored computed fields.

**Exercise**

Graph view

Add a Graph view in the Session object that displays, for each course, the number of attendees under the form of a bar chart.

1. Add the number of attendees as a stored computed field
2. Then add the relevant view

openacademy/models.py

**hours = fields.Float(string="Duration in hours",**

**compute='\_get\_hours', inverse='\_set\_hours')**

**attendees\_count = fields.Integer(**

**string="Attendees count", compute='\_get\_attendees\_count', store=True)**

**@api.depends('seats', 'attendee\_ids')**

**def \_taken\_seats(self):**

**for r in self:**

**for r in self:**

**r.duration = r.hours / 24**

**@api.depends('attendee\_ids')**

**def \_get\_attendees\_count(self):**

**for r in self:**

**r.attendees\_count = len(r.attendee\_ids)**

**@api.constrains('instructor\_id', 'attendee\_ids')**

**def \_check\_instructor\_not\_in\_attendees(self):**

**for r in self:**

openacademy/views/openacademy.xml

**</field>**

**</record>**

**<record model="ir.ui.view" id="openacademy\_session\_graph\_view">**

**<field name="name">openacademy.session.graph</field>**

**<field name="model">openacademy.session</field>**

**<field name="arch" type="xml">**

**<graph string="Participations by Courses">**

**<field name="course\_id"/>**

**<field name="attendees\_count" type="measure"/>**

**</graph>**

**</field>**

**</record>**

**<record model="ir.actions.act\_window" id="session\_list\_action">**

**<field name="name">Sessions</field>**

**<field name="res\_model">openacademy.session</field>**

**<field name="view\_type">form</field>**

**<field name="view\_mode">tree,form,calendar,gantt,graph</field>**

**</record>**

**<menuitem id="session\_menu" name="Sessions"**

### Kanban

Used to organize tasks, production processes, etc… their root element is **<kanban>**.

A kanban view shows a set of cards possibly grouped in columns. Each card represents a record, and each column the values of an aggregation field.

For instance, project tasks may be organized by stage (each column is a stage), or by responsible (each column is a user), and so on.

Kanban views define the structure of each card as a mix of form elements (including basic HTML) and **[QWeb](https://www.odoo.com/documentation/9.0/reference/qweb.html" \l "reference-qweb)**.

**Exercise**

Kanban view

Add a Kanban view that displays sessions grouped by course (columns are thus courses).

1. Add an integer **color** field to the Session model
2. Add the kanban view and update the action

openacademy/models.py

**duration = fields.Float(digits=(6, 2), help="Duration in days")**

**seats = fields.Integer(string="Number of seats")**

**active = fields.Boolean(default=True)**

**color = fields.Integer()**

**instructor\_id = fields.Many2one('res.partner', string="Instructor",**

**domain=['|', ('instructor', '=', True),**

openacademy/views/openacademy.xml

**</record>**

**<record model="ir.ui.view" id="view\_openacad\_session\_kanban">**

**<field name="name">openacad.session.kanban</field>**

**<field name="model">openacademy.session</field>**

**<field name="arch" type="xml">**

**<kanban default\_group\_by="course\_id">**

**<field name="color"/>**

**<templates>**

**<t t-name="kanban-box">**

**<div**

**t-attf-class="oe\_kanban\_color\_{{kanban\_getcolor(record.color.raw\_value)}}**

**oe\_kanban\_global\_click\_edit oe\_semantic\_html\_override**

**oe\_kanban\_card {{record.group\_fancy==1 ? 'oe\_kanban\_card\_fancy' : ''}}">**

**<div class="oe\_dropdown\_kanban">**

**<!-- dropdown menu -->**

**<div class="oe\_dropdown\_toggle">**

**<i class="fa fa-bars fa-lg"/>**

**<ul class="oe\_dropdown\_menu">**

**<li>**

**<a type="delete">Delete</a>**

**</li>**

**<li>**

**<ul class="oe\_kanban\_colorpicker"**

**data-field="color"/>**

**</li>**

**</ul>**

**</div>**

**<div class="oe\_clear"></div>**

**</div>**

**<div t-attf-class="oe\_kanban\_content">**

**<!-- title -->**

**Session name:**

**<field name="name"/>**

**<br/>**

**Start date:**

**<field name="start\_date"/>**

**<br/>**

**duration:**

**<field name="duration"/>**

**</div>**

**</div>**

**</t>**

**</templates>**

**</kanban>**

**</field>**

**</record>**

**<record model="ir.actions.act\_window" id="session\_list\_action">**

**<field name="name">Sessions</field>**

**<field name="res\_model">openacademy.session</field>**

**<field name="view\_type">form</field>**

**<field name="view\_mode">tree,form,calendar,gantt,graph,kanban</field>**

**</record>**

**<menuitem id="session\_menu" name="Sessions"**

**parent="openacademy\_menu"**

## Workflows

Workflows are models associated to business objects describing their dynamics. Workflows are also used to track processes that evolve over time.

**Exercise**

Almost a workflow

Add a **state** field to the Session model. It will be used to define a workflow-ish.

A sesion can have three possible states: Draft (default), Confirmed and Done.

In the session form, add a (read-only) field to visualize the state, and buttons to change it. The valid transitions are:

* Draft -> Confirmed
* Confirmed -> Draft
* Confirmed -> Done
* Done -> Draft

1. Add a new **state** field
2. Add state-transitioning methods, those can be called from view buttons to change the record's state
3. And add the relevant buttons to the session's form view

openacademy/models.py

**attendees\_count = fields.Integer(**

**string="Attendees count", compute='\_get\_attendees\_count', store=True)**

**state = fields.Selection([**

**('draft', "Draft"),**

**('confirmed', "Confirmed"),**

**('done', "Done"),**

**], default='draft')**

**@api.multi**

**def action\_draft(self):**

**self.state = 'draft'**

**@api.multi**

**def action\_confirm(self):**

**self.state = 'confirmed'**

**@api.multi**

**def action\_done(self):**

**self.state = 'done'**

**@api.depends('seats', 'attendee\_ids')**

**def \_taken\_seats(self):**

**for r in self:**

openacademy/views/openacademy.xml

**<field name="model">openacademy.session</field>**

**<field name="arch" type="xml">**

**<form string="Session Form">**

**<header>**

**<button name="action\_draft" type="object"**

**string="Reset to draft"**

**states="confirmed,done"/>**

**<button name="action\_confirm" type="object"**

**string="Confirm" states="draft"**

**class="oe\_highlight"/>**

**<button name="action\_done" type="object"**

**string="Mark as done" states="confirmed"**

**class="oe\_highlight"/>**

**<field name="state" widget="statusbar"/>**

**</header>**

**<sheet>**

**<group>**

**<group string="General">**

Workflows may be associated with any object in Odoo, and are entirely customizable. Workflows are used to structure and manage the lifecycles of business objects and documents, and define transitions, triggers, etc. with graphical tools. Workflows, activities (nodes or actions) and transitions (conditions) are declared as XML records, as usual. The tokens that navigate in workflows are called workitems.

**Warning**

A workflow associated with a model is only created when the model's records are created. Thus there is no workflow instance associated with session instances created before the workflow's definition

**Exercise**

Workflow

Replace the ad-hoc Session workflow by a real workflow. Transform the Session form view so its buttons call the workflow instead of the model's methods.

openacademy/\_\_openerp\_\_.py

**'templates.xml',**

**'views/openacademy.xml',**

**'views/partner.xml',**

**'views/session\_workflow.xml',**

**],**

**# only loaded in demonstration mode**

**'demo': [**

openacademy/models.py

**('draft', "Draft"),**

**('confirmed', "Confirmed"),**

**('done', "Done"),**

**])**

**@api.multi**

**def action\_draft(self):**

openacademy/views/openacademy.xml

**<field name="arch" type="xml">**

**<form string="Session Form">**

**<header>**

**<button name="draft" type="workflow"**

**string="Reset to draft"**

**states="confirmed,done"/>**

**<button name="confirm" type="workflow"**

**string="Confirm" states="draft"**

**class="oe\_highlight"/>**

**<button name="done" type="workflow"**

**string="Mark as done" states="confirmed"**

**class="oe\_highlight"/>**

**<field name="state" widget="statusbar"/>**

openacademy/views/session\_workflow.xml

**<openerp>**

**<data>**

**<record model="workflow" id="wkf\_session">**

**<field name="name">OpenAcademy sessions workflow</field>**

**<field name="osv">openacademy.session</field>**

**<field name="on\_create">True</field>**

**</record>**

**<record model="workflow.activity" id="draft">**

**<field name="name">Draft</field>**

**<field name="wkf\_id" ref="wkf\_session"/>**

**<field name="flow\_start" eval="True"/>**

**<field name="kind">function</field>**

**<field name="action">action\_draft()</field>**

**</record>**

**<record model="workflow.activity" id="confirmed">**

**<field name="name">Confirmed</field>**

**<field name="wkf\_id" ref="wkf\_session"/>**

**<field name="kind">function</field>**

**<field name="action">action\_confirm()</field>**

**</record>**

**<record model="workflow.activity" id="done">**

**<field name="name">Done</field>**

**<field name="wkf\_id" ref="wkf\_session"/>**

**<field name="kind">function</field>**

**<field name="action">action\_done()</field>**

**</record>**

**<record model="workflow.transition" id="session\_draft\_to\_confirmed">**

**<field name="act\_from" ref="draft"/>**

**<field name="act\_to" ref="confirmed"/>**

**<field name="signal">confirm</field>**

**</record>**

**<record model="workflow.transition" id="session\_confirmed\_to\_draft">**

**<field name="act\_from" ref="confirmed"/>**

**<field name="act\_to" ref="draft"/>**

**<field name="signal">draft</field>**

**</record>**

**<record model="workflow.transition" id="session\_done\_to\_draft">**

**<field name="act\_from" ref="done"/>**

**<field name="act\_to" ref="draft"/>**

**<field name="signal">draft</field>**

**</record>**

**<record model="workflow.transition" id="session\_confirmed\_to\_done">**

**<field name="act\_from" ref="confirmed"/>**

**<field name="act\_to" ref="done"/>**

**<field name="signal">done</field>**

**</record>**

**</data>**

**</openerp>**

**Tip**

In order to check if instances of the workflow are correctly created alongside sessions, go to **Settings ‣ Technical ‣ Workflows ‣ Instances**

**Exercise**

Automatic transitions

Automatically transition sessions from Draft to Confirmed when more than half the session's seats are reserved.

openacademy/views/session\_workflow.xml

**<field name="act\_to" ref="done"/>**

**<field name="signal">done</field>**

**</record>**

**<record model="workflow.transition" id="session\_auto\_confirm\_half\_filled">**

**<field name="act\_from" ref="draft"/>**

**<field name="act\_to" ref="confirmed"/>**

**<field name="condition">taken\_seats &gt; 50</field>**

**</record>**

**</data>**

**</openerp>**

**Exercise**

Server actions

Replace the Python methods for synchronizing session state by server actions.

Both the workflow and the server actions could have been created entirely from the UI.

openacademy/views/session\_workflow.xml

**<field name="on\_create">True</field>**

**</record>**

**<record model="ir.actions.server" id="set\_session\_to\_draft">**

**<field name="name">Set session to Draft</field>**

**<field name="model\_id" ref="model\_openacademy\_session"/>**

**<field name="code">**

**model.search([('id', 'in', context['active\_ids'])]).action\_draft()**

**</field>**

**</record>**

**<record model="workflow.activity" id="draft">**

**<field name="name">Draft</field>**

**<field name="wkf\_id" ref="wkf\_session"/>**

**<field name="flow\_start" eval="True"/>**

**<field name="kind">dummy</field>**

**<field name="action"></field>**

**<field name="action\_id" ref="set\_session\_to\_draft"/>**

**</record>**

**<record model="ir.actions.server" id="set\_session\_to\_confirmed">**

**<field name="name">Set session to Confirmed</field>**

**<field name="model\_id" ref="model\_openacademy\_session"/>**

**<field name="code">**

**model.search([('id', 'in', context['active\_ids'])]).action\_confirm()**

**</field>**

**</record>**

**<record model="workflow.activity" id="confirmed">**

**<field name="name">Confirmed</field>**

**<field name="wkf\_id" ref="wkf\_session"/>**

**<field name="kind">dummy</field>**

**<field name="action"></field>**

**<field name="action\_id" ref="set\_session\_to\_confirmed"/>**

**</record>**

**<record model="ir.actions.server" id="set\_session\_to\_done">**

**<field name="name">Set session to Done</field>**

**<field name="model\_id" ref="model\_openacademy\_session"/>**

**<field name="code">**

**model.search([('id', 'in', context['active\_ids'])]).action\_done()**

**</field>**

**</record>**

**<record model="workflow.activity" id="done">**

**<field name="name">Done</field>**

**<field name="wkf\_id" ref="wkf\_session"/>**

**<field name="kind">dummy</field>**

**<field name="action"></field>**

**<field name="action\_id" ref="set\_session\_to\_done"/>**

**</record>**

**<record model="workflow.transition" id="session\_draft\_to\_confirmed">**

## Security

Access control mechanisms must be configured to achieve a coherent security policy.

### Group-based access control mechanisms

Groups are created as normal records on the model **res.groups**, and granted menu access via menu definitions. However even without a menu, objects may still be accessible indirectly, so actual object-level permissions (read, write, create, unlink) must be defined for groups. They are usually inserted via CSV files inside modules. It is also possible to restrict access to specific fields on a view or object using the field's groups attribute.

### Access rights

Access rights are defined as records of the model **ir.model.access**. Each access right is associated to a model, a group (or no group for global access), and a set of permissions: read, write, create, unlink. Such access rights are usually created by a CSV file named after its model: **ir.model.access.csv**.

**id,name,model\_id/id,group\_id/id,perm\_read,perm\_write,perm\_create,perm\_unlink**

**access\_idea\_idea,idea.idea,model\_idea\_idea,base.group\_user,1,1,1,0**

**access\_idea\_vote,idea.vote,model\_idea\_vote,base.group\_user,1,1,1,0**

**Exercise**

Add access control through the OpenERP interface

Create a new user "John Smith". Then create a group "OpenAcademy / Session Read" with read access to the Session model.

1. Create a new user John Smith through **Settings ‣ Users ‣ Users**
2. Create a new group **session\_read** through **Settings ‣ Users ‣ Groups**, it should have read access on the Session model
3. Edit John Smith to make them a member of **session\_read**
4. Log in as John Smith to check the access rights are correct

**Exercise**

Add access control through data files in your module

Using data files,

* Create a group OpenAcademy / Manager with full access to all OpenAcademy models
* Make Session and Course readable by all users

1. Create a new file **openacademy/security/security.xml** to hold the OpenAcademy Manager group
2. Edit the file **openacademy/security/ir.model.access.csv** with the access rights to the models
3. Finally update **openacademy/\_\_openerp\_\_.py** to add the new data files to it

openacademy/\_\_openerp\_\_.py

**# always loaded**

**'data': [**

**'security/security.xml',**

**'security/ir.model.access.csv',**

**'templates.xml',**

**'views/openacademy.xml',**

**'views/partner.xml',**

openacademy/security/ir.model.access.csv

**id,name,model\_id/id,group\_id/id,perm\_read,perm\_write,perm\_create,perm\_unlink**

**course\_manager,course manager,model\_openacademy\_course,group\_manager,1,1,1,1**

**session\_manager,session manager,model\_openacademy\_session,group\_manager,1,1,1,1**

**course\_read\_all,course all,model\_openacademy\_course,,1,0,0,0**

**session\_read\_all,session all,model\_openacademy\_session,,1,0,0,0**

openacademy/security/security.xml

**<openerp>**

**<data>**

**<record id="group\_manager" model="res.groups">**

**<field name="name">OpenAcademy / Manager</field>**

**</record>**

**</data>**

**</openerp>**

### Record rules

A record rule restricts the access rights to a subset of records of the given model. A rule is a record of the model **ir.rule**, and is associated to a model, a number of groups (many2many field), permissions to which the restriction applies, and a domain. The domain specifies to which records the access rights are limited.

Here is an example of a rule that prevents the deletion of leads that are not in state **cancel**. Notice that the value of the field**groups** must follow the same convention as the method **[write()](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.models.Model.write)** of the ORM.

**<record id="delete\_cancelled\_only" model="ir.rule">**

**<field name="name">Only cancelled leads may be deleted</field>**

**<field name="model\_id" ref="crm.model\_crm\_lead"/>**

**<field name="groups" eval="[(4, ref('base.group\_sale\_manager'))]"/>**

**<field name="perm\_read" eval="0"/>**

**<field name="perm\_write" eval="0"/>**

**<field name="perm\_create" eval="0"/>**

**<field name="perm\_unlink" eval="1" />**

**<field name="domain\_force">[('state','=','cancel')]</field>**

**</record>**

**Exercise**

Record rule

Add a record rule for the model Course and the group "OpenAcademy / Manager", that restricts **write** and**unlink** accesses to the responsible of a course. If a course has no responsible, all users of the group must be able to modify it.

Create a new rule in **openacademy/security/security.xml**:

openacademy/security/security.xml

**<record id="group\_manager" model="res.groups">**

**<field name="name">OpenAcademy / Manager</field>**

**</record>**

**<record id="only\_responsible\_can\_modify" model="ir.rule">**

**<field name="name">Only Responsible can modify Course</field>**

**<field name="model\_id" ref="model\_openacademy\_course"/>**

**<field name="groups" eval="[(4, ref('openacademy.group\_manager'))]"/>**

**<field name="perm\_read" eval="0"/>**

**<field name="perm\_write" eval="1"/>**

**<field name="perm\_create" eval="0"/>**

**<field name="perm\_unlink" eval="1"/>**

**<field name="domain\_force">**

**['|', ('responsible\_id','=',False),**

**('responsible\_id','=',user.id)]**

**</field>**

**</record>**

**</data>**

**</openerp>**

## Wizards

Wizards describe interactive sessions with the user (or dialog boxes) through dynamic forms. A wizard is simply a model that extends the class **TransientModel** instead of **[Model](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.models.Model)**. The class **TransientModel** extends **[Model](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.models.Model)** and reuse all its existing mechanisms, with the following particularities:

* Wizard records are not meant to be persistent; they are automatically deleted from the database after a certain time. This is why they are called transient.
* Wizard models do not require explicit access rights: users have all permissions on wizard records.
* Wizard records may refer to regular records or wizard records through many2one fields, but regular records cannot refer to wizard records through a many2one field.

We want to create a wizard that allow users to create attendees for a particular session, or for a list of sessions at once.

**Exercise**

Define the wizard

Create a wizard model with a many2one relationship with the Session model and a many2many relationship with the Partner model.

Add a new file **openacademy/wizard.py**:

openacademy/\_\_init\_\_.py

**from . import controllers**

**from . import models**

**from . import partner**

**from . import wizard**

openacademy/wizard.py

**# -\*- coding: utf-8 -\*-**

**from openerp import models, fields, api**

**class Wizard(models.TransientModel):**

**\_name = 'openacademy.wizard'**

**session\_id = fields.Many2one('openacademy.session',**

**string="Session", required=True)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

### Launching wizards

Wizards are launched by **ir.actions.act\_window** records, with the field **target** set to the value **new**. The latter opens the wizard view into a popup window. The action may be triggered by a menu item.

There is another way to launch the wizard: using an **ir.actions.act\_window** record like above, but with an extra field **src\_model**that specifies in the context of which model the action is available. The wizard will appear in the contextual actions of the model, above the main view. Because of some internal hooks in the ORM, such an action is declared in XML with the tag **act\_window**.

**<act\_window id="launch\_the\_wizard"**

**name="Launch the Wizard"**

**src\_model="context.model.name"**

**res\_model="wizard.model.name"**

**view\_mode="form"**

**target="new"**

**key2="client\_action\_multi"/>**

Wizards use regular views and their buttons may use the attribute **special="cancel"** to close the wizard window without saving.

**Exercise**

Launch the wizard

1. Define a form view for the wizard.
2. Add the action to launch it in the context of the Session model.
3. Define a default value for the session field in the wizard; use the context parameter **self.\_context** to retrieve the current session.

openacademy/wizard.py

**class Wizard(models.TransientModel):**

**\_name = 'openacademy.wizard'**

**def \_default\_session(self):**

**return self.env['openacademy.session'].browse(self.\_context.get('active\_id'))**

**session\_id = fields.Many2one('openacademy.session',**

**string="Session", required=True, default=\_default\_session)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

openacademy/views/openacademy.xml

**parent="openacademy\_menu"**

**action="session\_list\_action"/>**

**<record model="ir.ui.view" id="wizard\_form\_view">**

**<field name="name">wizard.form</field>**

**<field name="model">openacademy.wizard</field>**

**<field name="arch" type="xml">**

**<form string="Add Attendees">**

**<group>**

**<field name="session\_id"/>**

**<field name="attendee\_ids"/>**

**</group>**

**</form>**

**</field>**

**</record>**

**<act\_window id="launch\_session\_wizard"**

**name="Add Attendees"**

**src\_model="openacademy.session"**

**res\_model="openacademy.wizard"**

**view\_mode="form"**

**target="new"**

**key2="client\_action\_multi"/>**

**</data>**

**</openerp>**

**Exercise**

Register attendees

Add buttons to the wizard, and implement the corresponding method for adding the attendees to the given session.

openacademy/views/openacademy.xml

**<field name="attendee\_ids"/>**

**</group>**

**<footer>**

**<button name="subscribe" type="object"**

**string="Subscribe" class="oe\_highlight"/>**

**or**

**<button special="cancel" string="Cancel"/>**

**</footer>**

**</form>**

**</field>**

**</record>**

openacademy/wizard.py

**session\_id = fields.Many2one('openacademy.session',**

**string="Session", required=True, default=\_default\_session)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

**@api.multi**

**def subscribe(self):**

**self.session\_id.attendee\_ids |= self.attendee\_ids**

**return {}**

**Exercise**

Register attendees to multiple sessions

Modify the wizard model so that attendees can be registered to multiple sessions.

openacademy/views/openacademy.xml

**<form string="Add Attendees">**

**<group>**

**<field name="session\_ids"/>**

**<field name="attendee\_ids"/>**

**</group>**

**<footer>**

**<button name="subscribe" type="object"**

openacademy/wizard.py

**class Wizard(models.TransientModel):**

**\_name = 'openacademy.wizard'**

**def \_default\_sessions(self):**

**return self.env['openacademy.session'].browse(self.\_context.get('active\_ids'))**

**session\_ids = fields.Many2many('openacademy.session',**

**string="Sessions", required=True, default=\_default\_sessions)**

**attendee\_ids = fields.Many2many('res.partner', string="Attendees")**

**@api.multi**

**def subscribe(self):**

**for session in self.session\_ids:**

**session.attendee\_ids |= self.attendee\_ids**

**return {}**

## Internationalization

Each module can provide its own translations within the i18n directory, by having files named LANG.po where LANG is the locale code for the language, or the language and country combination when they differ (e.g. pt.po or pt\_BR.po). Translations will be loaded automatically by Odoo for all enabled languages. Developers always use English when creating a module, then export the module terms using Odoo's gettext POT export feature (**Settings ‣ Translations ‣ Import/Export ‣ Export Translation** without specifying a language), to create the module template POT file, and then derive the translated PO files. Many IDE's have plugins or modes for editing and merging PO/POT files.

**Tip**

The Portable Object files generated by Odoo are published on [**Transifex**](https://www.transifex.com/odoo/public/), making it easy to translate the software.

**|- idea/ # The module directory**

**|- i18n/ # Translation files**

**| - idea.pot # Translation Template (exported from Odoo)**

**| - fr.po # French translation**

**| - pt\_BR.po # Brazilian Portuguese translation**

**| (...)**

**Tip**

By default Odoo's POT export only extracts labels inside XML files or inside field definitions in Python code, but any Python string can be translated this way by surrounding it with the function **openerp.\_()** (e.g.**\_("Label")**)

**Exercise**

Translate a module

Choose a second language for your Odoo installation. Translate your module using the facilities provided by Odoo.

1. Create a directory **openacademy/i18n/**
2. Install whichever language you want ( **Administration ‣ Translations ‣ Load an Official Translation**)
3. Synchronize translatable terms (**Administration ‣ Translations ‣ Application Terms ‣ Synchronize Translations**)
4. Create a template translation file by exporting ( **Administration ‣ Translations -> Import/Export ‣ Export Translation**) without specifying a language, save in **openacademy/i18n/**
5. Create a translation file by exporting ( **Administration ‣ Translations ‣ Import/Export ‣ Export Translation**) and specifying a language. Save it in **openacademy/i18n/**
6. Open the exported translation file (with a basic text editor or a dedicated PO-file editor e.g. [**POEdit**](http://poedit.net/) and translate the missing terms
7. In **models.py**, add an import statement for the function **openerp.\_** and mark missing strings as translatable
8. Repeat steps 3-6

openacademy/models.py

**# -\*- coding: utf-8 -\*-**

**from datetime import timedelta**

**from openerp import models, fields, api, exceptions, \_**

**class Course(models.Model):**

**\_name = 'openacademy.course'**

**default = dict(default or {})**

**copied\_count = self.search\_count(**

**[('name', '=like', \_(u"Copy of {}%").format(self.name))])**

**if not copied\_count:**

**new\_name = \_(u"Copy of {}").format(self.name)**

**else:**

**new\_name = \_(u"Copy of {} ({})").format(self.name, copied\_count)**

**default['name'] = new\_name**

**return super(Course, self).copy(default)**

**if self.seats < 0:**

**return {**

**'warning': {**

**'title': \_("Incorrect 'seats' value"),**

**'message': \_("The number of available seats may not be negative"),**

**},**

**}**

**if self.seats < len(self.attendee\_ids):**

**return {**

**'warning': {**

**'title': \_("Too many attendees"),**

**'message': \_("Increase seats or remove excess attendees"),**

**},**

**}**

**def \_check\_instructor\_not\_in\_attendees(self):**

**for r in self:**

**if r.instructor\_id and r.instructor\_id in r.attendee\_ids:**

**raise exceptions.ValidationError(\_("A session's instructor can't be an attendee"))**

## Reporting

### Printed reports

Odoo 8.0 comes with a new report engine based on **[QWeb](https://www.odoo.com/documentation/9.0/reference/qweb.html" \l "reference-qweb)**, [**Twitter Bootstrap**](http://getbootstrap.com/) and [**Wkhtmltopdf**](http://wkhtmltopdf.org/).

A report is a combination two elements:

* an **ir.actions.report.xml**, for which a **<report>** shortcut element is provided, it sets up various basic parameters for the report (default type, whether the report should be saved to the database after generation,…)

**<report**

**id="account\_invoices"**

**model="account.invoice"**

**string="Invoices"**

**report\_type="qweb-pdf"**

**name="account.report\_invoice"**

**file="account.report\_invoice"**

**attachment\_use="True"**

**attachment="(object.state in ('open','paid')) and**

**('INV'+(object.number or '').replace('/','')+'.pdf')"**

**/>**

* A standard **[QWeb view](https://www.odoo.com/documentation/9.0/reference/views.html" \l "reference-views-qweb)** for the actual report:

**<t t-call="report.html\_container">**

**<t t-foreach="docs" t-as="o">**

**<t t-call="report.external\_layout">**

**<div class="page">**

**<h2>Report title</h2>**

**</div>**

**</t>**

**</t>**

**</t>**

**the standard rendering context provides a number of elements, the most**

**important being:**

**``docs``**

**the records for which the report is printed**

**``user``**

**the user printing the report**

Because reports are standard web pages, they are available through a URL and output parameters can be manipulated through this URL, for instance the HTML version of the Invoice report is available through[**http://localhost:8069/report/html/account.report\_invoice/1**](http://localhost:8069/report/html/account.report_invoice/1) (if **account** is installed) and the PDF version through[**http://localhost:8069/report/pdf/account.report\_invoice/1**](http://localhost:8069/report/pdf/account.report_invoice/1).

**Danger**

If it appears that your PDF report is missing the styles (i.e. the text appears but the style/layout is different from the html version), probably your [**wkhtmltopdf**](http://wkhtmltopdf.org/) process cannot reach your web server to download them.

If you check your server logs and see that the CSS styles are not being downloaded when generating a PDF report, most surely this is the problem.

The [**wkhtmltopdf**](http://wkhtmltopdf.org/) process will use the **web.base.url** system parameter as the root path to all linked files, but this parameter is automatically updated each time the Administrator is logged in. If your server resides behind some kind of proxy, that could not be reachable. You can fix this by adding one of these system parameters:

* **report.url**, pointing to an URL reachable from your server (probably **http://localhost:8069** or something similar). It will be used for this particular purpose only.
* **web.base.url.freeze**, when set to **True**, will stop the automatic updates to **web.base.url**.

**Exercise**

Create a report for the Session model

For each session, it should display session's name, its start and end, and list the session's attendees.

openacademy/\_\_openerp\_\_.py

**'views/openacademy.xml',**

**'views/partner.xml',**

**'views/session\_workflow.xml',**

**'reports.xml',**

**],**

**# only loaded in demonstration mode**

**'demo': [**

openacademy/reports.xml

**<openerp>**

**<data>**

**<report**

**id="report\_session"**

**model="openacademy.session"**

**string="Session Report"**

**name="openacademy.report\_session\_view"**

**file="openacademy.report\_session"**

**report\_type="qweb-pdf" />**

**<template id="report\_session\_view">**

**<t t-call="report.html\_container">**

**<t t-foreach="docs" t-as="doc">**

**<t t-call="report.external\_layout">**

**<div class="page">**

**<h2 t-field="doc.name"/>**

**<p>From <span t-field="doc.start\_date"/> to <span t-field="doc.end\_date"/></p>**

**<h3>Attendees:</h3>**

**<ul>**

**<t t-foreach="doc.attendee\_ids" t-as="attendee">**

**<li><span t-field="attendee.name"/></li>**

**</t>**

**</ul>**

**</div>**

**</t>**

**</t>**

**</t>**

**</template>**

**</data>**

**</openerp>**

### Dashboards

**Exercise**

Define a Dashboard

Define a dashboard containing the graph view you created, the sessions calendar view and a list view of the courses (switchable to a form view). This dashboard should be available through a menuitem in the menu, and automatically displayed in the web client when the OpenAcademy main menu is selected.

1. Create a file **openacademy/views/session\_board.xml**. It should contain the board view, the actions referenced in that view, an action to open the dashboard and a re-definition of the main menu item to add the dashboard action

**Note**

Available dashboard styles are **1**, **1-1**, **1-2**, **2-1** and **1-1-1**

1. Update **openacademy/\_\_openerp\_\_.py** to reference the new data file

openacademy/\_\_openerp\_\_.py

**'version': '0.1',**

**# any module necessary for this one to work correctly**

**'depends': ['base', 'board'],**

**# always loaded**

**'data': [**

**'views/openacademy.xml',**

**'views/partner.xml',**

**'views/session\_workflow.xml',**

**'views/session\_board.xml',**

**'reports.xml',**

**],**

**# only loaded in demonstration mode**

openacademy/views/session\_board.xml

**<?xml version="1.0"?>**

**<openerp>**

**<data>**

**<record model="ir.actions.act\_window" id="act\_session\_graph">**

**<field name="name">Attendees by course</field>**

**<field name="res\_model">openacademy.session</field>**

**<field name="view\_type">form</field>**

**<field name="view\_mode">graph</field>**

**<field name="view\_id"**

**ref="openacademy.openacademy\_session\_graph\_view"/>**

**</record>**

**<record model="ir.actions.act\_window" id="act\_session\_calendar">**

**<field name="name">Sessions</field>**

**<field name="res\_model">openacademy.session</field>**

**<field name="view\_type">form</field>**

**<field name="view\_mode">calendar</field>**

**<field name="view\_id" ref="openacademy.session\_calendar\_view"/>**

**</record>**

**<record model="ir.actions.act\_window" id="act\_course\_list">**

**<field name="name">Courses</field>**

**<field name="res\_model">openacademy.course</field>**

**<field name="view\_type">form</field>**

**<field name="view\_mode">tree,form</field>**

**</record>**

**<record model="ir.ui.view" id="board\_session\_form">**

**<field name="name">Session Dashboard Form</field>**

**<field name="model">board.board</field>**

**<field name="type">form</field>**

**<field name="arch" type="xml">**

**<form string="Session Dashboard">**

**<board style="2-1">**

**<column>**

**<action**

**string="Attendees by course"**

**name="%(act\_session\_graph)d"**

**height="150"**

**width="510"/>**

**<action**

**string="Sessions"**

**name="%(act\_session\_calendar)d"/>**

**</column>**

**<column>**

**<action**

**string="Courses"**

**name="%(act\_course\_list)d"/>**

**</column>**

**</board>**

**</form>**

**</field>**

**</record>**

**<record model="ir.actions.act\_window" id="open\_board\_session">**

**<field name="name">Session Dashboard</field>**

**<field name="res\_model">board.board</field>**

**<field name="view\_type">form</field>**

**<field name="view\_mode">form</field>**

**<field name="usage">menu</field>**

**<field name="view\_id" ref="board\_session\_form"/>**

**</record>**

**<menuitem**

**name="Session Dashboard" parent="base.menu\_reporting\_dashboard"**

**action="open\_board\_session"**

**sequence="1"**

**id="menu\_board\_session" icon="terp-graph"/>**

**</data>**

**</openerp>**

## WebServices

The web-service module offer a common interface for all web-services :

* XML-RPC
* JSON-RPC

Business objects can also be accessed via the distributed object mechanism. They can all be modified via the client interface with contextual views.

Odoo is accessible through XML-RPC/JSON-RPC interfaces, for which libraries exist in many languages.

### XML-RPC Library

The following example is a Python program that interacts with an Odoo server with the library **xmlrpclib**:

**import xmlrpclib**

**root = 'http://%s:%d/xmlrpc/' % (HOST, PORT)**

**uid = xmlrpclib.ServerProxy(root + 'common').login(DB, USER, PASS)**

**print "Logged in as %s (uid: %d)" % (USER, uid)**

**# Create a new note**

**sock = xmlrpclib.ServerProxy(root + 'object')**

**args = {**

**'color' : 8,**

**'memo' : 'This is a note',**

**'create\_uid': uid,**

**}**

**note\_id = sock.execute(DB, uid, PASS, 'note.note', 'create', args)**

**Exercise**

Add a new service to the client

Write a Python program able to send XML-RPC requests to a PC running Odoo (yours, or your instructor's). This program should display all the sessions, and their corresponding number of seats. It should also create a new session for one of the courses.

**import functools**

**import xmlrpclib**

**HOST = 'localhost'**

**PORT = 8069**

**DB = 'openacademy'**

**USER = 'admin'**

**PASS = 'admin'**

**ROOT = 'http://%s:%d/xmlrpc/' % (HOST,PORT)**

**# 1. Login**

**uid = xmlrpclib.ServerProxy(ROOT + 'common').login(DB,USER,PASS)**

**print "Logged in as %s (uid:%d)" % (USER,uid)**

**call = functools.partial(**

**xmlrpclib.ServerProxy(ROOT + 'object').execute,**

**DB, uid, PASS)**

**# 2. Read the sessions**

**sessions = call('openacademy.session','search\_read', [], ['name','seats'])**

**for session in sessions:**

**print "Session %s (%s seats)" % (session['name'], session['seats'])**

**# 3.create a new session**

**session\_id = call('openacademy.session', 'create', {**

**'name' : 'My session',**

**'course\_id' : 2,**

**})**

Instead of using a hard-coded course id, the code can look up a course by name:

**# 3.create a new session for the "Functional" course**

**course\_id = call('openacademy.course', 'search', [('name','ilike','Functional')])[0]**

**session\_id = call('openacademy.session', 'create', {**

**'name' : 'My session',**

**'course\_id' : course\_id,**

**})**

### JSON-RPC Library

The following example is a Python program that interacts with an Odoo server with the standard Python libraries **urllib2** and**json**:

**import json**

**import random**

**import urllib2**

**def json\_rpc(url, method, params):**

**data = {**

**"jsonrpc": "2.0",**

**"method": method,**

**"params": params,**

**"id": random.randint(0, 1000000000),**

**}**

**req = urllib2.Request(url=url, data=json.dumps(data), headers={**

**"Content-Type":"application/json",**

**})**

**reply = json.load(urllib2.urlopen(req))**

**if reply.get("error"):**

**raise Exception(reply["error"])**

**return reply["result"]**

**def call(url, service, method, \*args):**

**return json\_rpc(url, "call", {"service": service, "method": method, "args": args})**

**# log in the given database**

**url = "http://%s:%s/jsonrpc" % (HOST, PORT)**

**uid = call(url, "common", "login", DB, USER, PASS)**

**# create a new note**

**args = {**

**'color' : 8,**

**'memo' : 'This is another note',**

**'create\_uid': uid,**

**}**

**note\_id = call(url, "object", "execute", DB, uid, PASS, 'note.note', 'create', args)**

Here is the same program, using the library [**jsonrpclib**](https://pypi.python.org/pypi/jsonrpclib):

**import jsonrpclib**

**# server proxy object**

**url = "http://%s:%s/jsonrpc" % (HOST, PORT)**

**server = jsonrpclib.Server(url)**

**# log in the given database**

**uid = server.call(service="common", method="login", args=[DB, USER, PASS])**

**# helper function for invoking model methods**

**def invoke(model, method, \*args):**

**args = [DB, uid, PASS, model, method] + list(args)**

**return server.call(service="object", method="execute", args=args)**

**# create a new note**

**args = {**

**'color' : 8,**

**'memo' : 'This is another note',**

**'create\_uid': uid,**

**}**

**note\_id = invoke('note.note', 'create', args)**

Examples can be easily adapted from XML-RPC to JSON-RPC.

**Note**

There are a number of high-level APIs in various languages to access Odoo systems without explicitly going through XML-RPC or JSON-RPC, such as:

* [**https://github.com/akretion/ooor**](https://github.com/akretion/ooor)
* [**https://github.com/syleam/openobject-library**](https://github.com/syleam/openobject-library)
* [**https://github.com/nicolas-van/openerp-client-lib**](https://github.com/nicolas-van/openerp-client-lib)
* [**https://pypi.python.org/pypi/oersted/**](https://pypi.python.org/pypi/oersted/)
* [**https://github.com/abhishek-jaiswal/php-openerp-lib**](https://github.com/abhishek-jaiswal/php-openerp-lib)

**[[1]](https://www.odoo.com/documentation/9.0/howtos/backend.html" \l "id2)** it is possible to **[disable the automatic creation of some fields](https://www.odoo.com/documentation/9.0/reference/orm.html" \l "openerp.models.Model._log_access)**

**[[2]](https://www.odoo.com/documentation/9.0/howtos/backend.html" \l "id1) writing raw SQL queries is possible, but requires care as it bypasses all Odoo authentication and security mechanisms.**